UNIVERSITY OF CALIFORNIA

Los Angeles

Online Learning for Energy-Efficient Multimedia Systems

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Electrical Engineering

by

Nicholas Howe Mastronarde

2011

The dissertation of Nicholas Howe Mastronarde is approved.

_____

Abeer Alwan

_____

Paulo Tabuada

_____

Jenn Vaughan

_____

Mihaela van der Schaar, Committee Chair

University of California, Los Angeles

2011

To my wife.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Mihaela van der Schaar, for her unending enthusiasm for research, as well as her support and mentorship throughout my entire graduate studies. I am lucky to have met her during my last quarter as an undergraduate at UC Davis and thankful that she convinced me to follow her to UCLA to do a PhD.

I would like to thank Professors Abeer Alwan, Paulo Tabuada, and Jennifer Wortman Vaughan, for taking time out of their busy schedules to serve on my dissertation committee. I would also like to thank Professor Rupak Majumdar for serving on my dissertation committee for my qualifying exam.

I would be remiss not to express my sincere gratitude to all of the amazing colleagues I have met in the Multimedia Communications and Systems Laboratory at UCLA. I would especially like to thank Dr. Yiannis Andreoupoulos, Dr. Fangwen Fu, Dr. Yi Su, Dr. Brian Foo, Dr. Hsien-Po Shiang, and Dr. Hyunggon Park whose friendship and support have been invaluable. I would also like to thank my supervisors and mentors Dr. Deepak Turaga and Dr. Octavian Udrea at IBM T.J. Watson Research Center, and Dr. Hong Jiang at Intel Corporation for the opportunity to do interesting work during summer internships.

I would like acknowledge that Chapter 5 is a version of a paper in progress that is co-authored by Francesco Verde, Donatella Darsena, and Anna Scaglione.

I am incredibly grateful for the love and support of my wife who moved away from home to be with me in Los Angeles, stayed up many late nights with me while I worked, and was always there when I needed her most. Lastly, I would like to thank my family for always being a phone call away.

VITA

February 21, 1983	Born, Oakland, California
2005	B.S. Electrical Engineering University of California, Davis Davis, California
2006	M.S. Electrical Engineering University of California, Davis Davis, California
2007	Internship Intel Corporation Folsom, California
2009-2010	Teaching Assistant Department of Electrical Engineering Los Angeles, California
2010	Internship IBM Research Hawthorne, New York

PUBLICATIONS

N. Mastronarde, Y. Andreopoulos, M. van der Schaar, D. Krishnaswamy and J. Vicente, "Cross-layer video streaming over 802.11e-enabled wireless mesh networks," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP),* vol. 5, pp. V-433- V-436, May 14-19, 2006.

N. Mastronarde, D. S. Turaga, and M. van der Schaar, "Collaborative resource management for video over wireless multi-hop mesh networks," *IEEE International Conference on Image Processing* (*ICIP*), pp. 1297-1300, Oct. 8-11, 2006.

Y. Andreopoulos, N. Mastronarde, and M. van der Schaar, "Cross-layer optimized video streaming over wireless multi-hop mesh networks," *IEEE J. on Select. Areas in Communications Multi-Hop Wireless Mesh Networks*, vol. 24, no. 11, pp. 2104-2115, Nov. 2006.

N. Mastronarde, D. S. Turaga, and M. van der Schaar. “Collaborative resource exchanges for peer-to-peer video streaming over wireless mesh networks,” *IEEE J. on Select. Areas in Communications Peer-to-peer Communications and Applications*, vol. 25, no. 1, pp. 108-118, Jan. 2007.

N. Mastronarde and M. van der Schaar, "A queuing-theoretic approach to task scheduling and processor selection for video decoding applications," *IEEE Trans. Multimedia*, vol. 8, no. 7, pp. 1493-1507, Nov. 2007.

N. Mastronarde and M. van der Schaar, “A bargaining theoretic approach to quality-fair system resource allocation for multiple decoding tasks,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 18, no. 3, Mar. 2008.

N. Mastronarde and M. van der Schaar, “A scalable complexity specification for video applications,” *IEEE International Conference on Image Processing* (*ICIP*), pp. 2576-2579, Oct. 12-15, 2008.

N. Mastronarde and M. van der Schaar, “Automated bidding for media services at the edge of a content delivery network,” *IEEE Trans. on Multimedia,* vol. 11, no. 3, pp. 543-555, Apr. 2009.

N. Mastronarde and M. van der Schaar, “Towards a General Framework for Cross-Layer Decision Making in Multimedia Systems,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 19, no. 5, pp. 719-732, May 2009.

N. Mastronarde and M. van der Schaar, “Designing autonomous layered video coders,” *Elsevier Journal Signal Processing: Image Communication – Special Issue on Scalable Coded Media Beyond Compression*, vol. 24, no. 6, pp. 417-436, July 2009.

N. Mastronarde and M. van der Schaar, “Autonomous decision making in layered and reconfigurable video coders,” *Asilomar Conference on Signals, Systems, and Computers*, pp. 553-557, Nov. 1-4, 2009.

N. Mastronarde and M. van der Schaar, “Online reinforcement learning for dynamic multimedia systems,” *IEEE Trans. on Image Processing*, vol. 19, no. 2, pp. 290-305, Feb. 2010.

N. Mastronarde and M. van der Schaar, “Online layered learning for cross-layer optimization of dynamic multimedia systems,” *ACM Multimedia Systems*, pp. 47-58, Feb. 22-23, 2010.

N. Mastronarde and M. van der Schaar, "Online reinforcement learning for multimedia buffer control," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1958-1961, Mar. 14-19, 2010.

N. Mastronarde and M. van der Schaar, "A new approach to cross-layer optimization of multimedia systems," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 2310-2313, Mar. 14-19, 2010.

N. Mastronarde, M. van der Schaar, A. Scaglione, F. Verde, and D. Darsena, "Sailing good radio waves and transmitting important bits: relay cooperation in wireless video transmission," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 5566-5569, Mar. 14-19, 2010.

N. Changuel, N. Mastronarde, M. van der Schaar, B. Sayadi, and M. Kieffer, "End-to-end stochastic scheduling of scalable video over time varying channels," *Proc. ACM Multimedia*, Oct. 25-29, 2010.

ABSTRACT OF THE DISSERTATION

Online Learning for Energy-Efficient Multimedia Systems

by

Nicholas Howe Mastronarde

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2011

Professor Mihaela van der Schaar, Chair

Designing energy-efficient resource management strategies for multimedia applications is a challenging problem because of their stringent delay constraints, mixed priorities, intense resource requirements, and sophisticated source coding dependency structures. In my dissertation research, I aim to develop a rigorous, formal, and unified modeling framework for supporting such applications in a large class of resource management scenarios including energy-efficient point-to-point wireless communication, cooperative multi-user wireless video transmission, and energy-efficient cross-layer system optimization.

The foundation of my approach is modeling multimedia systems as *stochastic* and *dynamic* systems. Unlike traditional resource management solutions, in which the goal is

to optimize the immediate utility (i.e. myopic optimization), the goal in the proposed framework is to optimize the *trajectory* of the system's underlying stochastic process by accounting for how decisions at the current time impact the future utility (i.e. dynamic optimization). To achieve this, I model system and network resource management problems as Markov decision processes (MDPs). Then, to address the fact that the statistics of the system's underlying stochastic process are typically unknown a priori, I adopt and often extend reinforcement learning techniques from artificial intelligence to enable devices to learn their experienced dynamics online, at run-time, in order to optimize their long-term performance.

Chapter 1

Introduction

1.1 Motivation

State-of-the-art multimedia technology is poised to enable widespread proliferation of a variety of life-enhancing applications, such as video conferencing, emergency services, surveillance, telemedicine, remote teaching and training, augmented reality, and distributed gaming. However, efficiently designing and implementing such delay-sensitive multimedia applications on energy-constrained, heterogeneous devices and systems is challenging due to their real-time constraints and high workload complexity, as well as the time-varying environmental dynamics experienced by the system (e.g. video source characteristics, user requirements, workload characteristics, number of running applications, memory/cache behavior, channel conditions, etc.). To address these problems, novel system and network resource management techniques are required.

1.2 Challenges

The majority of existing system and network resource management solutions rely on *myopic* optimizations [1]-[10], which ignore the impact of resource management decisions made at the current time on the system's future performance and ignore the dynamics in the system (e.g. application workload and traffic characteristics, memory and

cache behavior, and network conditions). However, in almost all resource management problems, the decisions made at the current time impact the system's immediate and future performance, and acting on accurate *predictions or forecasts* about the future dynamics can improve resource utilization, energy-efficiency, and application quality. For example, in a real-time system, the time required processing a task impacts the time available, and the power required, to process future tasks before their deadlines. Therefore, in order to minimize the total energy used to process a set of tasks, it is necessary to adapt the processor speed based not only on the immediate task's resource requirements, but also on the resource requirements of future tasks. In this dissertation, we interchangeably refer to this type of optimization as a *long-term*, *dynamic*, or *foresighted* optimization.

Reaping the full benefits of dynamic resource management hinges on our ability to quickly and accurately model the system's dynamics, which are typically *stochastic* and *unknown a priori*. The majority of existing resource management solutions cope with unknown dynamics by either (i) ignoring them, and deploying suboptimal heuristics [11] [12]; (ii) deploying learning and estimation solutions such as maximum likelihood estimation, statistical fitting, and regression techniques, which work well for estimating model parameters and distributions (e.g. workload distributions), but do not provide computationally tractable methods for performing dynamic optimization [2] [4] [6] [13]-[16]; or, (iii) deploying supervised learning techniques, which require time-consuming and domain-specific offline training, to predict the optimal resource management actions [13] [17].

In summary, a new paradigm for system and network resource management is necessary, where devices can autonomously learn online the unknown dynamics that they experience, and dynamically manage resources to obtain long-term optimal resource utilization, energy-efficiency, and application quality.

1.3 Contributions of the Dissertation

In this dissertation, we aim to develop a rigorous, formal, and unified energy-efficient resource management framework for supporting heterogeneous multimedia applications (e.g. loss-tolerant applications with different delay-constraints, priorities, and source coding dependency structures) in a large class of scenarios including energy-efficient point-to-point wireless communication, multi-user cooperative wireless communication, and energy-efficient cross-layer system optimization. To address the abovementioned challenges, we propose to view the multimedia system as a *stochastic* and *dynamic* system. Unlike traditional resource management solutions, in which the goal is to optimize the immediate performance, the goal in the proposed framework is to optimize the *trajectory* of the system's underlying stochastic process by accounting for how decisions at the current time impact the system's future behavior. To achieve this, we model system and network resource management problems as Markov decision processes (MDPs). Then, to address the fact that the statistics of the system's underlying stochastic process are only partially known or are entirely unknown a priori, we adopt and often extend reinforcement learning techniques from artificial intelligence to enable devices to autonomously learn their experienced dynamics online, at run-time, in order to optimize

their long-term performance.

A unique and distinguishing feature of this dissertation is the extent of multimedia specific modeling in developing the proposed learning and dynamic optimization framework. This is in contrast to much of the research in stochastic control [18]-[21] and artificial intelligence [22] [23], which uses utility functions that generally reflect utility as a simple function of resource consumption, and does not consider the application and system specific dynamics. At the same time, however, the foundations and principles developed in my research are not limited to multimedia systems and, in fact, can be extended to many other settings including real-time systems, embedded systems, data centers, multi-core systems, multiprocessor systems-on-chips, sensor networks, wireless mesh networks, wired networks etc.

The following sections summarize the remainder of this dissertation.

1.4 Chapter 2 and Chapter 3: System Resource Management

In Chapter 2 of this dissertation, we introduce a rigorous methodology for dynamic cross-layer system optimization. To illustrate the proposed solution, we consider the problem of jointly optimizing the video encoding parameters at the application layer, resource allocation decision at the operating system layer, and processor speed at the hardware layer of a multimedia system. We introduce a centralized formulation of the dynamic cross-layer optimization, which requires a single system layer to collect information from all of the layers (e.g. the operating system layer, application layer, or a middleware layer), solve a complex joint optimization, and then dictate to the layers the optimal actions to

perform. This is similar to conventional myopic cross-layer multimedia system solutions because it requires a centralized optimizer. We then show that the proposed centralized cross-layer optimization can be decomposed into sub-problems for each layer. These sub problems can be solved layer by layer, which preserves the layered architecture by maintaining a separation among the layers' decision processes, designs, and implementations.

In Chapter 2, the dynamic cross-layer optimization is solved offline, under the assumption that the multimedia system's probabilistic dynamics are *known* a priori. In practice, however, these dynamics are *unknown* a priori and therefore must be learned online. To address this challenge, in Chapter 3, we use the decomposition introduced in Chapter 2 to enable the multimedia system layers to learn, through repeated interactions with each other, to autonomously optimize the system's long-term performance at run-time. Together, Chapters 2 and 3 illustrate how dynamic cross-layer optimization can dramatically improve system resource utilization, energy-efficiency, and application utility even when layers initially have no information about their experienced dynamics or how they impact and are impacted by the layers they interact with.

1.5 Chapter 4 and Chapter 5: Wireless Resource Management

While Chapters 2 and 3 focus on energy-efficient system management in systems, Chapters 4 and 5 focus on resource management in wireless networks. In Chapter 4, we address the problem of energy efficient point-to-point transmission of delay-sensitive data over a fading channel. We utilize three power saving techniques -- namely power-control,

adaptive modulation and coding, and dynamic power management -- that are widely used in the literature, but have never been combined into a unified framework based on dynamic optimization and online learning. The advantages of the proposed online method are that (i) it does not require a priori knowledge of the traffic arrival and channel statistics to determine the jointly optimal power-control, adaptive modulation and coding, and dynamic power management policies; (ii) it exploits partial information about the system so that less information needs to be learned than when using conventional reinforcement learning algorithms; and (iii) it obviates the need for action exploration, which severely limits the adaptation speed and run-time performance of conventional reinforcement learning algorithms.

While Chapter 4 addresses the problem of energy-efficient point-to-point wireless communications, Chapter 5 explores how to jointly optimize the resource allocation and scheduling in a multi-user wireless video transmission scenario with cooperative relays. The idea behind cooperation is that feeble signals of nodes that are located far away from the access point can be enhanced via the cooperation of intermediate nodes that act as cooperative relays. However, cooperative throughput gains alone do not translate to good video quality at the application layer, so we systematically formulate the wireless video transmission problem as a dynamic optimization that explicitly considers the video users' heterogeneous traffic characteristics, the dynamically varying network conditions, and the coupling among the users' transmission strategies across time due to the shared wireless resource. The main analytical result in Chapter 5 is that the optimal selection of cooperative relays can be done myopically without loss of optimality. This result is

important because it allows us to separate decisions that the wireless user should make in dynamic manner, i.e. packet scheduling and resource allocation decisions, from those that it can make myopically without loss of optimality, i.e. recruiting cooperative relays. Without this result, the dynamic optimization would be computationally intractable.

1.6 Chapter 6: Conclusion

Chapter 6 concludes the dissertation and includes a discussion about future research directions.

References

[1] 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.

[2] Z. He, W. Cheng, X. Chen, "Energy minimization of portable video communication devices based on power-rate-distortion optimization," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 18, no. 5, May 2008.

[3] 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.

[4] 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.

[5] K. Nahstedt, W. Yuan, S. Shah, Y. Xue, and K. Chen, "QoS support in multimedia wireless environments," in Multimedia Over IP and Wireless Networks, ed. M. van der Schaar and P. Chou, Academic Press, 2007.

[6] 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.

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

[8] M. van der Schaar, S. Krishnamachari, S. Choi, and X. Xu, “Adaptive cross-layer protection strategies for robust scalable video transmission over 802.11 WLANs,” *IEEE JSAC*, vol. 21, no. 10, pp. 1752-1763.

[9] M. van der Schaar, Y. Andreopoulos, and Z. Hu, “Optimized scalable video streaming over 802.11 a/e HCCA wireless networks under delay constraints,” *IEEE Trans. on Mobile Computing*, vol. 5, no. 6, pp. 755-768, June 2006.

[10] M. Chiang, S. H. Low, A. R. Caldbank, and J.C. Doyle, “Layering as optimization decomposition: A mathematical theory of network architectures,” *Proc. of IEEE*, vol. 95, no. 1, 2007.

[11] M. J. Neely, "Energy Optimal Control for Time Varying Wireless Networks", *IEEE Trans. On Information Theory*, vol. 52, no. 7, pp. 2915-2934, July 2006.

[12] R. Berry and R. G. Gallager, “Communications over fading channels with delay constraints,” *IEEE Trans. Inf. Theory*, vol 48, no. 5, pp. 1135-1149, May 2002.

[13] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli, “Dynamic power management for nonstationary service requests,” *IEEE Trans. on Computers*, vol. 51, no. 11, Nov. 2002.

[14] Y. Andreopoulos and M. van der Schaar, "Adaptive Linear Prediction for Resource Estimation of Video Decoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, pp. 751-764, June 2007.

[15] J. Liang, K. Nahrstedt, and Y. Zhou, “Adaptive multi-resource prediction in distributed resource sharing environment,” *Proc. IEEE Internat. Sympos. On Cluster Comput. and the Grid*, CCGrid-04, pp. 293-300,April 2004.

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

[17] M. van der Schaar, D. Turaga, and R. Wong, "Classification-Based System For Cross-Layer Optimized Wireless Video Transmission," *IEEE Trans. Multimedia*, vol. 8, no. 5, pp. 1082-1095, Oct. 2006

[18] N. Salodkar, A. Karandikar, V. S. Borkar, “A stable online algorithm for energy-efficient multiuser scheduling,” IEEE Trans. on Mobile Computing, vol. 9, no. 10, Oct. 2010.

[19] N. Salodkar, A. Bhorkar, A. Karandikar, V. S. Borkar, “An on-line learning algorithm for energy efficient delay constrained scheduling over a fading channel,” *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 732-742, Apr. 2008.

[20] M. H. Ngo and V. Krishnamurthy, “Monotonocity of constrained optimal transmission policies in correlated fading channels with ARQ,” *IEEE Trans. on Signal Processing*, vol. 58, No. 1, pp. 438-451, Jan. 2010.

[21] D. V. Djonin and V. Krishnamurthy, “MIMO transmission control in fading channels – a constrained Markov decision process formulation with monotone randomized policies,” *IEEE Trans. on Signal Processing*, vol. 55, no. 10, Oct. 2007.

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

[23] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: a survey,” *Journal of Artificial Intelligence Research 4*, pp. 237-285, May 2005.

Chapter 2

Cross-Layer Multimedia System Optimization

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 chapter, 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.

2.1 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 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 chapter, 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 chapter 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 chapter, 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 chapter is organized as follows. In Section 2.2, we define all of the parameters and concepts in our framework and present the objective of the foresighted cross-layer optimization problem. In Section 2.3, we provide concrete examples of the abstract concepts introduced in Section 2.2 for an illustrative cross-layer video encoding problem similar to those explored in [5]-[8]. In Section 2.4, we describe a centralized cross-layer optimization framework for maximizing the objective function introduced in Section 2.2, and we discuss the framework's limitations. In Section 2.5, we propose a layered optimization framework for maximizing the same objective. In Section 2.6, we present our experimental results based on the example system described in Section 2.3. Finally, we conclude in Section 2.7.

2.2 Cross-Layer Problem Statement

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

system optimization problem.

2.2.1 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 chapter, 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 chapter 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.

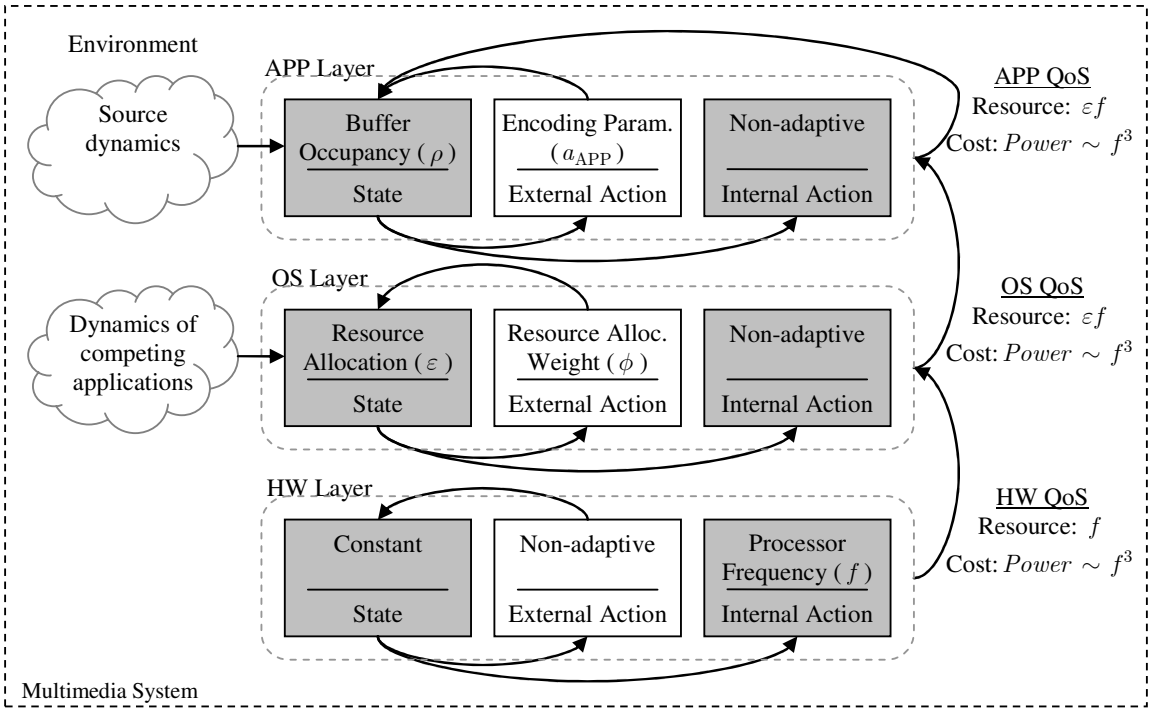


Figure 2.1. Layered decision making process illustrating the intra- and inter-layer dependencies in the system. System layers adapt to different components of the dynamic environment by deploying different "actions" at each layer.

Figure 2.1 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 Figure 2.1 in order to make the abstract concepts discussed throughout this section more concrete.

2.2.2 States

In this chapter, the state encapsulates all of the information relevant to making processing

decisions. Illustrative examples of layer states are provided in Section 2.3 and are included in Figure 2.1. 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 Figure 2.1), 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 $\boldsymbol{\mathcal{S}}$.

2.2.3 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 2.3.2 and 2.3.3 for examples of external actions at the OS and APP layers, which are also illustrated in Figure 2.1.) 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 2.2.6 for our definition of QoS, and Section 2.3.1 and Figure 2.1 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

[1] We investigate the validity of this assumption for a realistic multimedia system in Section 2.6.3.

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 $\boldsymbol{\xi} = [\xi_1,\ldots,\xi_L] \in \boldsymbol{\mathcal{X}} = \times_{l=1}^{L} \mathcal{X}_l$.

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 2.3 and in Figure 2.1.

2.2.4 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

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = \prod_{l=1}^{L} p(s'_l \mid \boldsymbol{s}'_{1\to l\text{-}1}, \boldsymbol{s}, \boldsymbol{\xi}), \tag{2.1}$$

where $\boldsymbol{s}'_{1\to l} = [s'_1, \ldots, s'_l]$ and $\boldsymbol{s}'_{1\to 0} = \mathbf{0}$ is an empty state. In this chapter, we assume that the next state 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 (2.1) as

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = \prod_{l=1}^{L} p(s'_l \mid \boldsymbol{s}, \boldsymbol{\xi}) \tag{2.2}$$

Second, due to the layered architecture and the definitions of the internal and external actions as illustrated in Figure 2.1, (2.2) can be factored further as follows:

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}) = p(s'_L \mid \boldsymbol{s}, a_L, \boldsymbol{b}) \prod_{l=1}^{L-1} p(s'_l \mid s_l, a_l). \tag{2.3}$$

We would like to make the following remarks about (2.3)**:**

- The term $\prod_{l=1}^{L-1} p(s'_l \mid s_l, a_l)$ in (2.3) is taken from the first $L-1$ layers in (2.2). 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 (2.2). The state transitions at the HW and OS layers illustrated in Figure 2.1 reflect this model.
- The term $p(s'_L \mid \boldsymbol{s}, a_L, \boldsymbol{b})$ in (2.3) is taken from layer L in (2.2). 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 Figure 2.1, except that the states and internal actions at the lower layers are abstracted by the QoS values, which we discuss in Section 2.2.6.

- If the l th 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 l th 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 2.3.

2.2.5 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 2.3 (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 2.3), i.e.

$$R(\boldsymbol{s}, \boldsymbol{\xi}) = g(\boldsymbol{s}, \boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^a \alpha_l(s_l, a_l) - \sum_{l=1}^{L} \omega_l^b \beta_l(s_l, b_l) \tag{2.4}$$

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. (2.4) 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

$$R_{\text{in}}(\boldsymbol{s},\boldsymbol{b}) = g(\boldsymbol{s},\boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^b \beta_l(s_l, b_l), \tag{2.5}$$

and the external reward is

$$R_{\text{ex}}(\boldsymbol{s},\boldsymbol{a}) = -\sum_{l=1}^{L} \omega_l^a \alpha_l(s_l, a_l). \tag{2.6}$$

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

2.2.6 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 chapter, 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

[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 performance across all layers. In

recursively computed given the QoS at layer $l-1$:

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

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 2.3.1, 2.3.2, and 2.3.3 we define illustrative QoS pairs for the HW, OS, and APP layers, respectively. These are also illustrated in Figure 2.1.

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 (2.3) as

$$p(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b}) = p(s_L' \mid s_L, a_L, Q_L), \tag{2.7}$$

and its internal reward function $R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b})$ in (2.5) as

$$R_{\text{in}}(\boldsymbol{s}, \boldsymbol{b}) = R_{\text{in}}(s_L, Q_L). \tag{2.8}$$

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 (2.7) and (2.8) are required

---

contrast, the external cost is determined by each individual layer when it selects its external action, which does not impact the immediate reward at the APP layer. Therefore, the external costs do not need to be included in the QoS.

for the layered optimization solution proposed in Section 2.5.

2.2.7 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.

$$E\left\{\sum_{n=n_0}^{\infty}(\gamma)^{n-n_0}R(\boldsymbol{s}^n,\boldsymbol{\xi}^n \mid \boldsymbol{s}^{n_0})\right\}, \tag{2.9}$$

where the parameter γ $(0 \leq \gamma < 1)$ 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 (2.9) 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 2.6.2, we discuss the impact of the discount factor on system's performance. In Section 2.4, we describe how to maximize (2.9) using a centralized MDP. Then, in Section 2.5, we present an alternative solution to the same problem based on a layered MDP.

2.3 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 chapter, 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 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.

2.3.1 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]:

$$\beta_{\mathrm{HW}}(s_{\mathrm{APP}}, a_{\mathrm{APP}}) = f^3 . \tag{2.10}$$

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 Figure 2.1.

2.3.2 OS Layer Examples

OS state: We denote the OS state by $s_{\mathrm{OS}} \in \boldsymbol{\mathcal{S}}_{\mathrm{OS}}$. 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 $s_{\mathrm{OS}} \in \boldsymbol{\mathcal{S}}_{\mathrm{OS}} = \{\varepsilon_1, \ldots, \varepsilon_M\}$.

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 Figure 2.1.

2.3.3 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.

$$C \sim p_{a_{\mathrm{APP}}}(c) = p(c \mid a_{\mathrm{APP}}) \tag{2.11}$$

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

$$\begin{aligned} &\rho(n+1) = \min\{\max\{\rho(n) + 1 - \lfloor t(n, s_{\mathrm{OS}}, a_{\mathrm{APP}}, \boldsymbol{b}) \cdot v \rfloor, 0\}, \rho^{\max}\} \\ &\rho(0) = \rho^{\mathrm{initial}}, \end{aligned} \tag{2.12}$$

where

$$t(n, s_{\mathrm{OS}}, a_{\mathrm{APP}}, \boldsymbol{b}) = \frac{c(n, a_{\mathrm{APP}})}{\varepsilon(n) f(n)} \text{ (seconds)} \tag{2.13}$$

[3] We know from (2.3) that, unlike the other layers, the APP layer's state transition depends on its external *and* internal actions. For our application, this makes the distinction between the two types of actions at the APP layer largely inconsequential.

is the n th DU's processing delay, which depends on the processor frequency $f(n)$ and the fraction of time, $s^n_{\mathrm{OS}} = \varepsilon(n)$, allocated to the application at stage n. In (2.12), 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 $\rho^{\mathrm{initial}} = \rho^{\mathrm{max}}/2$ 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 (2.3).

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

$$p(s'_{\mathrm{APP}}{=}\rho' | \boldsymbol{s}, a_{\mathrm{APP}}, \boldsymbol{b}) = \begin{cases} p_{a_{\mathrm{APP}}}\{\tilde{t} \geq \rho + 1\}, & \rho \in \{0,\ldots,\rho^{\max}\},\ \rho' = 0 \\ p_{a_{\mathrm{APP}}}\{0 \leq \tilde{t} < 2\}, & \rho = \rho' = \rho^{\max} \\ p_{a_{\mathrm{APP}}}\{\rho - \rho' + 1 \leq \tilde{t} < \rho - \rho' + 2\}, & \text{otherwise} \end{cases} \tag{2.14}$$

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

$$\alpha_{\mathrm{APP}}(s_{\mathrm{APP}}, a_{\mathrm{APP}}) = d(a_{\mathrm{APP}}) + \lambda_{\mathrm{rd}} r(a_{\mathrm{APP}}), \tag{2.15}$$

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 Figure 2.1.

2.3.4 The System Reward

Recall that the reward defined in (2.4) 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 (2.5) can be written as

$$R_{\text{in}}(\boldsymbol{s},\boldsymbol{b}) = g(\boldsymbol{s},\boldsymbol{b}) - \omega^{b}_{\text{HW}} \underbrace{\beta_{\text{HW}}(s_{\text{HW}}, a_{\text{HW}})}_{f^{3}}. \tag{2.16}$$

In this chapter, 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

$$g(\boldsymbol{s},\boldsymbol{b}) = \sum_{s'_{\text{APP}} \in \mathcal{S}} g\big(s_{\text{APP}}, \boldsymbol{b}, s'_{\text{APP}}\big)\, p(s'_{\text{APP}} | \boldsymbol{s}, a_{\text{APP}}, \boldsymbol{b}), \tag{2.17}$$

where we let

$$g\big(s_{\text{APP}}, \boldsymbol{b}, s'_{\text{APP}}\big) = \begin{cases} \Omega, & \text{if } \Lambda \le s'_{\text{APP}} \le \rho^{\max}\text{-}\Lambda \\ \dfrac{s'_{\text{APP}}}{\Lambda}\Omega, & \text{if } s'_{\text{APP}} < \Lambda \\ \dfrac{\rho^{\max} - s'_{\text{APP}}}{\Lambda}\Omega, & \text{otherwise} \end{cases} \tag{2.18}$$

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 (2.6) can be written as

$$R_{\text{ex}}(\boldsymbol{s},\boldsymbol{a}) = -\omega^{a}_{\text{APP}} \underbrace{\alpha_{\text{APP}}(s_{\text{APP}}, a_{\text{APP}})}_{d+\lambda_{\text{rd}} r} - \omega^{a}_{\text{OS}} \underbrace{\alpha_{\text{OS}}(s_{\text{OS}}, a_{\text{OS}})}_{\phi}. \tag{2.19}$$

2.4 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 (2.9) using a centralized MDP. Then, in Section 2.5, we present an alternative solution to the same problem based on a layered MDP.

2.4.1 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 (2.9). 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 (2.3), and the reward is given by (2.4). The solutions to the centralized MDP described below and the layered solution described in Section 2.5 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 2.6.3, we discuss the impact of an inaccurate probability transition model p on the system's performance.

In this chapter, 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

$$V^\pi(\boldsymbol{s}) = \underbrace{R(\boldsymbol{s}, \pi(\boldsymbol{s}))}_{\text{current expected reward}} + \underbrace{\gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p\big(\boldsymbol{s}' \mid \boldsymbol{s}, \pi(\boldsymbol{s})\big) V^\pi(\boldsymbol{s}')}_{\text{expected future rewards}}. \tag{2.20}$$

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 (2.20) for all $\boldsymbol{s}$ (or, equivalently, maximizes (2.9) for all $\boldsymbol{s}^{n_0}$). The optimal state-value function is defined as

$$V^*(\boldsymbol{s}) = \max_{\pi \in \Pi} \{V^\pi(\boldsymbol{s})\}, \ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}, \tag{2.21}$$

and the optimal policy π^* is defined as

$$\pi^*(\boldsymbol{s}) = \arg\max_{\pi \in \Pi} \{V^\pi(\boldsymbol{s})\}, \ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}. \tag{2.22}$$

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:

$$V_k^*(\boldsymbol{s}) = \max_{\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}} \left\{ R(\boldsymbol{s}, \boldsymbol{\xi}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p\left(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{\xi}\right) V_{k-1}^*(\boldsymbol{s}') \right\}, \tag{2.23}$$

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 (2.22) 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 $|\boldsymbol{\mathcal{S}}|$.

2.4.2 Centralized Cross-Layer System

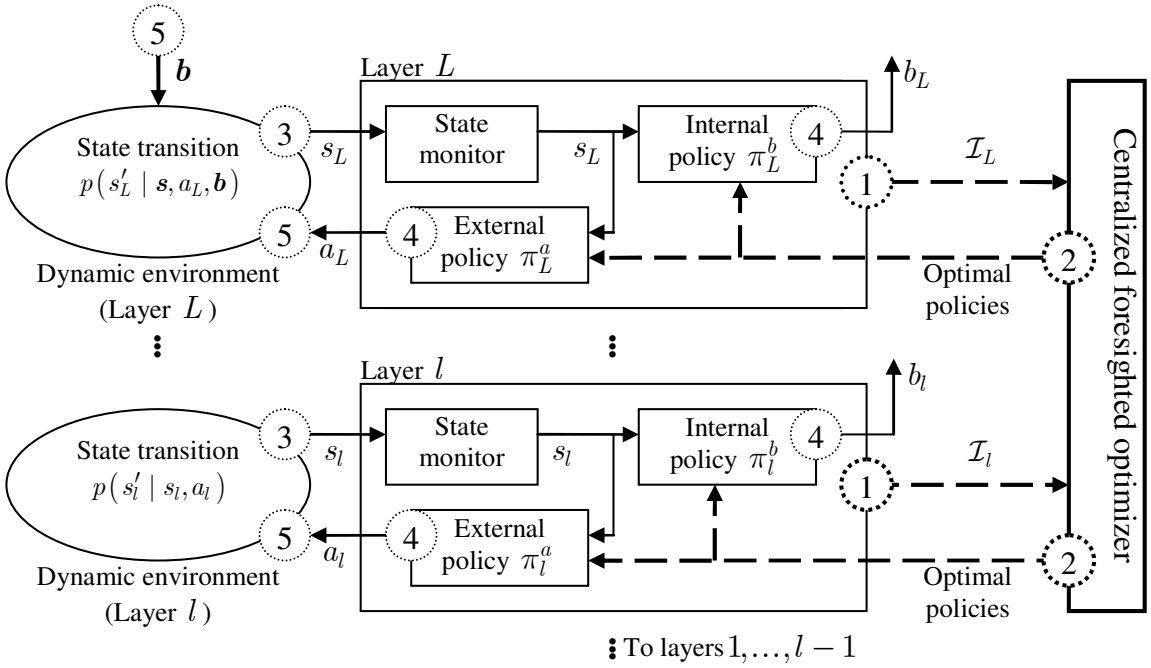


Figure 2.2. Centralized cross-layer system optimization framework. The portions of the figure that are in bold comprise the centralized information exchange process. If the dynamics are stationary, then steps 1 and 2 only need to happen once.

The centralized foresighted framework introduced in the previous subsection requires

each layer to convey the following information tuples to the centralized optimizer:

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

Figure 2.2 summarizes the centralized foresighted cross-layer framework.

There are several limitations to this centralized framework, which we have already described in the introduction (Section 2.1). 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.

2.5 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 (2.9)). 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. Figure 2.3 illustrates the layered framework. The flow of information in Figure 2.3 is the same as in Figure 2.2 except that steps 1 and 2 are replaced by the layered optimizer, which we describe in detail in Section 2.5.2, and the states and internal actions at the lower layers determine the QoS to the upper layers as we described in Section 2.2.6.

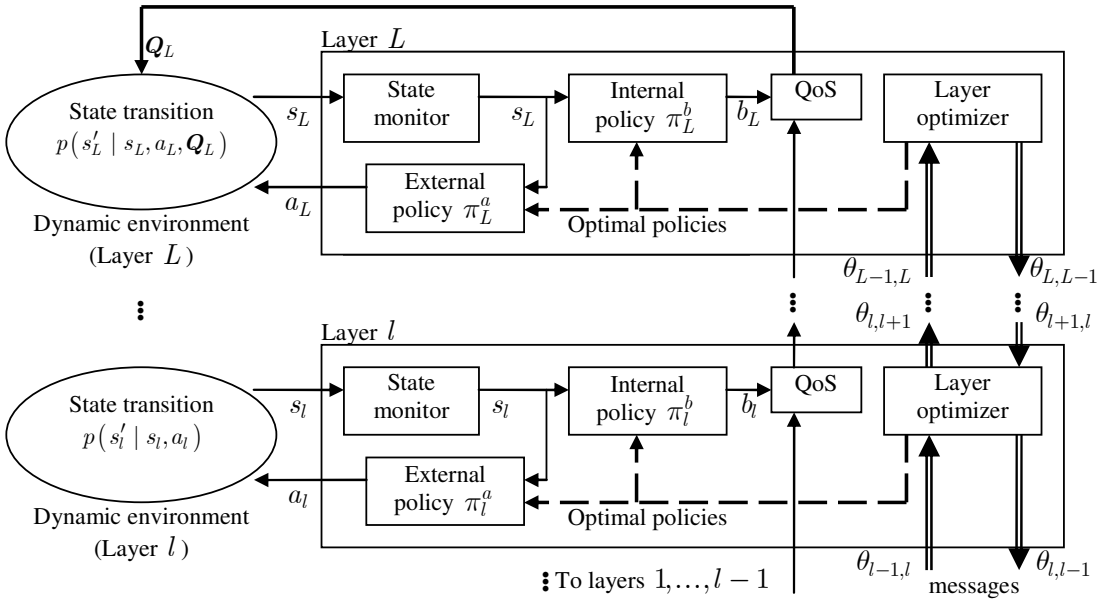


Figure 2.3. Layered cross-layer system optimization framework with message exchanges.

2.5.1 Layered MDP for Cross-Layer Decision Making

Definition: Layered MDP [3]**.** The layered MDP model is defined by the tuple $\left\langle \boldsymbol{\mathcal{L}},\ \boldsymbol{\mathcal{S}},\ \boldsymbol{\mathcal{X}},\ \{\Theta_{l,l+1}\}_{l=1}^{L-1},\ \{\Theta_{l,l-1}\}_{l=2}^{L},\ p,\ R,\ \gamma \right\rangle$, 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 l th 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.

2.5.2 Layered Value Iteration

In this subsection, we describe the proposed offline algorithm for evaluating the optimal state-value function V^* (see (2.21)) and the corresponding optimal policy π^* (see (2.22)) without a centralized optimizer. This is done using an algorithm called *layered value iteration* (layered VI) [3], which is illustrated in Figure 2.4.

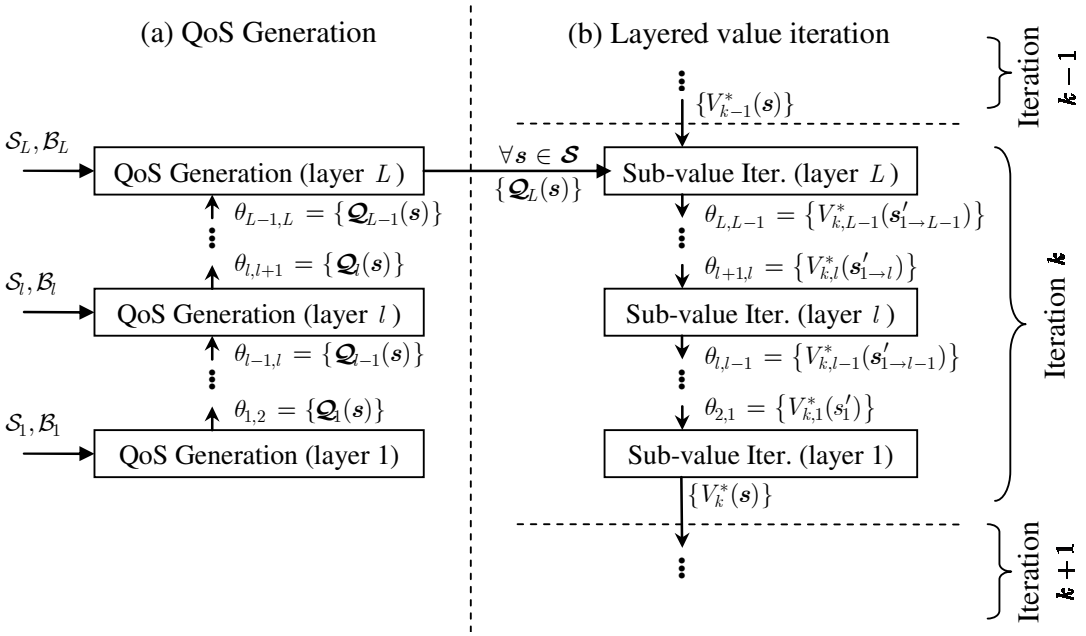


Figure 2.4. Layered VI with upward and downward messages. (a) During an initial QoS generation period, upward messages are used to tell the application layer which QoS levels are supported in each system state. The set of QoS levels at layer L is required to perform layered VI. (b) Downward messages 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 2.2.6. Figure 2.4(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 $\theta_{l,l+1} = \{\boldsymbol{\mathcal{Q}}_l(\boldsymbol{s})\}$ to layer $l+1$ (e.g. the HW layer tells the OS layer its available frequency and power combinations). We let the notation $\{\boldsymbol{\mathcal{Q}}_l(\boldsymbol{s})\}$ 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 Figure 2.4(b) begins.

We derive the layered VI algorithm by substituting the factored transition probability function and the factored reward function defined in (2.3) and (2.4), respectively, into the centralized VI algorithm defined in (2.23): i.e.,

$$V_k^*(\boldsymbol{s}) = \max_{\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}} \left\{ \begin{array}{l} g(\boldsymbol{s}, \boldsymbol{b}) - \sum_{l=1}^{L} \omega_l^a \alpha_l\,(s_l, a_l) - \sum_{l=1}^{L} \omega_l^b \beta_l\,(s_l, b_l) + \\ \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p\left(s_L' \mid \boldsymbol{s}, a_L, \boldsymbol{b}\right) \prod_{l=1}^{L-1} p\left(s_l' \mid s_l, a_l\right) V_{k-1}^*(\boldsymbol{s}') \end{array} \right\}. \tag{2.24}$$

Recall from (2.7) and (2.8) in Section 2.2.6 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 (2.24) 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 (2.24) as:

$$V_k^*(\boldsymbol{s}) = \max_{\substack{\boldsymbol{a}_{1\to L}\in\boldsymbol{\mathcal{A}}_{1\to L},\\ Q_L\in\boldsymbol{\mathcal{Q}}_L(\boldsymbol{s})}} \left\{ \begin{array}{l} R_{\text{in}}(s_L,Q_L) - \sum_{l=1}^{L} \omega_l^a \alpha_l\,(s_l,a_l\,) + \\ \gamma \sum_{\boldsymbol{s}'_{1\to L}\in\boldsymbol{\mathcal{S}}_{1\to L}} \prod_{l=1}^{L-1} p\big(s'_l \mid s_l,a_l\big) p\big(s'_L \mid s_L,a_L,Q_L\big) V_{k-1}^*(\boldsymbol{s}') \end{array} \right\}, \qquad (2.25)$$

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 $\sum_{\boldsymbol{s}'_{1\to L-1}\in\boldsymbol{\mathcal{S}}_{1\to L-1}} \prod_{l=1}^{L-1} p\big(s'_l \mid s_l,a_l\big) = 1$, 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:

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

(2.26)

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 (2.26) 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**:**

$$V^*_{k-1}(\boldsymbol{s}'_{1\to L\text{-}1}) = \max_{\substack{\boldsymbol{a}_L\in\mathcal{A}_L,\\ Q_L\in\boldsymbol{Q}_L}} \left\{ \begin{matrix} R_{\text{in}}(s_L,Q_L) - \omega^a_L \alpha_L\,(\,s_L,a_L\,) + \\ \gamma \sum\limits_{\boldsymbol{s}'_L\in\mathcal{S}_L} p\big(\,s'_L \mid s_L,a_L,Q_L\,\big) V^*_{k-1}(\boldsymbol{s}') \end{matrix} \right\}. \tag{2.27}$$

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 $\sum\limits_{(\boldsymbol{s}'_{1\to l\text{-}1})\in\boldsymbol{\mathcal{S}}_{1\to l\text{-}1}} \prod\limits_{l=1}^{l-1} p\big(\,s'_l \mid s_l,a_l\,\big) = 1$. Hence, similar to how we obtained (2.27) from (2.25), 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 $l = 2,\ldots,L-1$**:**

$$V^*_{k-1}(\boldsymbol{s}'_{1\to l\text{-}1}) = \max_{\boldsymbol{a}_l\in\mathcal{A}_l} \left\{ -\omega^a_l \alpha_l\,(\,s_l,a_l\,) + \gamma \sum_{\boldsymbol{s}'_l\in\mathcal{S}_l} p\big(\,s'_l \mid s_l,a_l\,\big) V^*_{k-1}(\boldsymbol{s}'_{1\to l}) \right\}, \tag{2.28}$$

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. $\theta_{l+1,l} = \big\{ V^*_{k-1}(\boldsymbol{s}'_{1\to l}) \big\}$.

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

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

$$V^*_k(\boldsymbol{s}) = \max_{\boldsymbol{a}_1\in\mathcal{A}_1} \left\{ -\omega^a_1 \alpha_1\,(\,s_1,a_1\,) + \gamma \sum_{\boldsymbol{s}'_1\in\mathcal{S}_1} p\big(\,s'_1 \mid s_1,a_1\,\big) V^*_{k-1}(s'_1) \right\}, \tag{2.29}$$

where $\theta_{2,1} = \big\{ V^*_{k-1}(s'_1) \big\}$ 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 (2.22), and can be stored in a look-up table at each layer to determine the optimal state-to-action mappings at run-time.

2.5.3 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 (2.27) has complexity

$$Comp_L = O\left(|\boldsymbol{\mathcal{S}}| |\mathcal{S}_L| \prod_{l'=1}^{L-1} |\mathcal{S}_{l'}| |\mathcal{A}_L| |\boldsymbol{\mathcal{Q}}_L| \right) = O\left(|\boldsymbol{\mathcal{S}}|^2 |\mathcal{A}_L| \overbrace{\prod_{l'=1}^{L} |\mathcal{B}_{l'}|}^{|\boldsymbol{\mathcal{Q}}_L|} \right). \tag{2.30}$$

The sub-VIs at layers $l \in \{2, \ldots, L-1\}$ defined in (2.28) have complexity

$$Comp_l = O\left(|\boldsymbol{\mathcal{S}}||\mathcal{S}_l|\prod_{l'=1}^{l-1}|\mathcal{S}_{l'}||\mathcal{A}_l|\right). \tag{2.31}$$

Finally, the sub-VI at layer 1 defined in (2.29) has complexity

$$Comp_1 = O\left(|\boldsymbol{\mathcal{S}}||\mathcal{S}_1||\mathcal{A}_1|\right). \tag{2.32}$$

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

$$Comp = \sum_{l=1}^{L} Comp_l\,. \tag{2.33}$$

We compare the centralized VI and layered VI complexities in Table 2.1 assuming that states and actions are as defined in Table 2.2.

Table 2.1. 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	$O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert\lvert\mathcal{S}_1\rvert\lvert\mathcal{A}_1\rvert}_{(29\cdot3)\cdot1\cdot1}\right) + O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert\lvert\mathcal{S}_2\rvert\lvert\mathcal{S}_1\rvert\lvert\mathcal{A}_2\rvert}_{(29\cdot3)\cdot3\cdot1\cdot2}\right) + O\left(\underbrace{\lvert\boldsymbol{\mathcal{S}}\rvert^2\lvert\mathcal{A}_3\rvert\overbrace{\lvert\boldsymbol{\mathcal{B}}\rvert}^{\lvert\mathcal{Q}_3\rvert}}_{(29\cdot3)^2\cdot3\cdot3\cdot1}\right)$ $= 87 + 522 + 68121$ $= 68730$		

2.6 Results

In this section, we test the performance of the proposed framework using the illustrative cross-layer system described in Section 2.3. Table 2.2 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 2.2. 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 2.2. Nevertheless, our most important observations are about the fundamental properties of the proposed framework, which are independent of the chosen example.

Table 2.2. 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	$\omega^{b}_{\text{HW}} = 9 \times 10^{-27}$

2.6.1 Equivalence of Centralized and Layered Value Iteration

Figure 2.5 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.

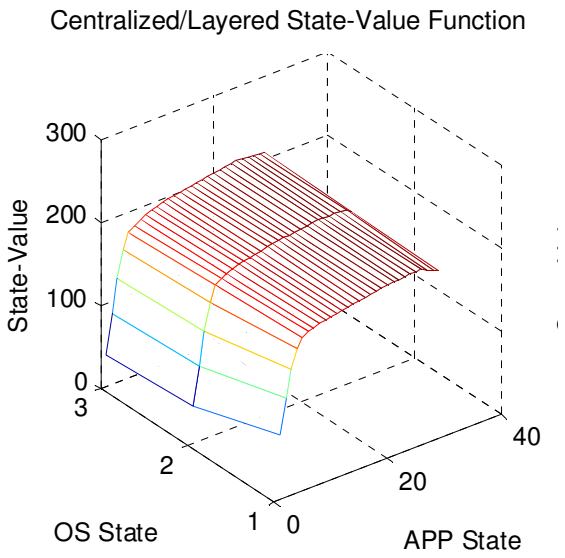


Figure 2.5. State-value function obtained from centralized VI and layered VI $(\gamma = 0.9)$.

2.6.2 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 2.3 illustrates the impact of the discount factor on the average reward (defined as the sum of (2.16) and (2.19)) achieved over five simulations of the centralized system and five simulations of the layered system. (Because the reward is very abstract, in Section 2.6.5 we provide more detailed simulation results in a variety of simulation scenarios.) As expected, the results in Table 2.3 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 2.3. 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

2.6.3 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 chapter, 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:

$$\bar{R}^{\pi} = \sum_{\boldsymbol{s}\in\boldsymbol{\mathcal{S}}} \mu^{\pi}(\boldsymbol{s}) R(\boldsymbol{s},\pi(\boldsymbol{s}))\,. \tag{2.34}$$

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:

$$\bar{R}^{\pi}(N) = \frac{1}{N} \sum_{n=0}^{N-1} R(\boldsymbol{s}^n,\pi(\boldsymbol{s}^n))\,. \tag{2.35}$$

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. Figure 2.6 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 (2.11). Because the actual complexity traffic is not stationary, the buffer transition model in (2.14) is not accurately represented as a stationary Markov chain. Nevertheless, the data in Table 2.4 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 2.4. 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

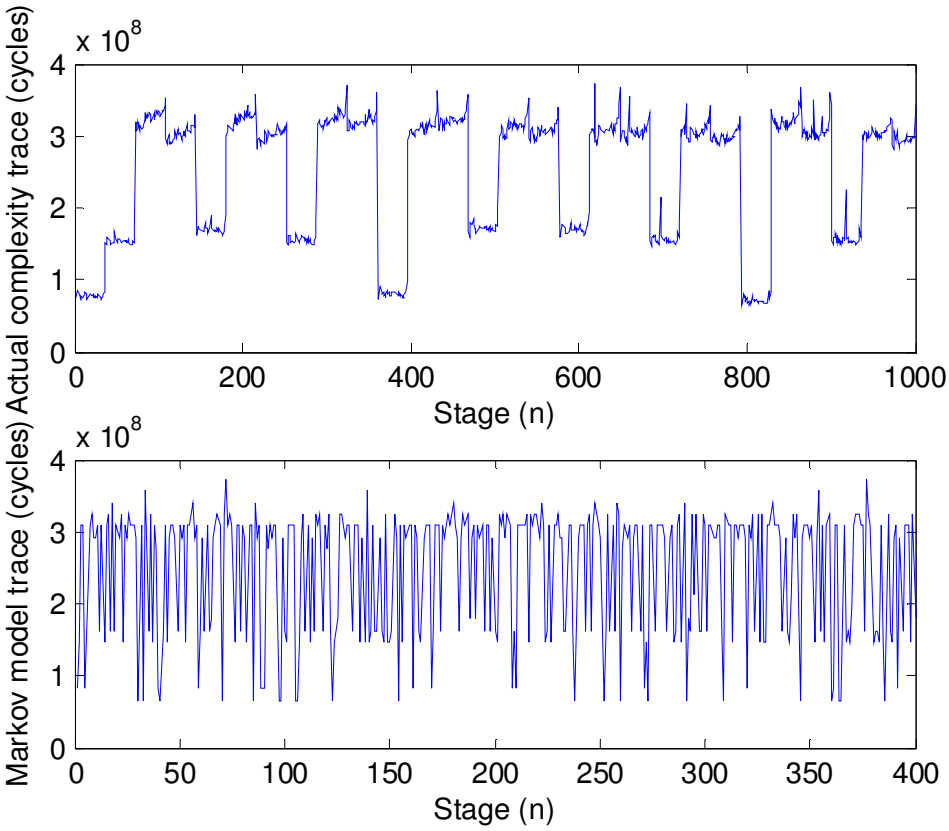


Figure 2.6. Actual complexity trace (top) vs. trace generated by a stationary model (bottom). The stationary model's parameters are trained based on the actual complexity trace.

2.6.4 Myopic and Foresighted Simulation Trace Comparison

Figure 2.7 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 2.3.

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.

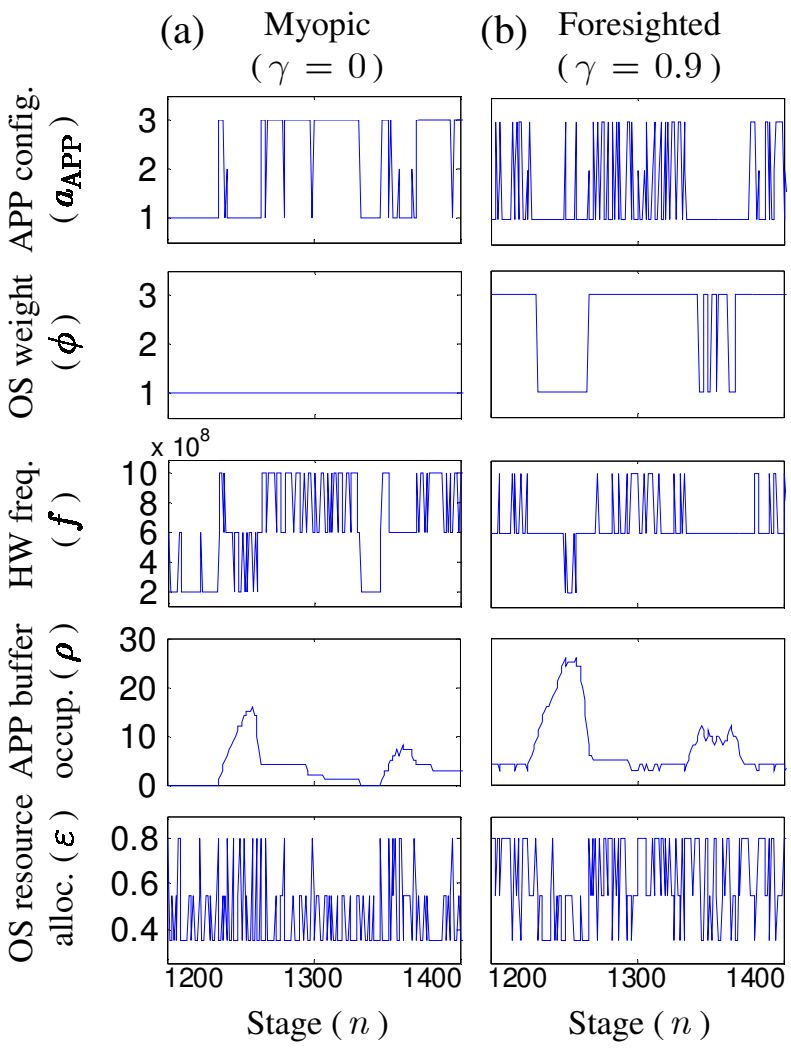


Figure 2.7. Simulation traces from stage 1200 to 1400. (a) Simulation with myopic cross-layer optimization ($\gamma = 0$). (b) Simulation with foresighted cross-layer optimization ($\gamma = 0.9$).

2.6.5 Performance of Simplifications of the Proposed Framework

In Figure 2.8, 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 Figure 2.8 is shown in Table 2.5.

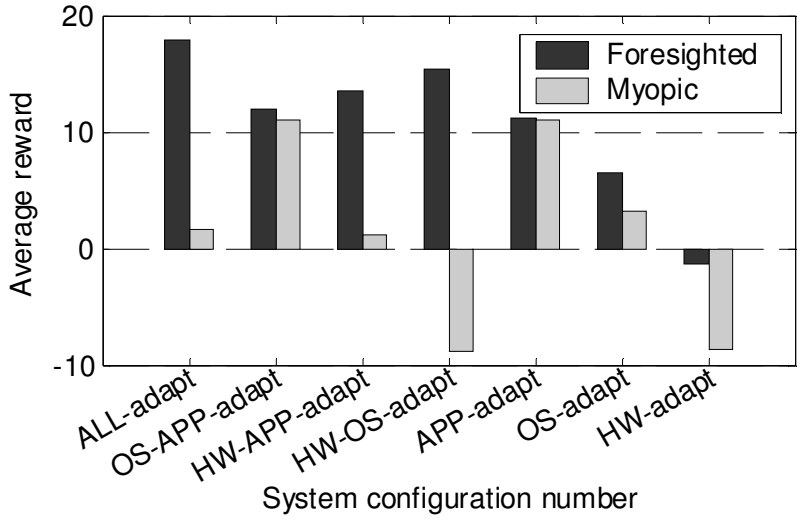


Figure 2.8. Comparison of the performance of different system configurations.

Table 2.5. 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 Figure 2.8 and Table 2.5. For each of these configurations, the cross-layer

system's parameters are defined as in Table 2.2 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 Figure 2.8 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 $a_{\text{APP}} = 2$. 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 $a_{\text{APP}} = 2$.

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 Figure 2.7). 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.

2.7 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.

Chapter 3

Layered Reinforcement Learning

In the previous chapter, we proposed a systematic cross-layer framework for dynamic multimedia systems, which allows each layer to make *autonomous* and *foresighted* decisions that maximize the system's *long-term performance*, while meeting the application's real-time delay constraints. The proposed solution solved the cross-layer optimization *offline*, under the assumption that the multimedia system's probabilistic dynamics were *known* a priori, by modeling the system as a layered Markov decision process. In practice, however, these dynamics are *unknown* a priori and therefore must be learned *online*. In this chapter, we address this problem by allowing the multimedia system layers to learn, through repeated interactions with each other, to autonomously optimize the system's long-term performance at run-time. The two key challenges in this layered learning setting are: (i) each layer's learning performance is directly impacted by not only its own dynamics, but also by the learning processes of the other layers with which it interacts; and, (ii) selecting a learning model that appropriately balances time-complexity (i.e. learning speed) with the multimedia system's limited memory and the multimedia application's real-time delay constraints. We propose two reinforcement learning algorithms for optimizing the system under different design constraints: the first

algorithm solves the cross-layer optimization in a centralized manner, and the second solves it in a decentralized manner. We analyze both algorithms in terms of their required computation, memory, and inter-layer communication overheads. After noting that the proposed reinforcement learning algorithms learn too slowly, we introduce a complementary accelerated learning algorithm that exploits partial knowledge about the system's dynamics in order to dramatically improve the system's performance. In our experiments, we demonstrate that decentralized learning can perform equally as well as centralized learning, while enabling the layers to act autonomously. Additionally, we show that existing application-independent reinforcement learning algorithms, and existing myopic learning algorithms deployed in multimedia systems, perform significantly worse than our proposed application-aware and foresighted learning methods.

3.1 Introduction

State-of-the-art multimedia technology is poised to enable widespread proliferation of a variety of life-enhancing applications, such as video conferencing, emergency services, surveillance, telemedicine, remote teaching and training, augmented reality, and distributed gaming. However, efficiently designing and implementing such 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 experienced by the system (e.g. video source characteristics, user requirements, workload characteristics, number of running

applications, memory/cache behavior etc.).

Cross-layer adaptation is an increasingly popular solution for addressing these challenges in dynamic multimedia systems (DMSs) [1]-[8]. This is because the performance of DMSs (e.g. video rate-distortion costs, delay, and power consumption) can be significantly improved by jointly optimizing parameters, configurations, and algorithms across two or more system layers (i.e. the application, operating system, and hardware layers), rather than optimizing and adapting them in isolation. However, existing cross-layer solutions have several important limitations.

Centralized cross-layer solutions: The majority of these solutions require a central optimizer to coordinate the application, operating system, and hardware adaptations to optimize the performance of one or multiple multimedia applications sharing the DMS's limited resources. To this end, a new interface between the central optimizer and all of the layers is created, requiring extensive modifications to the operating system [4] [5] or the introduction of an entirely new middleware layer [1] [2] [3] [24] [25]. Unfortunately, these approaches ignore the fact that the majority of modern system's are comprised of layers (components or modules) that are designed by different manufacturers, which makes such tightly integrated designs impractical, if not impossible [8] [18] [19].

Myopic cross-layer solutions: Another limitation of many existing cross-layer solutions for DMSs is that they are *myopic*. In other words, cross-layer decisions are made reactively in order to optimize the *immediate* utility, without considering the impact of these decisions on the future utility. However, in DMSs, it is essential to predict the impact of the current decisions on the long-term utility because the multimedia source

characteristics, workload characteristics, OS and hardware dynamics are often correlated across time. Moreover, in DMSs, utility fluctuations across time lead to poor user experience and bad resource planning leads to inefficient resource usage and wasted power [26] [27]. In contrast, *foresighted* (i.e. long-term) optimization techniques take into account the impact of immediate cross-layer decisions on the DMS's expected future utility. Importantly, foresighted policy optimizations have been successfully deployed at the hardware layer to solve the dynamic power management problem [11]-[13]; however, with the exception of our prior work [9], they have never been used for cross-layer DMS optimization, where *all the layers make foresighted decisions*.

Single-layer learning solutions: The various multimedia system layers must be able to adapt at run-time to their experienced dynamics. Most existing learning solutions for multimedia systems are concerned with modeling the unknown and potentially time-varying environment experienced by a single layer. A broad range of single-layer learning techniques have been deployed in multimedia (and general purpose) systems in recent years, which include estimation techniques such as maximum likelihood estimation [4] [5] [10] [12] [13], statistical fitting [7], regression methods [28], adaptive linear prediction [29], and ad-hoc estimation heuristics [3]. However, these solutions ignore the fact that DMSs do not only experience dynamics at a single layer (e.g. time-varying workload), but at multiple (possibly all) layers, which interact with each other. Hence, it is necessary to enable the layers to learn at run-time how to autonomously and asynchronously adapt their processing strategies based on forecasts about (i) their *future dynamics* and, importantly, (ii) *how they impact and are impacted by the other layers*.

In summary, while significant contributions have been made to enhance the performance of DMSs using cross-layer design techniques, no systematic cross-layer framework exists that explicitly considers: (i) the design and implementation constraints imposed by a layered DMS architecture; (ii) the ability of the various layers to autonomously make foresighted decisions in order to jointly maximize the DMS's utility; and, most importantly, (iii) how layers can learn their unknown environmental dynamics and determine how they impact and are impacted by the other layers.

In this chapter, similar to [6] [7] [8], we consider a multimedia system in which (i) a video encoder at the application layer makes rate-distortion-complexity tradeoffs by adapting its configuration and (ii) the operating system layer makes energy-delay tradeoffs by adapting the hardware layer's operating frequency. Our contributions are as follows:

- We propose a centralized reinforcement learning algorithm for online cross-layer DMS optimization based on the well-known Q-learning algorithm. This centralized solution can be easily implemented if a single manufacturer designs all of the layers in the multimedia system.
- We propose a novel layered Q-learning algorithm, which allows the loosely-coupled multimedia system layers to autonomously learn their optimal foresighted policies online, through repeated interactions with each other. This decentralized solution can be implemented if the layers are designed by different manufacturers. We show experimentally that the proposed layered learning algorithm, which adheres to the layered system architecture, performs equally as well as a traditional centralized

learning algorithm, which does not.

- We propose a reinforcement learning 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 reinforcement learning techniques [14], [15], 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. Importantly, the accelerated learning algorithm can be implemented in both the centralized and layered scenarios.

The remainder of this chapter is organized as follows. In Section 3.2, we present the system model. In Section 3.3, we formulate the cross-layer problem as a layered Markov decision process (MDP). In Section 3.4, we propose a centralized and layered (decentralized) Q-learning algorithm for run-time cross-layer optimization. In Section 3.5, we propose a technique to accelerate the rate of learning by exploiting partial a priori knowledge that we have about the system's dynamics. In Section 3.6, we present our experimental results and we conclude the chapter in Section 3.7.

3.2 System Specification

In this section, we model a system in which a video encoding application makes rate-distortion-complexity tradeoffs by adapting its configuration and the operating system makes energy-delay tradeoffs by dynamically adapting the DVS-enabled (dynamic voltage scaling) hardware's operating frequency. In Section 3.3, we map this illustrative

system into the proposed layered MDP model.

As in [21], 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 into the application's pre-encoding buffer. Similar to [11], we assume that the system operates over discrete time slots. Additionally, we assume that the time slots have variable length, such that one data unit is encoded in each time slot. We interchangeably refer to the time slot during which the nth data unit is encoded as the nth "time slot" or "stage," where $n \in \mathbb{N}$.

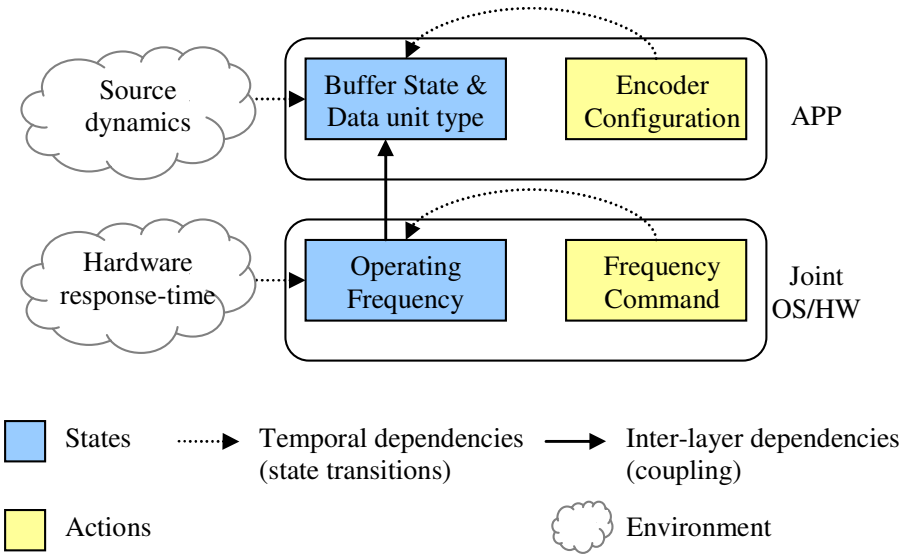


Figure 3.1. System diagram showing the states, actions, and environment at each layer.

3.2.1 Dynamic Voltage Scaling Model

The nth data unit is processed at the current operating frequency, which is a member of the state set $\mathcal{S}_f = \{f_i : i = 1,\ldots,N_f\}$. The operating frequency can be adapted from one

time slot to the next by asserting a command in the action set $\mathcal{A}_u = \{u : u \in \mathcal{S}_f\}$. Similar to [11], we assume that there is a non-deterministic delay associated with the operating frequency transition such that

$$p_f\left(f^{n+1} \mid f^n, u^n\right) = \begin{cases} \beta, & \text{if } f^{n+1} = u^n \neq f^n \\ 1-\beta, & \text{if } f^{n+1} = f^n \neq u^n \\ 1, & \text{if } f^{n+1} = f^n = u^n \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where $\{f^n : n \in \mathbb{N}\}$ is the sequence of operating frequencies and $\beta \in (0,1]$ $(1-\beta)$ is the probability of a successful (an unsuccessful) operating frequency transition. In Figure 3.1, this transition is illustrated by a temporal dependency between the frequency command (action) and operating frequency (state) in the joint OS/HW layer. Note that, regardless of the frequency command u^n, the nth data unit is processed at the operating frequency $f^n \in \mathcal{S}_f$. We define the cost $J_{\text{OS}}(f^n) = P(f^n)$ (watts) as the power dissipated at operating frequency $f \in \mathcal{S}_f$, where $P(\cdot)$ is the system's power-frequency function (see, for example, [5] [7]).

3.2.2 Application Model

The nth data unit can be classified as being one of N_z types in the state set $\mathcal{S}_z = \{z_i : i = 1,\ldots,N_z\}$. The set of states $\mathcal{S}_z$ depends on the specific video coder being used at the APP layer. For illustration, 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 [7] [20]. It has been shown that transitions among data unit types and frame activity levels in an (adaptive) group of pictures structure can be modeled as a stationary Markov process [7] [20] [30]: i.e.,

$$p_z(z^{n+1} \mid z^n) \tag{3.2}$$

where $\{z^n : n \in \mathbb{N}\}$ is the sequence of data unit types. We assume that the choice of data unit type is governed by an algorithm similar to the one in [20]; therefore, the probabilities in (3.2) depend on the desired ratio of I, P, and B data units (e.g. IBBP…), the source characteristics (i.e. the APP layer's environment illustrated in Figure 3.1), and the condition used to decide when to code an I frame [20]. Modeling how the multimedia data evolves over time enables us to utilize the system's resources more efficiently, while still ensuring continuity of the multimedia stream's quality over time.

The nth data unit can be coded using any one of N_h encoding parameter configurations in the action set $\mathcal{A}_h = \{h_i : i = 1,\ldots,N_h\}$ (for example, to support its real-time requirements, a video encoder can adapt its choice of quantization parameter, macroblock partition size, deblocking filter, entropy coding scheme, sub-pixel motion accuracy, motion-vector search range, and motion estimation search algorithm). Given the data unit type $z^n \in \mathcal{S}_z$, each configuration $h^n \in \mathcal{A}_h$ achieves different operating points in the rate-distortion-complexity space [6]. We let $b^n(z^n,h^n)$, $d^n(z^n,h^n)$, and $c^n(z^n,h^n)$ represent the encoded bit-rate (bits per data unit), encoded distortion (mean square error), and encoding complexity (cycles), respectively. We assume that $b^n(z^n,h^n)$,

$d^n(z^n,h^n)$, and $c^n(z^n,h^n)$ are instances of the i.i.d. random variables $B(z^n,h^n)$, $D(z^n,h^n)$, and $C(z^n,h^n)$, respectively. Note that, in the proposed learning framework, we do not need to know the distributions of these random variables (see Section 3.4).

We penalize the application configuration by employing the Lagrangian cost measure $J_{\mathrm{APP}}(z^n,h^n) = d^n(z^n,h^n) + \lambda_{rd} b^n(z^n,h^n)$ used in the H.264/AVC reference encoder for making rate-distortion optimal mode decisions, where $\lambda_{rd} \in [0,\infty)$ is a Lagrangian multiplier, which can be set based on the rate-constraints.

At stage n, the application's pre-encoding buffer contains $q^n \in \mathcal{S}_q = \{q_i : i = 0,\ldots,N_q\}$ data units, where N_q is the maximum number of data units that can be stored in the buffer. In this chapter, we assume that N_q is not limited by the available memory, but instead it is determined by the specific application's delay-constraints [21]. We assume that data units arrive in the pre-encoding buffer at η data units per second based on a deterministic arrival process from the video capture device (for example, if data units are frames, then η will typically be 15, 24, or 30 frames per second). If the buffer is full when a data unit arrives, then that data unit is discarded. In the following paragraphs, we discuss the buffer model in detail because the accelerated learning algorithm proposed in Section 3.5 exploits its structure.

The pre-encoding buffer's state at stage $n+1$ can be expressed recursively based on its state at stage n: i.e.,

$$\begin{aligned} q^{n+1} &= \min\left\{\left[q^n + \lfloor t^n(z^n,h^n,f^n)\cdot\eta\rfloor - 1\right]^+, N_q\right\} \\ q^0 &= q_{\mathrm{init}}, \end{aligned} \tag{3.3}$$

where

$$t^n(z^n, h^n, f^n) = \frac{c^n(z^n, h^n)}{f^n} \text{ (seconds)} \tag{3.4}$$

is the n th data unit's processing delay (in Figure 3.1, the coupling between the operating frequency and processing delay is represented by the inter-layer dependency arrow); $\lfloor x \rfloor$ is the integer part of x ; and $\lceil x \rceil^+ = \max\{x, 0\}$. Note that $c^n(z^n, h^n)$ is an instance of the random variable $C(z^n, h^n)$ with distribution $C(z^n, h^n) \sim p_C\left(c^n \mid z^n, h^n\right)$.

In (3.3), the -1 indicates that the n th data unit departs the pre-encoding buffer after the non-deterministic encoding delay $T(f^n, z^n, h^n) = C(z^n, h^n) / f^n$ (seconds). Meanwhile, the number of data units that arrive in the pre-encoding buffer during the n th time slot, $\tilde{t}^n(z^n, h^n, f^n) = t^n(z^n, h^n, f^n) \cdot \eta$, is an instance of the random variable $\tilde{T}(f^n, z^n, h^n) = \eta C(z^n, h^n) / f^n$ (data units) with distribution

$$\tilde{T}(f^n, z^n, h^n) \sim p_{\tilde{T}}(\tilde{t}^n \mid f^n, z^n, h^n) = \frac{f^n}{\eta} \cdot p_C\left(\frac{f^n}{\eta} \cdot c^n \mid z^n, h^n\right) \tag{3.5}$$

Based on (3.3) and (3.5), the pre-encoding buffer's state transition can be modeled as a controllable Markov chain with transition probabilities

$$\begin{aligned} &p_{\text{APP}}^{(q)}\left(q^{n+1} \mid q^n, f^n, z^n, h^n\right) \\ &\quad = \begin{cases} p_{\tilde{T}}\left\{\tilde{T} < 2 \mid f^n, z^n, h^n\right\}, & q^n = q^{n+1} = 0 \\ p_{\tilde{T}}\left\{\tilde{T} \geq N_q - q + 1 \mid f^n, z^n, h^n\right\}, & q^{n+1} = N_q \\ p_{\tilde{T}}\left\{\Omega \leq \tilde{T} < \Omega + 1 \mid f^n, z^n, h^n\right\}, & \text{otherwise,} \end{cases} \end{aligned} \tag{3.6}$$

where $\Omega = q^{n+1} - q^n + 1$ and $\{q^n : n \in \mathbb{N}\}$ is the sequence of buffer states. Note that, from (3.3), $q^{n+1} - q^n \geq -1$, with equality when there are no data unit arrivals, i.e.

$\tilde{t}^n = 0$.

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

$$g_{\mathrm{APP}}([f^n, q^n, z^n], h^n) = 1 - \left(\frac{q^n + \lfloor \tilde{t}^n(f^n, z^n, h^n) \rfloor - 1}{N_q} \right)^2, \tag{3.7}$$

which is near its maximum when the buffer is empty (corresponding to zero queuing delay) and is minimized when the buffer is full. In Appendix A, we discuss in detail why (3.7) is a good definition for the utility gain.

3.3 Formulation as a Layered MDP

In subsection 3.3.1, we formulate the cross-layer problem as a layered MDP and map the system described in Section 3.2 to the proposed layered MDP model. Then, in subsection 3.3.2, we present the system optimization objective.

3.3.1 Layered MDP Model

A layered MDP is a tuple $\langle \mathcal{L}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{S}}, p, R \rangle$, where

- $\mathcal{L} = \{1, \ldots, L\}$ is a set of L autonomous layers, which participate in the cross-layer optimization.[1] Each layer is indexed $l \in \{1, \ldots, L\}$ with layer 1 corresponding to the lowest participating layer (e.g. the HW layer) and layer L corresponding to the

highest participating layer (e.g. the APP layer).

- $\boldsymbol{\mathcal{A}}$ is the global action set $\boldsymbol{\mathcal{A}} = \times_{l \in \boldsymbol{\mathcal{L}}} \mathcal{A}_l$[2], where $\mathcal{A}_l$ is the lth layer's local action set.
- $\boldsymbol{\mathcal{S}}$ is the global state set $\boldsymbol{\mathcal{S}} = \times_{l \in \boldsymbol{\mathcal{L}}} \mathcal{S}_l$, where $\mathcal{S}_l$ is the lth layer's local state set.
- p is the joint transition probability function mapping the global state, global action, and global next-state to a value in $[0,1]$, i.e. $p : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \times \boldsymbol{\mathcal{S}} \mapsto [0,1]$.
- R is the expected reward function, which maps the global state and global action to a real number representing the system's expected reward, i.e. $R : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \mapsto \mathbb{R}_+$.
- $\gamma \in [0,1)$ is a discount factor, defined below.

We now map the system described in Section 3.2 to the layered MDP model. We consider a system in which two layers participate in the cross-layer optimization (i.e. $\mathcal{L} = \{1,2\}$). In particular, the *application layer* (APP, $l = 2$) makes rate-distortion-complexity tradeoffs by adapting its configuration and the *operating system layer* (OS, $l = 1$) makes energy-delay tradeoffs by adapting the *hardware layer's* (HW) operating frequency (using, for example, the platform independent open standard known as the Advanced Configuration and Power Interface [11]). Due to the fact that the OS layer controls the HW layer, we combine them into a single decision making layer, which we call the joint OS/HW layer.

---

[1] 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 APP layer can adapt its source coding parameters); alternatively, if a layer does not "participate", then it is omitted.

Table 3.1 summarizes the mapping between the system defined in Section 3.2 and the layered MDP model. Since f, q, and z are Markovian, the transition of the global state $\boldsymbol{s}$ is Markovian with transition probability

$$p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) = \underbrace{p_f(f' \mid f, u)}_{p_1(s_1' \mid s_1, a_1)} \underbrace{p_q(q' \mid q, f, z, h)\, p_z(z' \mid z)}_{p_2(s_2' \mid \boldsymbol{s}, a_2)}. \tag{3.8}$$

Lastly, the reward function is defined as

$$R(\boldsymbol{s}, \boldsymbol{a}) = g_{\mathrm{APP}}(\boldsymbol{s}, \boldsymbol{a}) - \omega_{\mathrm{APP}} J_{\mathrm{APP}}(z, h) - \omega_{\mathrm{OS}} J_{\mathrm{OS}}(f), \tag{3.9}$$

where ω_{APP} and ω_{OS} are positive weights that are assumed to be given. The forms of the factored transition probability function and the additively decomposed reward function are exploited in Section 3.4.3 to derive the proposed layered learning algorithm.

Table 3.1. Abbreviated list of notation.

f	Operating frequency	u	Frequency command
z	Data unit type	h	Encoding parameter configuration
q	Buffer occupancy	$a_1 = a_{\mathrm{OS}} = u$	Joint OS/HW layer action
$s_1 = s_{\mathrm{OS}} = f$	Joint OS/HW layer state	$a_2 = a_{\mathrm{APP}} = h$	APP layer action
$s_2 = s_{\mathrm{APP}} = (z, q)$	APP layer state	$\boldsymbol{a} = (a_1, a_2)$	Global action
$\boldsymbol{s} = (s_1, s_2)$	Global state	$g_2(\boldsymbol{s}, a_2)$	Utility gain
$p_1(s_1' \mid s_1, a_1)$, $p_2(s_2' \mid \boldsymbol{s}, a_2)$	Transition probability functions for the joint OS/HW and APP layers	$J_1(s_1)$, $J_2(s_2, a_2)$	Cost functions for the joint OS/HW and APP layers

[2] $\times_{l \in \mathcal{L}} \mathcal{A}_l = \mathcal{A}_1 \times \cdots \times \mathcal{A}_L$ is the L-ary Cartesian product.

3.3.2 System Optimization Objective

Unlike existing cross-layer optimization solutions, which focus on optimizing the myopic (i.e. immediate) utility, the goal in the proposed cross-layer framework is to find the optimal actions at each stage $n \in \mathbb{N}$ that maximize the *discounted sum of future rewards* [9] [11] [14] [15], i.e.

$$\sum_{n=0}^{\infty} (\gamma)^n R\left(\boldsymbol{s}^n, \boldsymbol{a}^n \mid \boldsymbol{s}^0\right) \tag{3.10}$$

where the parameter γ $(0 \leq \gamma < 1)$ is the "discount factor," which defines the relative importance of present and future rewards, and $\boldsymbol{s}^0$ is the initial state. We refer to the Markov decision policy $\pi^* : \boldsymbol{\mathcal{S}} \mapsto \boldsymbol{\mathcal{A}}$, which maximizes the discounted sum of future rewards from each initial state $\boldsymbol{s}^0 \in \boldsymbol{\mathcal{S}}$, as the optimal *foresighted* policy. We use a *discounted* sum of rewards instead of, for example, the *average* reward, because: (i) typical video sources have temporally correlated statistics over short time intervals such that the future environmental dynamics cannot be easily predicted without error [7], and therefore the system may benefit by weighting its immediate reward more heavily than future rewards; and, (ii) The multimedia session's lifetime is not known a priori (i.e. the session may end unexpectedly) and therefore rewards should be maximized sooner rather than later.

Throughout this chapter, we will find it convenient to work with the optimal *action-value function* $Q^* : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \mapsto \mathbb{R}$ [15], which satisfies the following Bellman optimality equation:

$$Q^*(\boldsymbol{s},\boldsymbol{a}) = R(\boldsymbol{s},\boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \boldsymbol{\mathcal{S}}} p(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a}) V^*(\boldsymbol{s}'), \tag{3.11}$$

where $V^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q^*(\boldsymbol{s},\boldsymbol{a}),\ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}$, is the optimal *state-value function* [15]. In words, $Q^*(\boldsymbol{s},\boldsymbol{a})$ is the expected sum of discounted rewards achieved by taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ and then following the optimal policy $\pi^* : \boldsymbol{\mathcal{S}} \mapsto \boldsymbol{\mathcal{A}}$ thereafter, where

$$\pi^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s},\boldsymbol{a}),\ \forall \boldsymbol{s} \in \boldsymbol{\mathcal{S}}. \tag{3.12}$$

3.4 Learning the Optimal Decision Policy

3.4.1 Selecting a Learning Model

As we mentioned before, the multimedia system's dynamics (i.e. $R(\boldsymbol{s},\boldsymbol{a})$ and $p(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a})$) are *unknown* and therefore the optimal action-value function Q^* and the optimal policy π^* must be learned online, based on experience. However, it remains to explain how we can learn the optimal policy online despite these unknown quantities.

One option is to directly estimate the reward and transition probability function and then to perform value iteration [15] to determine the optimal policy; however, this solution is too complex because each iteration of the algorithm has complexity $O\left(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|\right)$ and the number of iterations required to converge to the optimal policy is polynomial in the discount factor γ.[3] Additionally, storing the estimated transition probability function incurs a memory overhead of $O\left(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|\right)$, which for a large number of states is unacceptably large (e.g. our experimental test-bed in Section 3.6 uses 765

states and 15 actions, and therefore requires over 33 MBs of memory to directly store the transition probability function using 32 bit single-precision floating point numbers). A second option, proposed in [12], avoids the complex value iteration algorithm, but incurs even greater memory overheads by requiring a large number of policies to be computed and stored offline to facilitate the online decision making process.

Clearly, given the multimedia system's limited memory and the multimedia application's real-time delay constraints, neither of the above solutions is practical. Thus, in our resource-constrained cross-layer setting with many states and actions (and many unknown parameters), we adopt a *model-free* reinforcement learning solution, which can be used to learn the optimal action-value function Q^* and optimal policy π^* online, without estimating the reward and transition probability functions. Specifically, we adopt a low complexity algorithm called Q-learning, which also has limited memory requirements (see Table 3.2 in Section 3.4.4). To the best of our knowledge, Q-learning has never been deployed in a multimedia system, let alone for cross-layer optimization.

3.4.2 Proposed Centralized Q-learning

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

$$\delta^n = \left[r^n + \gamma \max_{\boldsymbol{a}' \in \boldsymbol{\mathcal{A}}} Q^n(\boldsymbol{s}^{n+1}, \boldsymbol{a}')\right] - Q^n(\boldsymbol{s}^n, \boldsymbol{a}^n), \tag{3.13}$$

[3] Policy iteration or linear programming could also be used, but these solutions are also too complex.

$$Q^{n+1}(\boldsymbol{s}^n, \boldsymbol{a}^n) \leftarrow Q^n(\boldsymbol{s}^n, \boldsymbol{a}^n) + \alpha^n \delta^n, \tag{3.14}$$

where $\boldsymbol{s}^n$, $\boldsymbol{a}^n$, and $r^n = g_L^n - \sum_{l \in \mathcal{L}} \omega_l J_l^n$ are the state, performed action, and corresponding reward in time slot n, respectively; $\boldsymbol{s}^{n+1}$ is the resulting state in time slot $n+1$; $\boldsymbol{a}'$ is the *greedy action*[4] in state $\boldsymbol{s}^{n+1}$; δ^n is the so-called *temporal-difference* (TD) error [14]; 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 $n = 0$.

It is well known that if (i) the rewards and transition probability functions are stationary, (ii) all of the state-action pairs are visited infinitely often, and (iii) α^n satisfies the *stochastic approximation conditions*[5] $\sum_{n=0}^{\infty} (\alpha^n) = \infty$ and $\sum_{n=0}^{\infty} (\alpha^n)^2 < \infty$, then Q converges with probability 1 to Q^* [16]. Subsequently, the optimal policy can be found using (3.12).

During the learning process, it is not obvious what the best action is to take in each state. On the one hand, the optimal action-value function can be learned by randomly *exploring* the available actions in each state. Unfortunately, unguided randomized exploration cannot guarantee acceptable performance during the learning process. On the other hand, taking greedy actions, which *exploit* the available information in the action-value function $Q(\boldsymbol{s}, \boldsymbol{a})$, can guarantee a certain level of performance. Unfortunately, exploiting what is already known about the system prevents the discovery of new, better,

[4] A greedy action $\boldsymbol{a}^*$ is one that maximizes the current estimate of the action-value function, i.e. $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} \{Q(\boldsymbol{s}, \boldsymbol{a})\}$.

actions. To judiciously trade off exploration and exploitation, we use the so-called ε-*greedy* action selection method [14]:

ε-*greedy* action selection: With probability $1-\varepsilon$, take the greedy action that maximizes the action-value function, i.e. $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} \{Q(\boldsymbol{s},\boldsymbol{a})\}$; and, with probability ε, take an action randomly and uniformly over the action set.

We write $\boldsymbol{a} = \Phi_\varepsilon(Q,\boldsymbol{s})$ to show that $\boldsymbol{a}$ is an ε-*greedy* joint-action. Note that, if the ε-*greedy* action is taken from the optimal state-value function Q^*, then actions corresponding to the optimal policy will only be taken with probability $1-\varepsilon$; therefore, if ε does not decay to 0, then optimal performance will not be achieved.

Figure 3.2 illustrates the information exchanges required to deploy the centralized Q-learning algorithm in a two layer multimedia system during one time slot. In Figure 3.2, the top and bottom blocks represent the APP layer and joint OS/HW layer, respectively. The center block represents the centralized optimizer, which selects both layers' actions and updates the system's global action-value function Q. As we mentioned before, the centralized optimizer may be located at either the APP layer, the joint OS/HW layer, or a separate middleware layer. To best highlight the information exchanges, we illustrate the centralized optimizer as a separate middleware layer. Although we do not explicitly indicate this in Figure 3.2, the ET $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')$ is used by the block labeled "Update $Q(\boldsymbol{s},\boldsymbol{a})$," which performs the update step defined in (3.14).

---

[5] For example, $\alpha^n = 1/(n+1)$ satisfies the stochastic approximation conditions.

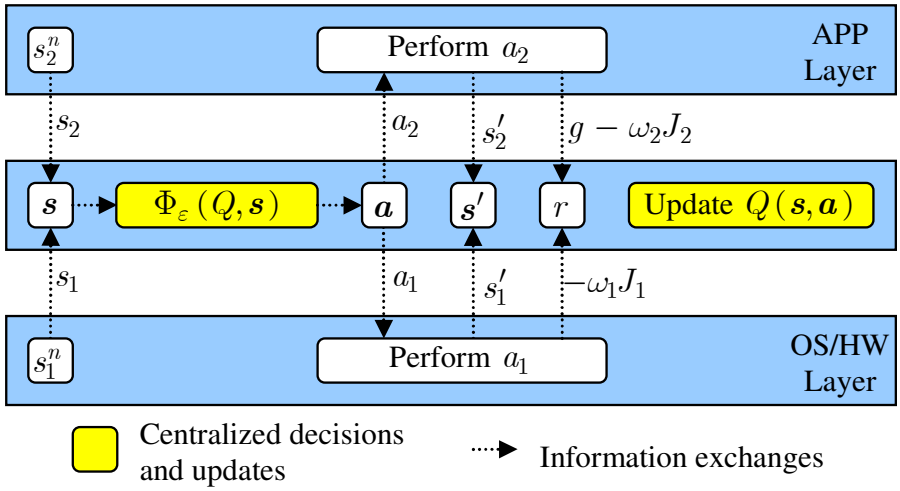


Figure 3.2. Information exchanges required to deploy the centralized Q-learning update step in one time slot.

3.4.3 Proposed Layered Q-learning

Thus far, we have discussed how the traditional Q-learning algorithm can be implemented by a centralized optimizer. As we discussed in the introduction, this centralized solution is most appropriate when a single manufacturer designs all of the system's layers. However, we are also interested in developing learning solutions that allow layers designed by different manufactures to act autonomously, but still cooperatively optimize the system's performance. This is a challenging problem in our *cross-layer* setting because the dynamics experienced at each layer may depend on the learning processes of the other layers. Hence, a key challenge is determining how to coordinate the autonomous learning processes of the various layers to enable them to successfully learn. In this section, we propose a *layered Q-learning* algorithm, which we derive by exploiting the

structure of the transition probability and reward functions in order to decompose the global action-value function into local action-value functions at each layer. Then, in subsection 3.4.4, we compare the computation, communication, and memory overheads incurred by the centralized and the proposed layered learning algorithms.

In this subsection, we first show how the action-value function can be decomposed by exploiting the structure of the considered cross-layer problem. Subsequently, we show how this decomposition leads to a layered Q-learning algorithm, which solves the cross-layer optimization online, in a *decentralized manner*, when the transition probability and reward functions are unknown a priori. For notational simplicity, we drop the time slot index n, and denote the next-state $\boldsymbol{s}^{n+1}$ using $\boldsymbol{s}'$.

3.4.3.1 Action-Value Function Decomposition

The proposed decomposition assumes that (i) each layer has access to the global state $\boldsymbol{s}$ in each time slot; (ii) the APP layer knows the number of states at the joint OS/HW layer, i.e. $|\mathcal{S}_{\text{OS}}|$; and, (iii) the joint OS/HW layer knows the number of states and actions at the APP layer, i.e. $|\mathcal{S}_{\text{APP}}|$ and $|\mathcal{A}_{\text{APP}}|$.

Given the additive reward function defined in (3.9) and the factored transition probability function defined in (3.8), we can rewrite the optimal action-value function defined in (3.11) as follows:

$$\begin{aligned} Q^*(\boldsymbol{s},\boldsymbol{a}) = \; & g_2(\boldsymbol{s},a_2) - \omega_1 J_1(s_1,a_1) - \omega_2 J_2(s_2,a_2) + \\ & \gamma \sum_{s_1' \in \mathcal{S}_1, s_2' \in \mathcal{S}_2} p_1(s_1' \mid s_1,a_1) p_2(s_2' \mid \boldsymbol{s},a_2) V^*\big(\boldsymbol{s}'\big), \end{aligned} \tag{3.15}$$

where, $V^*(\boldsymbol{s}') = \max_{\boldsymbol{a}' \in \boldsymbol{\mathcal{A}}} Q^*(\boldsymbol{s}', \boldsymbol{a}')$ is the optimal state-value for state $\boldsymbol{s}' = (s_1', s_2')$ (refer to Table 3.1 for a review of the notation).

Observing that $g_2(\boldsymbol{s}, a_2) - \omega_2 J_2(s_2, a_2)$ is independent of s_1' and that $\sum_{s_1' \in \mathcal{S}_1} p_1(s_1' \mid s_1, a_1) = 1$, we may rewrite the Bellman optimality equation in (3.15) as follows:

$$Q^*(\boldsymbol{s}, a_1, a_2) = -\omega_1 J_1(s_1, a_1) + \sum_{s_1' \in \mathcal{S}_1} \left[p_1(s_1' \mid s_1, a_1) \left(\begin{array}{l} g_2(\boldsymbol{s}, a_2) - \omega_2 J_2(s_2, a_2) + \\ \gamma \sum_{s_2' \in \mathcal{S}_2} p_2(s_2' \mid \boldsymbol{s}, a_2) V^*(s_1', s_2') \end{array} \right) \right]. \tag{3.16}$$

Given the global state $\boldsymbol{s} = (s_1, s_2)$ and the optimal state-value function V^*, the APP layer's action-value function can be expressed using the inner Bellman optimality equation:

$$Q_1^*(\boldsymbol{s}, a_2, s_1') = \left(\begin{array}{l} g_2(\boldsymbol{s}, a_2) - \omega_2 J_2(s_2, a_2) + \\ \gamma \sum_{s_2' \in \mathcal{S}_2} p_2(s_2' \mid \boldsymbol{s}, a_2) V^*(s_1', s_2') \end{array} \right), \ \forall a_2 \in \mathcal{A}_2 \text{ and } \forall s_1' \in \mathcal{S}_1 \tag{3.17}$$

which is independent of the immediate action at the joint OS/HW layer (i.e. a_1), but depends on the joint OS/HW layer's potential next-state s_1'. For each global state $\boldsymbol{s} = (s_1, s_2)$, the APP layer must compute the set $\left\{ Q_1^*(\boldsymbol{s}, a_2, s_1') : a_2 \in \mathcal{A}_2, s_1' \in \mathcal{S}_1 \right\}$ using (3.17). Then, given this set from the APP layer, the joint OS/HW layer's action value function can be expressed using the outer Bellman optimality equation in (3.16): i.e.,

$$Q^*(\boldsymbol{s}, a_1, a_2) = \left(\begin{array}{l} -\omega_1 J_1(s_1, a_1) + \\ \sum_{s_1' \in \mathcal{S}_1} p_1(s_1' \mid s_1, a_1) Q_1^*(\boldsymbol{s}, a_2, s_1') \end{array} \right), \ \forall a_1 \in \mathcal{A}_1 \text{ and } \forall a_2 \in \mathcal{A}_2. \tag{3.18}$$

Intuitively, $Q_1^*(\boldsymbol{s}, a_2, s_1')$ can be interpreted as the APP layer's estimate of the expected

discounted future rewards; however, there are two important differences between $Q_1^*(\boldsymbol{s}, a_2, s_1')$ and the centralized action-value function $Q^*(\boldsymbol{s}, \boldsymbol{a})$. First, $Q_1^*(\boldsymbol{s}, a_2, s_1')$ does not include the immediate reward at the joint OS/HW layer (i.e. $-\omega_1 J_1(s_1, a_1)$) because it is unknown to the APP layer; and, second, $Q_1^*(\boldsymbol{s}, a_2, s_1')$ depends on the joint OS/HW layer's next-state s_1' (instead of the expectation over of the next-states as it does in the centralized case) because the APP layer does not know the joint OS/HW layer's transition probability function. After receiving $Q_1^*(\boldsymbol{s}, a_2, s_1')$ from the APP layer, the joint OS/HW layer is able to "fill in" this missing information. Clearly, from the derivation, $Q^*(\boldsymbol{s}, a_1, a_2)$ at the joint OS/HW layer is equivalent to the centralized action-value function $Q^*(\boldsymbol{s}, \boldsymbol{a})$.

For more information about this decomposition, we refer the interested reader to our prior work [9], in which we use a similar decomposition to solve the cross-layer optimization *offline*, in a decentralized manner, under the assumption that the transition probability and reward functions are *known* a priori.

3.4.3.2 Layered Q-learning

Recall that the centralized Q-learning algorithm described in Section 3.4.2 requires a centralized manager to select actions for both of the layers. In contrast, the layered Q-learning algorithm that we propose in this subsection – made possible by the action-value function decomposition described above – enables each layer to autonomously select its own actions and to update its own action-value function. In the following, we discuss the

local information requirements for each layer, how each layer selects its local actions, and how each layer updates its local action-value function.

Local information: The proposed layered Q-learning algorithm requires that the APP layer maintains an estimate of the action-value function on the left-hand side of (3.17): i.e.,

$$\left\{ Q_1\left(\boldsymbol{s}, a_2, s_1'\right) : \boldsymbol{s} \in \boldsymbol{\mathcal{S}},\, a_2 \in \mathcal{A}_2,\, s_1' \in \mathcal{S}_1 \right\},$$

and that the joint OS/HW layer maintains an estimate of the action-value function on the left-hand side of (3.18): i.e.,

$$\left\{ Q\left(\boldsymbol{s}, a_1, a_2\right) : \boldsymbol{s} \in \boldsymbol{\mathcal{S}},\, \left(a_1, a_2\right) \in \boldsymbol{\mathcal{A}} \right\}.$$

At time slot $n = 0$, these tables can be initialized to 0, i.e. $Q_1^0\left(\boldsymbol{s}, a_2, s_1'\right) = 0$ and $Q^0\left(\boldsymbol{s}, a_1, a_2\right) = 0$. Thus, at initialization, the only information required by the APP layer about the joint OS/HW layer is the number of states at the joint OS/HW layer, i.e. $|\mathcal{S}_{\mathrm{OS}}|$; and, the only information required by the joint OS/HW layer about the APP layer is the number of states and actions at the APP layer, i.e. $|\mathcal{S}_{\mathrm{APP}}|$ and $|\mathcal{A}_{\mathrm{APP}}|$.

Local action selection: At run-time, given the current global state $\boldsymbol{s} = (s_1, s_2)$, the APP layer selects an ε-greedy action $a_2 \in \mathcal{A}_2$ as described in Section 3.4.2, but with the greedy action selected as follows:

$$\left(a_2^*, \hat{s}_1'\right) = \underset{a_2 \in \mathcal{A}_2,\ s_1' \in \mathcal{S}_1}{\arg\max} \left\{ Q_1\left(\boldsymbol{s}, a_2, s_1'\right) \right\}, \tag{3.19}$$

and, the joint OS/HW layer selects its ε-greedy action $a_1 \in \mathcal{A}_1$, but with the greedy

action selected as follows:

$$(a_1^*, \hat{a}_2) = \underset{a_1 \in \mathcal{A}_1, a_2 \in \mathcal{A}_2}{\arg\max} \{Q(\boldsymbol{s}, a_1, a_2)\}. \tag{3.20}$$

Note that the APP layer selects action a_2^* under the assumption that the joint OS/HW layer will select its action a_1^* to transition to the next-state $\hat{s}_1'$, which maximizes the APP layer's *estimated* discounted future rewards $Q_1(\boldsymbol{s}, a_2, s_1')$. Similarly, the joint OS/HW layer selects action a_1^* under the assumption that the APP layer will select action $\hat{a}_2$, which maximizes the joint OS/HW layer's *estimated* discounted future rewards $Q(\boldsymbol{s}, a_1, a_2)$. The message exchanges during the local learning update procedures (described immediately below) serve to "teach" each layer how they impact and are impacted by the other layer. Thus, through the learning process, the layers improve their assumptions about each other, which enables them to improve their greedy actions.

Local learning updates: After executing the ε-greedy action $\boldsymbol{a}^n = (a_1^n, a_2^n)$ in state $\boldsymbol{s}^n = (s_1^n, s_2^n)$, the system obtains the reward $r^n = g_2^n - \omega_1 J_1^n - \omega_2 J_2^n$ and transitions to state $\boldsymbol{s}^{n+1} = \left(s_1^{n+1}, s_2^{n+1}\right)$. Based on the experience tuple $\left(\boldsymbol{s}^n, \boldsymbol{a}^n, r^n, \boldsymbol{s}^{n+1}\right)$, each layer updates its action-value function as follows. First, the joint OS/HW layer must forward the scalar $V^n\left(s_1^{n+1}, s_2^{n+1}\right) = \underset{a_1' \in \mathcal{A}_1, a_2' \in \mathcal{A}_2}{\max} Q^n\left(s_1^{n+1}, s_2^{n+1}, a_1', a_2'\right)$ to the APP layer. Using this forwarded information, the APP layer can perform its action-value function update based on the form of (3.17): i.e.,

$$\delta_2^n = \left[g_2^n - \omega_2 J_2^n + \gamma V^n\left(s_1^{n+1}, s_2^{n+1}\right)\right] - Q_1^n\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right), \tag{3.21}$$

$$Q_1^{n+1}\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right) \leftarrow Q_1^n\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right) + \alpha_2^n \delta_2^n \,. \tag{3.22}$$

Then, given the scalar $Q_1^{n+1}\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right)$ from the APP layer and the APP layer's selected action a_2^n, the joint OS/HW layer can perform its action-value function update based on the form of (3.18) as follows:

$$\delta_1^n = \left[-\omega_1 J_1^n + Q_1^{n+1}\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right)\right] - Q^n\left(\boldsymbol{s}^n, a_1^n, a_2^n\right), \tag{3.23}$$

$$Q^{n+1}\left(\boldsymbol{s}^n, a_1^n, a_2^n\right) \leftarrow Q^n\left(\boldsymbol{s}^n, a_1^n, a_2^n\right) + \alpha_1^n \delta_1^n \,. \tag{3.24}$$

Note that, by using the layered Q-learning algorithm, the layers do not need to directly share their local rewards (i.e. the local learning updates only require the APP layer to know its local reward $g_2^n - \omega_2 J_2^n$ and for the joint OS/HW layer to know its local cost $\omega_1 J_1^n$). This is because of the coordinating message exchanges during the local learning update process (i.e. $V^n\left(s_1^{n+1}, s_2^{n+1}\right)$ is forwarded from the joint OS/HW layer to the APP layer and $Q_1^{n+1}\left(\boldsymbol{s}^n, a_2^n, s_1^{n+1}\right)$ is forwarded from the APP layer to the joint OS/HW layer).

Figure 3.3 illustrates the information exchanges required to deploy the layered Q-learning algorithm in a two layer multimedia system during one time slot. Although we do not explicitly indicate this in Figure 3.3, the ETs $\left(\boldsymbol{s}, a_2, g_2 - \omega_2 J_2, s_1'\right)$ and $\left(\boldsymbol{s}, a_1, -\omega_1 J_1, s_1'\right)$ are used by the blocks labeled "Update $Q_1\left(\boldsymbol{s}, a_2, s_1'\right)$" at the APP layer and "Update $Q\left(\boldsymbol{s}, a_1, a_2\right)$" at the joint OS/HW layer, respectively.

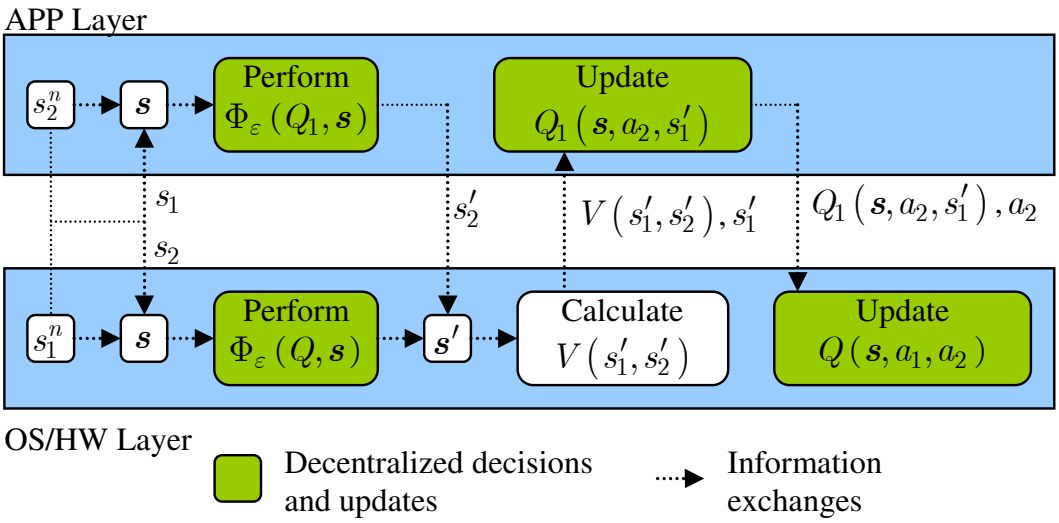


Figure 3.3. Information exchanges required to deploy the layered Q-learning update step in one time slot.

3.4.4 Computation, Communication, and Memory Overheads

In this subsection, we compare the computation, communication, and memory overheads associated with the centralized and layered Q-learning algorithms. Figure 3.2 and Figure 3.3 illustrate the exact information exchanges required for each algorithm and Table 3.2 lists the per time slot computation, communication, and memory overheads associated with each algorithm.

Interestingly, the layered Q-learning algorithm is more complex and requires more memory than the centralized algorithm. Note that, in our setting, $|\mathcal{S}_1| = |\mathcal{A}_1|$; hence, the layered Q-learning algorithm incurs approximately twice the computational and memory overheads as the centralized algorithm. The increased overheads can be interpreted as the cost of optimal decentralized learning (optimal in the sense that it performs equally as well as the centralized learning algorithm). Nevertheless, the proposed centralized and

layered Q-learning algorithms are far less complex and use less memory than the alternative learning solutions discussed in Section 3.4.1. We also observe that, in both cases, the communication overheads per time slot are $O(1)$ because they are independent of the size of the state and action sets at each layer; however, the precise number of messages exchanged depends on the deployed learning algorithm as illustrated in Figure 3.2 and Figure 3.3, and noted in Table 3.2.

Table 3.2. Comparison of computation, memory, and communication overheads (per time slot).

	Centralized Q-learning	Layered Q-learning
Computation overheads	Action Selection: $O(\lvert\boldsymbol{\mathcal{A}}\rvert)$ Update: $O(\lvert\boldsymbol{\mathcal{A}}\rvert)$	Action Selection: $O(\lvert\boldsymbol{\mathcal{A}}\rvert + \lvert\mathcal{S}_1 \times \mathcal{A}_2\rvert)$ Update: $O(\lvert\boldsymbol{\mathcal{A}}\rvert)$
Memory overheads	$O(\lvert\boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}}\rvert)$	$O(\lvert\boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}}\rvert + \lvert\boldsymbol{\mathcal{S}} \times \mathcal{A}_2 \times \mathcal{S}_1\rvert)$
Communication overheads	$O(1)$ (8 messages)	$O(1)$ (7 messages)

3.5 Accelerated Learning Using Virtual Experience

The large number of buffer states at the APP layer significantly limits the system's learning speed because the Q-learning update step defined in (3.14) 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-λ updates, and model-based algorithms such as Dyna and prioritized sweeping [14] [15]; 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 (our experimental results in Section 3.6.4 strongly support this conclusion).

In this subsection, we propose a new Q-learning variant, which can be used in addition to the abovementioned variants. Unlike conventional reinforcement learning algorithms, which assume that no a priori information is available about the environmental dynamics[6], the proposed algorithm exploits the *form* of the transition probability and reward functions (defined in Section 3.2) in order to update the action-value function for multiple *statistically equivalent*[7] state-action pairs in each time slot, including those that have never been visited. In our specific setting, we exploit the fact that the data unit arrival distribution $p_{\tilde{T}}(\tilde{t} \mid f, z, h)$ (defined in (3.5)) is conditionally independent of the current buffer state q^n (given z^n, h^n, and f^n). 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

[6] In conventional reinforcement learning [14] [15], the actors (i.e. the system layers in our setting) are assumed to have no a priori information about the form of the transition probability and reward functions beyond possible high-level structural knowledge about the factored transition and reward dynamics [17] [23]. In other words, in a conventional reinforcement learning framework, the additive decomposition structure of the reward function defined in (3.9) and the factored transition probability structure defined in (3.8) may be known a priori, but the actual form of the utility gain defined in (3.7) and the actual form of the transition probability function defined in (3.6) cannot be known a priori.

[7] We say that a state-action pair $(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}})$ is *statistically equivalent* to the pair $(\boldsymbol{s}, \boldsymbol{a})$ if $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) = p(\boldsymbol{s}' \mid \tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}),\ \forall \boldsymbol{s}' \in \boldsymbol{\mathcal{S}}$ and $R(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}})$ can be determined from $R(\boldsymbol{s}, \boldsymbol{a})$.

complexity. For ease of exposition, we discuss the algorithm in terms of the centralized system; however, it can be easily extended to work with the layered Q-learning algorithm proposed in Section 3.4.3.

Let $\boldsymbol{\sigma}^n = \left(\boldsymbol{s}^n, \boldsymbol{a}^n, r^n, \boldsymbol{s}^{n+1}\right)$ represent the ET at stage n, where $\boldsymbol{s}^n = (z^n, q^n, f^n)$, $\boldsymbol{a}^n = (u^n, h^n)$, $r^n = g_2^n - \omega_1 J_1^n - \omega_2 J_2^n$, and $\boldsymbol{s}^{n+1} = \left(z^{n+1}, q^{n+1}, f^{n+1}\right)$. Given q^n and the number of data unit arrivals $\left\lfloor \tilde{t}^n \right\rfloor = \left\lfloor \tilde{t}^n(z^n, h^n, f^n) \right\rfloor$, the next buffer state q^{n+1} can be easily determined from (3.3). By exploiting our partial knowledge about the system's dynamics, we can use the statistical information provided by $\left\lfloor \tilde{t}^n \right\rfloor$ 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 $\tilde{\boldsymbol{\sigma}}^n = \left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}, \tilde{r}, \tilde{\boldsymbol{s}}'\right) \in \boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ represent one virtual ET in the set of virtual ETs $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$. In order to be statistically equivalent to the actual ET, the virtual ETs in $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ must satisfy the following two conditions:

1. The data unit arrival distribution $p_{\tilde{T}}(\tilde{t} \mid f, z, h)$ defined in (3.5) must be the same for the virtual ETs as it is for the actual ET. In other words, the virtual operating frequency $\tilde{f}$, the virtual type $\tilde{z}$, and the virtual configuration $\tilde{h}$ must be the same as the actual operating frequency f^n, the actual type z^n, and the actual configuration h^n, respectively. This also implies that the virtual costs at the APP and joint OS/HW layers are the same as the actual costs at these layers, i.e. $\tilde{J}_1 = J_1^n$

and $\tilde{J}_2 = J_2^n$.

2. The next virtual buffer state $\tilde{q}' \in \mathcal{S}_q$ must be related to the current virtual buffer state $\tilde{q} \in \mathcal{S}_q$ through the buffer evolution equation defined in (3.3): i.e.,

$$\tilde{q}' = \min\left\{\left[\tilde{q} + \left\lfloor \tilde{t}^n \right\rfloor - 1\right]^+, N_q\right\}, \tag{3.25}$$

where $\left\lfloor \tilde{t}^n \right\rfloor$ is the number of data unit arrivals under the actual ET.

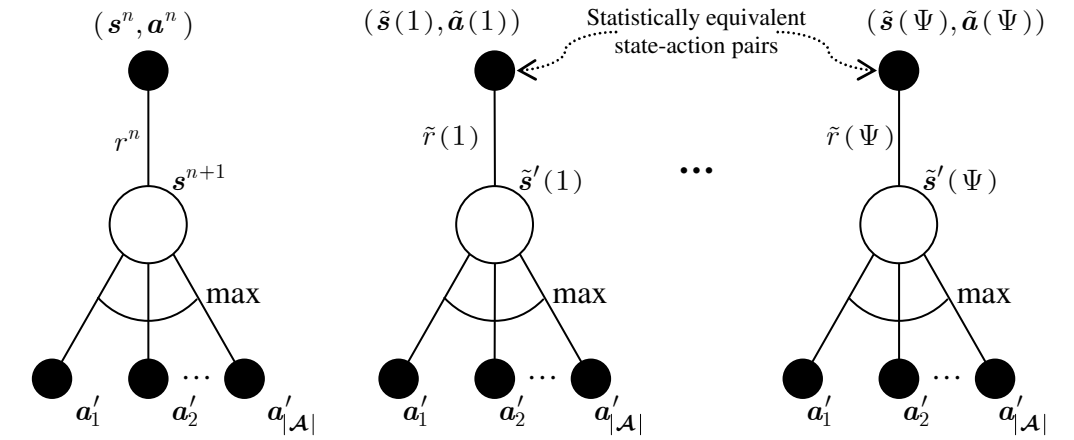
Any virtual ET that satisfies the two above conditions can have its reward determined using information embedded in the actual ET. Specifically, from the first condition, we know that the virtual ET's local costs are $\tilde{J}_1 = J_1^n$ and $\tilde{J}_2 = J_2^n$. Then, based on the second condition, we can compute the virtual ET's utility gain as $\tilde{g}_2 = 1 - \left(\frac{\tilde{q} + \left\lfloor \tilde{t}^n \right\rfloor - 1}{N_q}\right)^2$. Finally, the Q-learning update step defined in (3.14) can be performed on every virtual ET $\tilde{\boldsymbol{\sigma}}^n = \left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}, \tilde{r}, \tilde{\boldsymbol{s}}'\right) \in \boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ as if it is the actual ET.

Performing the Q-learning update step on every virtual ET in $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ incurs a computational overhead of approximately $O(|\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n) \times \boldsymbol{\mathcal{A}}|)$ in time slot n (note that $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ is typically as large as the buffer state set $\mathcal{S}_{\text{APP}}^{(q)} = \{q : i = 0, \ldots, N_q\}$ if you include the actual ET). Unfortunately, it may be impractical to incur such large overheads in every time slot, especially if the data unit granularity is small (e.g. one macroblock). Hence, in our experimental results in Section 3.6, we show how the learning performance is impacted by updating $\Psi \leq |\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)|$ virtual ETs in each time slot by selecting them randomly and uniformly from the virtual ET set $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$.

Table 3.3 describes the virtual ET based learning procedure in pseudo-code and Figure 3.4 compares the backup diagram [14] of the conventional Q-learning algorithm to the proposed algorithm with virtual ET updates. Importantly, because the virtual ETs are statistically equivalent to the actual ET, the virtual ET based learning algorithm is *not* an approximation of the Q-learning algorithm; in fact, the proposed algorithm accelerates Q-learning by exploiting our partial knowledge about the structure of the considered problem.

Table 3.3. Accelerated learning using virtual experience tuples.

1.	**Initialize** $Q(\boldsymbol{s},\boldsymbol{a})$ arbitrarily for all $(\boldsymbol{s},\boldsymbol{a}) \in \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}}$;
2.	**Initialize** state $\boldsymbol{s}^0$;
3.	**For** $n = 0,1,\ldots$
4.	Take action $\boldsymbol{a}^n$ using ε-greedy action selection on $Q(\boldsymbol{s}^n,\cdot)$;
5.	Obtain experience tuple $\boldsymbol{\sigma}^n = \left(\boldsymbol{s}^n,\boldsymbol{a}^n,r^n,\boldsymbol{s}^{n+1}\right)$ and record $\lfloor \tilde{t}^n \rfloor$;
6.	**For** all $\tilde{q} \in \mathcal{S}_{\text{APP}}^{(q)}$ % Generate the virtual ET set $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$
7.	$\tilde{q}' = \min\left\{\left[\tilde{q} + \lfloor \tilde{t}^n \rfloor - 1\right]^+, N_q\right\}$; % virtual ET next buffer state
8.	$\tilde{\boldsymbol{s}} = \left(\tilde{f},\tilde{z},\tilde{q}\right) = \left(f^n,z^n,\tilde{q}\right)$; % virtual ET state
9.	$\tilde{\boldsymbol{a}} = \left(\tilde{u},\tilde{h}\right) = \left(u^n,h^n\right)$; % virtual ET action
10.	$\tilde{r} = \left[1 - \left(\frac{\tilde{q} + \lfloor \tilde{t} \rfloor - 1}{N_q}\right)^2\right] - \omega_1 J_1^n - \omega_2 J_2^n$; % virtual ET reward
11.	$\tilde{\boldsymbol{s}}' = \left(\tilde{f}',\tilde{z}',\tilde{q}'\right) = \left(f^{n+1},z^{n+1},\tilde{q}'\right)$; % virtual ET next state
12.	$\tilde{\boldsymbol{\sigma}} = \left(\tilde{\boldsymbol{s}},\tilde{\boldsymbol{a}},\tilde{r},\tilde{\boldsymbol{s}}'\right) \in \boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$; % store virtual ET
13.	**For** Ψ virtual ETs $\tilde{\boldsymbol{\sigma}} \in \boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ % Update virtual ETs
14.	$\tilde{\delta} = \left[\tilde{r} + \gamma \max_{\boldsymbol{a}' \in \boldsymbol{\mathcal{A}}} Q(\tilde{\boldsymbol{s}}',\boldsymbol{a}')\right] - Q(\tilde{\boldsymbol{s}},\tilde{\boldsymbol{a}})$; % TD error for virtual ET $\tilde{\boldsymbol{\sigma}}^n$
15.	$Q(\tilde{\boldsymbol{s}},\tilde{\boldsymbol{a}}) \leftarrow Q(\tilde{\boldsymbol{s}},\tilde{\boldsymbol{a}}) + \alpha\tilde{\delta}$; % Q-learning update for virtual ET $\tilde{\boldsymbol{\sigma}}^n$



(a) Q-learning backup for actual ET $\boldsymbol{\sigma}^n = \left(\boldsymbol{s}^n, \boldsymbol{a}^n, r^n, \boldsymbol{s}^{n+1}\right)$

(b) Q-learning backup for Ψ virtual ETs $\tilde{\boldsymbol{\sigma}}^n = \left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}, \tilde{r}, \tilde{\boldsymbol{s}}'\right) \in \boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$

Figure 3.4. Backup diagrams. (a) Conventional Q-learning; (b) Q-learning with virtual updates. The virtual update algorithm applies a backup on the actual ET and an additional Ψ backups on virtual ETs in $\boldsymbol{\Sigma}(\boldsymbol{\sigma}^n)$ indexed by $1 \leq \psi \leq \Psi$.

3.6 Experiments

In this section, we compare the performance of the proposed learning algorithms against the performance of several existing algorithms in the literature using the cross-layer DMS described in Section 3.2. Table 3.4 details the parameters used in our DMS 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 parameters are listed in Table

3.4. We use a data unit granularity of one macroblock. As in [7], we assume that the power frequency function is of the form $P(f) = \kappa f^{\theta}$, where $\kappa \in \mathbb{R}_+$ and $\theta \in [1,3]$. Since real-time encoding is not possible with the available encoder, we set the data unit arrival rate to $\eta = 44$ DUs/sec, which corresponds to 1/9 frames per second.

For the simulations in Sections 3.6.2 and 3.6.2, we use stationary data traces, which we generate from the measured (non-stationary) data traces by assuming that the rate, distortion, and complexity samples are drawn from i.i.d. random variables with distributions equivalent to the distributions of the quantities over the entire non-stationary data traces. In Section 3.6.4, we perform simulations using both the measured (non-stationary) and generated (stationary) data traces.

Table 3.4. Simulation parameters (*Foreman* sequence, 30 Hz, CIF resolution, quantization parameter 24).

Layer	Parameter	Value
Application Layer (APP)	Data Unit Granularity	1 Macroblock
	Buffer State Set	$\mathcal{S}_q = \{0,\ldots,N_q\}$, $N_q = 50$ DUs
	Data Unit Type Set	$\mathcal{S}_q = \{z_1, z_2, z_3\}$ $z_1 = \mathrm{P},\ z_2 = \mathrm{B},\ z_3 = \mathrm{I}$
	Parameter Configuration	$\mathcal{A}_{\mathrm{APP}} = \{h_1, h_2, h_3\}$ h_1 : Quarter-pel MV, 8x8 block ME h_2 : Full-pel MV, 8x8 block ME h_3 : Full-pel MV, 16x16 block ME
	APP Cost Weight	$\omega_{\mathrm{APP}} = 22/1875$
	Rate-Distortion Lagrangian	$\lambda_{\mathrm{rd}} = 1/16$
	Data Unit Arrival Rate	$\eta = 44$ (Data Units/Sec)
Operating System / Hardware Layer (Joint OS/HW)	Power-Frequency Function	$P(f) = \kappa f^{\theta}$ $\kappa \in 1.5 \times 10^{-27}$ and $\theta = 3$.
	Operating Frequency Set	$\mathcal{A}_{\mathrm{OS}} = \{200, 400, 600, 800, 1000\}$ Mhz
	Joint OS/HW Cost Weight	$\omega_{\mathrm{OS}} = 22/125$

3.6.1 Evaluation Metrics

We deploy two different metrics to compare the performance of the various learning algorithms discussed in this chapter. The first metric is the *weighted estimation error*, which we define as

$$\text{Weighted estimation error} \triangleq \sum_{s \in \mathcal{S}} \boldsymbol{\mu}^*(s) \left| \frac{V^*(s) - V^n(s)}{V^*(s)} \right|, \tag{3.26}$$

where $\boldsymbol{\mu}^*$ is the stationary distribution under the optimal policy π^*, $V^*(s) = \max_a Q^*(s,a)$ is the optimal state-value function, and $V^n(s) = \max_a Q^n(s,a)$ is the state-value function estimate at stage n. In (3.26), we weight the estimation error $|V^*(s) - V^n(s)|$ by $\boldsymbol{\mu}^*(s)$ because the optimal policy will often only visit a small subset of the states, and the remaining states will never be visited; thus, if we were to estimate $V^*(s)$ while following the optimal policy, a *non-weighted* estimation error would not converge to 0, but the proposed weighted estimation error would.

Although the weighted estimation error tells us *how quickly an algorithm can learn the optimal policy*, it does not tell us *how well the algorithm performs while learning*. To see why this is, consider an (exaggerated) scenario in which $|V^*(s) - V^n(s)| = 0$, but the exploration rate in the greedy action selection procedure is $\varepsilon = 1$ (i.e. always explore). In this scenario, the greedy actions are optimal, but the random policy implemented by always exploring will be far from optimal. For this reason, we define a second evaluation metric using the *average reward*, which more accurately measures the system's performance. For a simulation of duration N, the average reward metric is

defined as

$$\text{Average reward} \triangleq \frac{1}{N}\sum_{n=0}^{N-1} r^n, \tag{3.27}$$

where r^n is the reward sample obtained at stage n. If the exploration rate ε decays appropriately to 0 as $n \to \infty$, then the average reward will increase as the weighted estimation error decreases. Determining an optimal decay strategy for ε is an important problem, but is beyond the scope of this chapter. We refer the interested reader to [31] [14] [15] for more information on the exploitation-exploration trade off.

We note that the weighted estimation error cannot be evaluated under non-stationary dynamics because the optimal state value function cannot be determined. Hence, for non-stationary dynamics, we rely solely on the average reward to evaluate the learning performance. Nevertheless, the rate at which the weighted estimation error converges under stationary dynamics for a particular learning algorithm is indicative of how well that algorithm will perform when the dynamics are non-stationary.

3.6.2 Single-Layer Learning Results

Before presenting our cross-layer learning results, we believe it is informative to see how well a single layer can learn (say layer l) when the other layer (say layer $-l$) deploys a static policy. The learning layer deploys a simple adaptation of the centralized Q-learning algorithm in which it knows the global state and global reward, but does not know the actions implemented by the other layer. The details of the single-layer learning algorithm are omitted due to space constraints; however, we note that it is essential that the learning layer knows the global state and global reward so that it can learn how it impacts the

other system layers. Otherwise, if the joint OS/HW layer is unaware of its impact on the application's delay and quality, for example, it will always selfishly minimize its own costs by operating at its lowest frequency and power, which can adversely impact the real-time application's performance.

For illustration, we assume that the static layer deploys its optimal local policy π^*_{-l} corresponding to the global optimal policy $\pi^* = (\pi^*_l, \pi^*_{-l})$; hence, layer l attempts to learn π^*_l. Figure 3.5 illustrates the optimal policies at the APP layer (π^*_2) and the joint OS/HW layer (π^*_1) for each buffer state and data unit type when the current operating frequency is $f = 600$ MHz. In Figure 3.5, the application actions (i.e. parameter configurations) are as defined in Table 3.4.

Figure 3.6 compares the cumulative average reward obtained using single-layer learning to the optimal achievable reward, and Table 3.5 shows the corresponding power, rate-distortion costs, utility gain, and buffer overflows, for a simulation of duration $N = 192,000$ time slots (approximately 485 frames drawn from the Foreman sequence, CIF resolution, by repeating the sequence from the beginning after 300 frames). We observe that the total average reward obtained when the joint OS/HW layer learns in response to the APP layer's optimal local policy π^*_2 is lower than when the APP layer learns in response to the joint OS/HW layer's optimal local policy π^*_1. This is because there are more actions to explore at the joint OS/HW layer (5 compared to 3), and the joint OS/HW layer's policy is more important to the system's overall performance than the APP layer's policy for the chosen simulation parameters (see Table 3.4). For instance,

given the action sets defined in Table 3.4, the joint OS/HW layer can significantly impact the application's experienced delay (i.e. a factor of 5 change from 200 MHz to 1000 Mhz), while the APP layer cannot (i.e. its actions impact the delay by less than a factor of 2). Consequently, when the APP layer learns in response to the joint OS/HW layer's optimal policy (Figure 3.5(b)), the system is initially better off than when the joint OS/HW layer learns in response to the APP layer's optimal policy (Figure 3.5(a)).

We note that, in Figure 3.6, the saturated performance of the APP and joint OS/HW layers' learning algorithms is due to the finite exploration probability (i.e. $\varepsilon > 0$ in the ε-greedy action selection procedure), which forces the layers to occasionally test suboptimal actions.

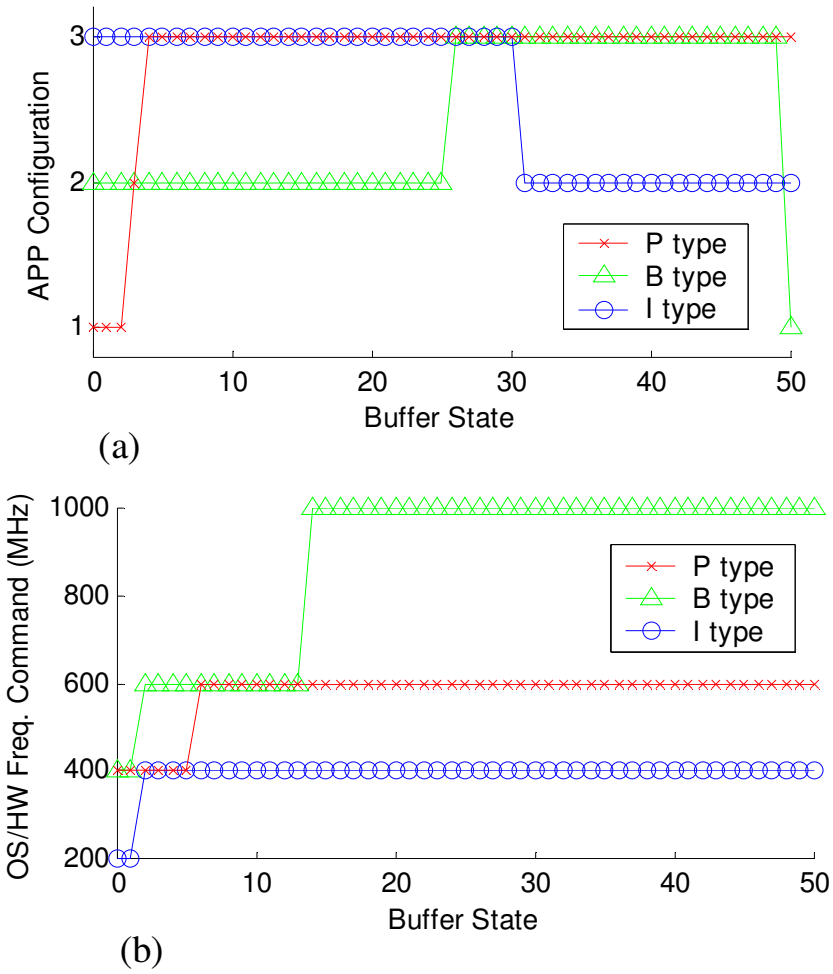


Figure 3.5. Optimal policies for each data unit type. (a) APP parameter configurations. (b) Joint OS/HW layer frequency command. The current operating frequency is set to be $f = 600$ Mhz.

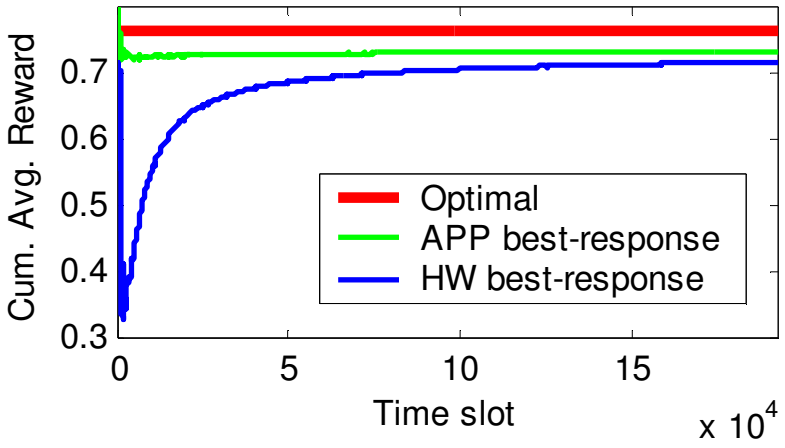


Figure 3.6. Cumulative average reward for single-layer learning with stationary trace ($N = 192,000$).

Table 3.5. Single-layer learning performance statistics for stationary trace ($N = 192,000$).

	APP Layer Learns	Joint OS/HW Layer Learns	Optimal
Avg. Reward	0.7337	0.7172	0.7631
Avg. Power (W)	0.2835	0.4512	0.2435
Avg. R-D	14.93	15.08	14.75
Avg. Gain	0.9588	0.9736	0.9790
No. Overflows	0	61	0

3.6.3 Cross-Layer Learning Results

In this subsection, we evaluate the performance of the centralized and layered Q-learning algorithms proposed in Section 3.4.2 and 3.4.3, respectively. Figure 3.7 compares the cumulative average reward obtained using layered Q-learning to the performance of the centralized learning algorithm, the optimal achievable reward, and the performance of the myopic learning algorithm deployed in the state-of-the-art cross-layer coordination

framework called GRACE-1[8] [5]; Figure 3.8 compares the weighted estimation error for the centralized and layered Q-learning algorithms; and, Table 3.6 shows the corresponding power, rate-distortion costs, utility gain, and buffer overflows. As in the previous simulations, the simulation duration is $N = 192,000$ time slots. Recall from Table 3.2 that the algorithms compared in this subsection have roughly the same computational complexity (note that the complexity of the GRACE-1 algorithm is roughly $O(|\mathcal{A}|)$ in our setting because it requires finding the action that will most closely meet a data unit's deadline).

As expected, the layered Q-learning algorithm performs equally as well as the centralized Q-learning algorithm, but also allows the layers to act autonomously through decentralized decisions and updates. Meanwhile, the myopic learning algorithm deployed by GRACE-1 performs very poorly in terms of overall reward. This is because, for each data unit, GRACE-1 myopically selects the lowest processing frequency that can successfully encode it before its deadline; consequently, the buffer quickly fills, which makes the system susceptible to delay deadline violations due to the time-varying workload (this is similar to what happens when using the "conventional gain" function described in Appendix A).

[8] In GRACE-1, the statistical cycle demand (complexity) of a video encoding application is measured using an exponential average of the 95th-percentile complexity of recently completed jobs (processed data units) in a sliding window. Based on the statistical cycle demand, GRACE-1 myopically encodes jobs at the lowest frequency which will meet the job's deadline. If slack remains after processing a job, then it is reclaimed for future jobs.

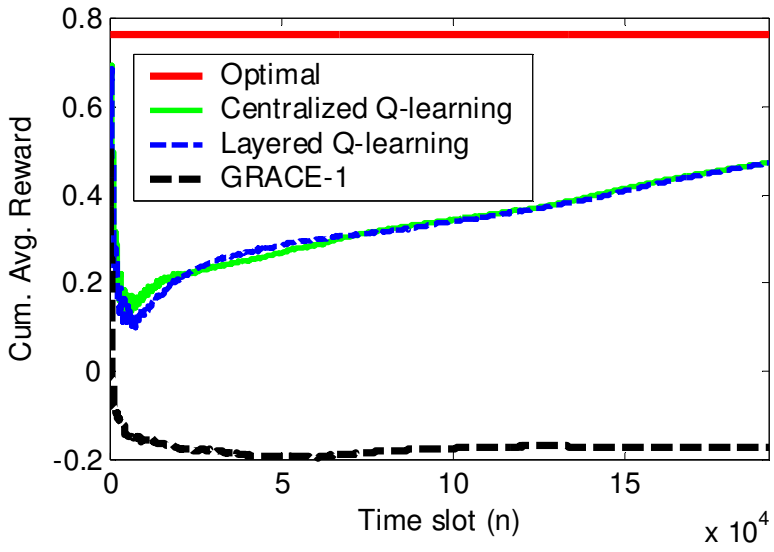


Figure 3.7. Cumulative average reward for cross-layer learning algorithms with stationary trace ($N = 192,000$).

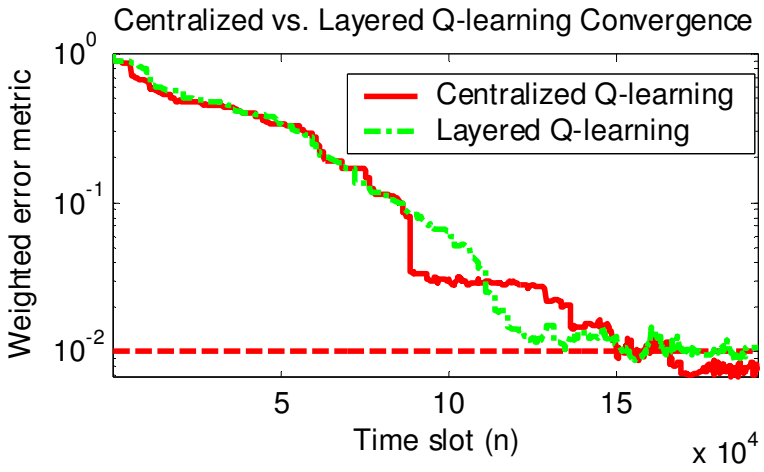


Figure 3.8. Weighted estimation error metric for cross-layer learning algorithms with stationary trace ($N = 192,000$).

Table 3.6. Cross-layer learning performance statistics for stationary trace ($N = 192,000$).

	Optimal	Centralized Q-learning	Layered Q-Learning	GRACE-1
Avg. Reward	0.7631	0.4727	0.4721	-0.1730
Avg. Power (W)	0.2435	0.3815	0.4061	0.4982
Avg. Rate-Distortion	14.75	15.00	14.99	15.30
Avg. Utility Gain	0.9790	0.7158	0.7195	0.0942
No. Overflows	0	1231	1251	3040

It is important to note that, even though the proposed learning algorithms outperform the existing myopic learning algorithm, there is still significant improvement to be made. In particular, the centralized and layered Q-learning algorithms learn too slowly because they only update one state-action pair in each time slot. Consequently, the number of buffer overflows and the power consumption are both higher, and the utility gain is lower, than in the optimal case. In the next subsection, we show that the learning performance (including the weighted estimation error and the average reward) can be dramatically improved by *smartly* updating multiple state-action pairs in each time slot.

3.6.4 Accelerated Learning with Virtual ETs

In this subsection, we compare the performance of the proposed virtual ET based learning algorithm (see Section 3.5) to an existing reinforcement learning algorithm called temporal-difference-λ learning[9] (i.e. *TD*(λ) [15]), which has comparable complexity (roughly $O((\Psi+1)|\mathcal{A}|)$ in each time slot for Ψ virtual ET or *TD*(λ) updates in addition

[9] In our implementation, *TD*(λ) applies the update step defined in (3.14) to the Ψ 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 previously visited states' "eligibility" as defined in [15]. Informally, the "eligibility" is the discounted frequency with which states have been visited in the past.

to the actual ET update). Recall that, in this subsection, we simulate the system with the measured (non-stationary) trace data and the generated (stationary) trace data.

Figure 3.9 illustrates the cumulative average reward achieved when we allow $\Psi \in \{0,1,15,30,45\}$ virtual ET or *TD*(λ) 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); Figure 3.10 compares the weighted estimation error for the same learning algorithms; and, Table 3.7 illustrates corresponding detailed simulation results.

Interestingly, the *TD*(λ) updates are completely ineffectual at improving the system's performance. In particular, it is obvious that increasing the number of *TD*(λ) updates in each time slot does not guarantee higher average rewards or lower weighted estimation errors. Meanwhile, increasing the number of virtual ET updates in each time slot dramatically improves the weighted estimation error metric and 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*(λ) learning algorithm can only learn about *previously visited state-action pairs*. The ability to learn about unvisited states is very important in our problem setting because after the buffer initially fills from early exploration (resulting in the initial drop in rewards illustrated in Figure 3.6, Figure 3.7, and Figure 3.9), the learning 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. $q = N_q - 1$) can be

extrapolated to a "near, near full" buffer state (e.g. $q = N_q - 2$) and so on. When using *TD*(λ) 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*(λ) (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.

One might expect that violating the stationarity assumption, which is widely used in the reinforcement learning literature (see [14][15][16]), would be disastrous to the system's performance. We observe in Figure 3.9 and Table 3.7, however, that the proposed virtual ET based learning algorithm is robust to the non-stationary dynamics in the measured data traces. This is primarily due to its ability to quickly propagate new information obtained at one state-action pair throughout the state-action space. From the data in Table 3.7, we observe that the percent difference between the reward obtained in the stationary case and the non-stationary case is less than 6%.

As in the single-layer learning case, the learning performance with virtual ETs saturates due to the finite exploration probability (i.e. $\varepsilon > 0$), which prevents the learning algorithm from converging to the optimal policy.

It is important to note that, in these experiments, we have assumed that the update complexity is negligible. This would be true if the updates were performed infrequently (e.g. per video frame) or if the updates were performed in parallel (e.g. using a vector processor); however, it is not a valid assumption for the very frequent and serial updates

deployed here (i.e. per video macroblock, without a vector processor). Hence, performing 15, 30, or 45 updates in each time slot is not reasonable, but performing 1 or 2 virtual ET updates is. Despite this technicality, we have shown the relative performance improvements that can be achieved by updating multiple virtual experience tuples in each time slot. An interesting trade off to investigate in future research is the impact of a coarser data unit granularity with a simultaneous increase in the number of virtual updates (such that the average learning complexity remains constant). Lastly, we note that the learning performance could be further improved (for the same number of virtual ET updates) by directing the virtual ET updates to states that are most likely to be visited in the near future (e.g. states near the observed next-state), instead of randomly updating the virtual experience tuples.

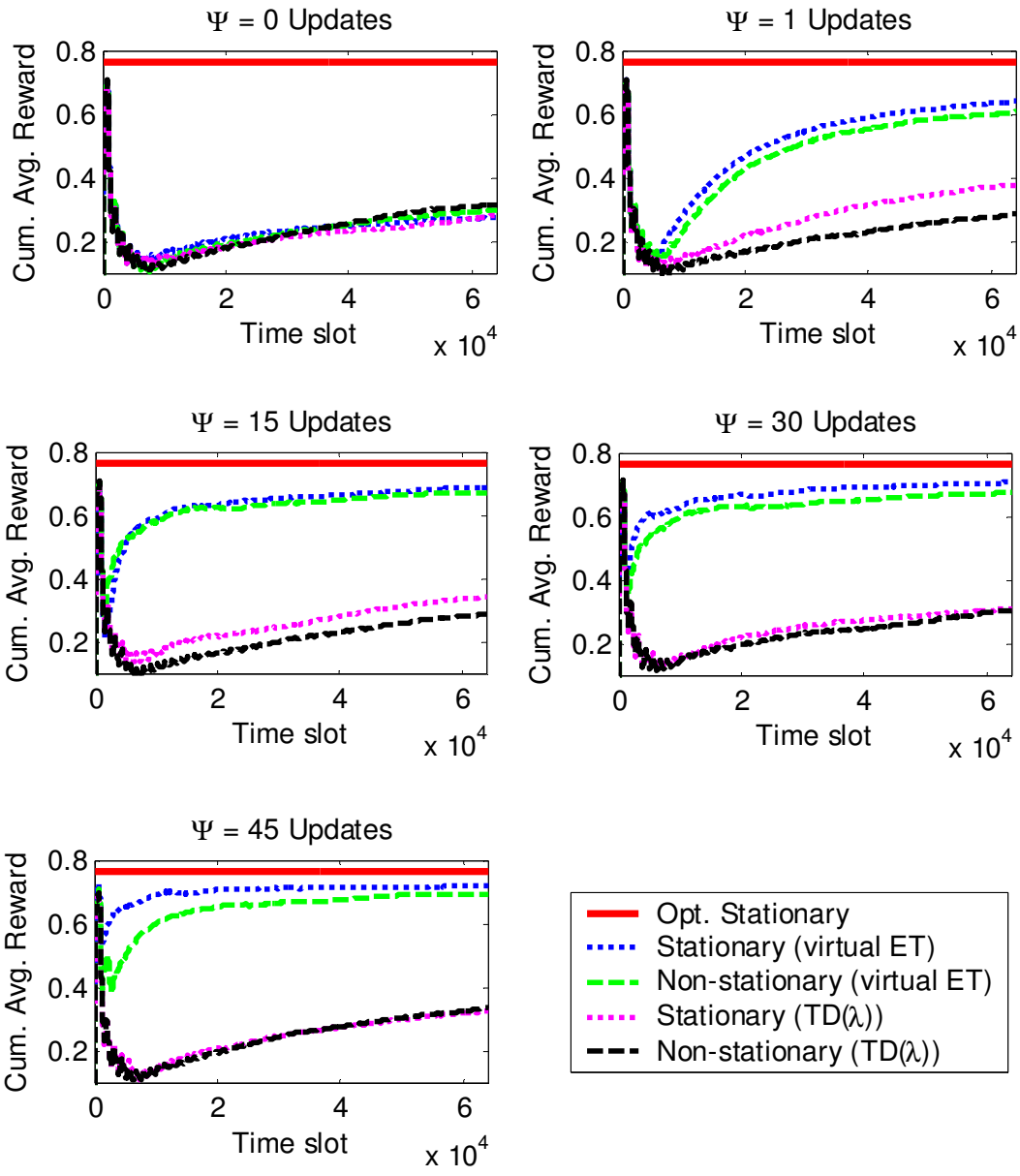


Figure 3.9. Cumulative average reward achieved with virtual ET and *TD*(λ) updates for a stationary trace and a non-stationary trace, compared to the optimal achievable reward under the stationary trace ($N = 64,000$).

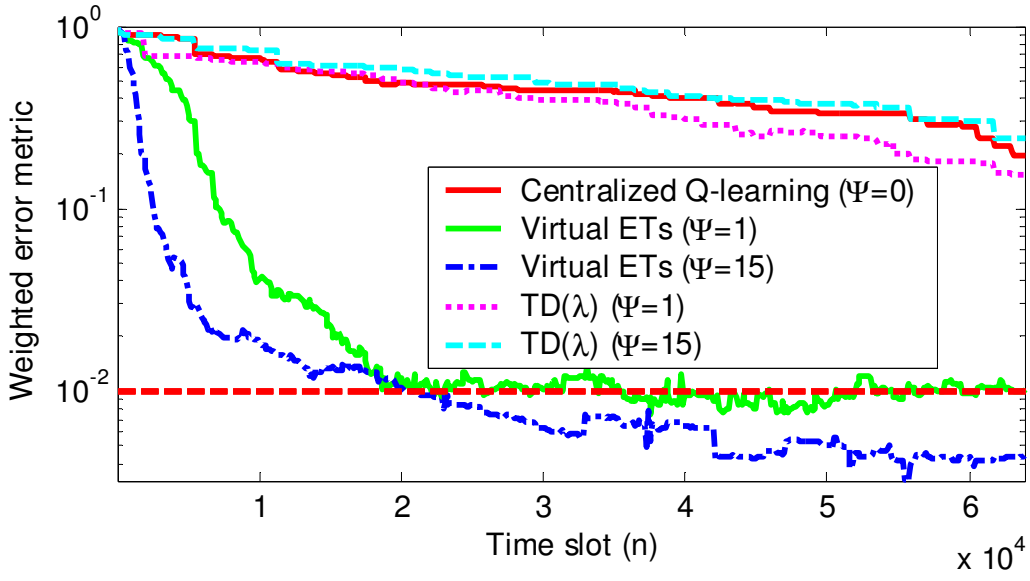


Figure 3.10. Weighted estimation error metric for virtual ETs and TD(λ) with stationary trace ($N = 64,000$).

Table 3.7. Virtual experience learning statistics for stationary (non-stationary) data trace outside (inside) parentheses ($N = 64,000$).

	Number of Virtual ET Updates				
	$\Psi = 0$	$\Psi = 1$	$\Psi = 15$	$\Psi = 30$	$\Psi = 45$
Avg. Reward	0.2785 (0.3015)	0.6414 (0.6078)	0.6893 (0.6710)	0.7076 (0.6759)	0.7200 (0.6944)
Avg. Power (W)	0.4212 (0.4632)	0.4164 (0.4314)	0.3817 (0.3864)	0.3432 (0.3657)	0.3108 (0.3381)
Avg. Rate-Distortion	15.00 (14.89)	14.92 (14.77)	15.06 (14.84)	14.93 (14.81)	14.87 (14.71)
Avg. Utility Gain	0.5287 (0.5577)	0.8898 (0.8570)	0.9332 (0.9131)	0.9432 (0.9140)	0.9492 (0.9264)
No. Overflows	1196 (1410)	1035 (1271)	502 (437)	226 (378)	103 (438)
	Number of *TD*(λ) Updates				
	$\Psi = 0$	$\Psi = 1$	$\Psi = 15$	$\Psi = 30$	$\Psi = 45$
Avg. Reward	0.2835 (0.3188)	0.3807 (0.2899)	0.3449 (0.2903)	0.314 (0.3088)	0.3287 (0.3388)
Avg. Power (W)	0.4494 (0.4707)	0.4649 (0.4732)	0.4551 (0.4672)	0.4530 (0.4644)	0.4544 (0.4875)
Avg. Rate-Distortion	15.07 (14.88)	15.14 (14.92)	15.14 (14.88)	15.03 (14.84)	15.15 (14.91)
Avg. Utility Gain	0.5395 (0.5763)	0.6402 (0.5483)	0.6026 (0.5472)	0.5703 (0.5647)	0.5866 (0.5997)
No. Overflows	1255 (1630)	1230 (1491)	1267 (1511)	1234 (1491)	1250 (1317)

3.7 Conclusion

Foresighted decisions are required to optimize the performance of resource-constrained multimedia systems. However, in practical settings, in which the system's dynamics are unknown a priori, efficiently learning the optimal foresighted decision policy can be a challenge. Specifically, selecting an improper learning model can lead to an unacceptably slow learning speed, incur excessive memory overheads, and/or adversely impact the application's real-time performance due to large computational overheads. In this chapter, we choose reinforcement learning to appropriately balance time-complexity, memory

complexity, and computational complexity. We propose a centralized and a layered reinforcement learning algorithm and extend these algorithms to exploit our partial knowledge of the system's dynamics. The proposed accelerated learning algorithm, which is based on updating multiple statistically equivalent state-action pairs in each time slot, allows us to judiciously tradeoff per-stage computational complexity and time-complexity (i.e. learning speed), while incurring low memory overheads.

In our experimental results, we verify that the layered learning solution performs equally as well as the centralized solution, which may be impractical to implement if the layers are designed by different manufacturers or operate at different time-scales. We also illustrate that our proposed foresighted learning algorithms outperform the myopic learning algorithm deployed in an existing state-of-the-art cross-layer optimization framework and that, by exploiting knowledge about the system's dynamics, we can significantly outperform an existing application-independent reinforcement learning algorithm that has comparable computational overheads.

In the long term, we believe this work can catalyze a shift in the design and implementation of DMSs (and systems in general) by enabling system layers (modules or components) to proactively reason and interact based on forecasts, as well as evolve, and become smarter, by learning from their interactions. A fruitful direction for future research is the application of the techniques introduced in this chapter to cross-layer optimization of the network protocol stack.

Appendix A: Utility Gain Function

In this appendix, we discuss the form of the utility gain function defined in (3.7). Conventionally, if a multimedia buffer does not overflow (i.e. the buffer constraints are not violated), then there is no penalty (see, for example, [21] [22]). Within the proposed MDP-based framework, where it is difficult to explicitly impose constraints because the dynamics are not known a priori, the conventional buffer model must be integrated into the system's reward function. Specifically, it can be integrated into the reward through a utility gain function of the form:

$$g_{\text{conv}}(\boldsymbol{s}, a_L) = \begin{cases} 1, & q + \lfloor \tilde{t} \rfloor - 1 \leq N_q \\ N_q - \left(q + \lfloor \tilde{t} \rfloor - 1\right), & \text{otherwise,} \end{cases}$$

which provides a reward of 1 if the buffer does not overflow, and a penalty proportional to the number of overflows otherwise.

In contrast to the proposed continuous utility gain function defined in (3.7), $g_{\text{conv}}(\boldsymbol{s}, a_L)$ is disjoint. As a result, the optimal foresighted policy obtained using $g_{\text{conv}}(\boldsymbol{s}, a_L)$ will initially fill the buffer rapidly in an attempt to minimize rate-distortion and power costs (because it is not penalized for filling the buffer) as illustrated in Figure 3.11(a). Subsequently, the policy will attempt to keep the buffer nearly full in order to balance the cost of overflow with the rate-distortion and power costs required to reduce the buffer's occupancy. Unfortunately, with the buffer nearly full, any sudden burst in the complexity of a data unit will immediately overflow the buffer as illustrated in Figure 3.11(b). In contrast, the proposed utility gain function is robust against bursts in

complexity because it encourages the buffer occupancy to remain low as illustrated in Figure 3.11(c,d).

The data in Table 3.8 shows that the proposed utility gain function not only prevents buffer overflows, but it also achieves comparable power consumption as the conventional utility gain function. Thus, we have verified that our choice of utility gain function is good. An added benefit of the proposed utility gain function is that it aids in the learning process. This is because actions are *immediately* rewarded (or penalized) based on how they impact the buffer state.

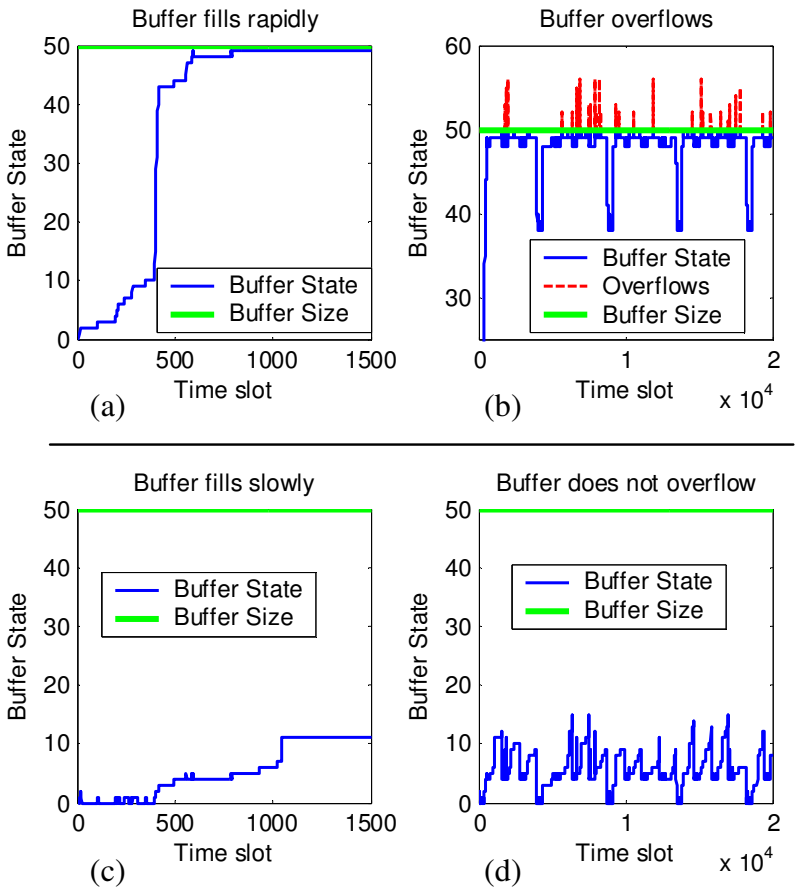


Figure 3.11. Buffer evolution in $N = 20,000$ time slot simulation. (a) Conventional utility gain results in the buffer filling rapidly; (b) Conventional utility gain leads to overflows; (c) Proposed utility gain keeps the buffer occupancy low; (d) Proposed utility gain prevents overflows.

Table 3.8. Performance statistics using the conventional utility gain function and the proposed utility gain function.

	Form of the utility gain	
	Conventional	Proposed
Avg. Power (W)	0.2421	0.2427
Avg. Rate-Distortion	14.95	14.84
No. Overflows	394	0

References

[1] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated power management for video streaming to mobile handheld devices," *Proc. of the 11th ACM international conference on Multimedia*, pp. 582-591, 2003.

[2] R. Cornea, S. Mohapatra, N. Dutt, A. Nicolau, N. Venkatasubramanian, "Managing Cross-Layer Constraints for Interactive Mobile Multimedia," *Proc. of the IEEE Workshop on Constraint-Aware Embedded Software*, 2003.

[3] 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," p. 218a, 19th *IEEE International Parallel and Distributed Processing Symposium*, 2005.

[4] 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.

[5] 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.

[6] 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.

[7] Z. He, W. Cheng, X. Chen, "Energy minimization of portable video communication devices based on power-rate-distortion optimization," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 18, no. 5, May 2008.

[8] 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.

[9] N. Mastronarde and M. van der Schaar, "Towards a general framework for cross-layer decision making in multimedia systems," *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 19, no. 5, pp. 719-732, May 2009.

[10] S. Irani, G. Singh, S. K. Shukla, R. K. Gupta, "An overview of the competitive and adversarial approaches to designing dynamic power management strategies," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 13, no. 12, pp. 1349-1361, Dec. 2005.

[11] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. on computer-aided design of integrated circuits*, vol. 18, no. 6, June 1999.

[12] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. on Computers*, vol. 51, no. 11, Nov. 2002.

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

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

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: a survey,” *Journal of Artificial Intelligence Research 4*, pp. 237-285, May 2005.

[16] C. J. C. H. Watkins and P. Dayan, “Technical Note: Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279-292, May 1992.

[17] S. Russell and A. L. Zimdars, “Q-decomposition for reinforcement learning agents,” in *Proc. of the International Conference on Machine Learning*, pp. 656-663, 2003.

[18] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, “Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs,” *Proc. of the 4th International Conference on Autonomic Computer*, 2007.

[19] Omer F. Rana and Jeffrey O. Kephart, "Building Effective Multivendor Autonomic Computing Systems," *IEEE Distributed Systems Online*, vol. 7, no. 9, 2006, art. no. 0609-o9003.

[20] D. S. Turaga and T. Chen, “Hierarchical modeling of variable bit rate video sources,” *Packet Video Workshop*, May 2001.

[21] 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.

[22] E. Akyol and and M. van der Schaar, "Complexity Model Based Proactive Dynamic Voltage Scaling for Video Decoding Systems," *IEEE Trans. Multimedia*, vol. 9, no. 7, pp. 1475-1492, Nov. 2007.

[23] S. P. Sanner, "First-order decision-theoretic planning in structured relational environments," Ph.D. Thesis, University of Toronto, 2008.

[24] N. Nahrstedt, D. Xu, D. Wichadakul, and B. Li QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments, *IEEE Communications Magazine*, 2001.

[25] S. Mohapatra and N. Venkatasubramanian, "PARM: Power-aware reconfigurable middleware," *Proc. 23rd Internat. Conf. on Distributed Computing Systems*, 2003.

[26] 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.

[27] P. Goyal, X. Guo, H.M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," *Usenix 2nd Symposium on OS Design and Implementation*, pp. 107-122, 1996.

[28] P. A. Dinda, and D. R. O'Hallaron, "An evaluation of linear models for host load prediction," *Proc. IEEE Internat. Sympos. High Perf. Distrib. Comput.*, pp. 87-96, Aug. 1999.

[29] Y. Andreopoulos and M. van der Schaar, "Adaptive Linear Prediction for Resource Estimation of Video Decoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, pp. 751-764, June 2007.

[30] F. Fu and M. van der Schaar, "A Systematic Framework for Dynamically Optimizing Multi-User Video Transmission," Technical Report. Available online at: http://arxiv.org/abs/0903.0207

[31] A. Schaerf, Y. Shoham, M. Tennenholtz, “Adaptive load balancing: a study in multi-agent learning,” *Journal of Artificial Intelligence Research*, vol. 2, 1995.

Chapter 4

Fast Reinforcement Learning for Energy-Efficient Wireless Communication

We consider the problem of energy-efficient point-to-point transmission of delay-sensitive data (e.g. multimedia data) over a fading channel. Existing research on this topic utilizes either physical-layer centric solutions, namely power-control and adaptive modulation and coding (AMC), or system-level solutions based on dynamic power management (DPM); however, there is currently no rigorous and unified framework for simultaneously utilizing both physical-layer centric and system-level techniques to achieve the minimum possible energy consumption, under delay constraints, in the presence of stochastic and a priori unknown traffic and channel conditions. In this chapter, we propose such a framework. We formulate the stochastic optimization problem as a Markov decision process (MDP) and solve it online using reinforcement learning. The advantages of the proposed online method are that (i) it does not require a priori knowledge of the traffic arrival and channel statistics to determine the jointly optimal power-control, AMC, and DPM policies; (ii) it exploits partial information about the system so that less information needs to be learned than when using conventional reinforcement learning algorithms; and (iii) it obviates the need for action exploration,

which severely limits the adaptation speed and run-time performance of conventional reinforcement learning algorithms.

4.1 Introduction

Delay-sensitive communication systems often operate in dynamic environments where they experience time-varying channel conditions (e.g. fading channel) and dynamic traffic loads (e.g. variable bit-rate). In such systems, the primary concern has typically been the reliable delivery of data to the receiver within a tolerable delay. Increasingly, however, battery-operated mobile devices are becoming the primary means by which people consume, author, and share delay-sensitive content (e.g. real-time streaming of multimedia data, videoconferencing, gaming etc.). Consequently, energy-efficiency is becoming an increasingly important design consideration. To balance the competing requirements of energy-efficiency and low delay, fast learning algorithms are needed to quickly adapt the transmission decisions to the time-varying and a priori unknown traffic and channel conditions.

Existing research that addresses the problem of energy-efficient wireless communications can be roughly divided into two categories: physical (PHY) layer-centric solutions such as power-control and adaptive modulation and coding (AMC); and system-centric solutions such as dynamic power management (DPM).[1] Although these techniques differ significantly, they can all be used to tradeoff delay and energy to increase the lifetime of battery-operated mobile devices [2] [3] [8].

[1] DPM enables system components such as the wireless network card to be put into low-power states when they are not needed [3]-[6]. We discuss this in more detail in Section 4.2.2.

PHY-centric solutions: A plethora of existing PHY-centric solutions focus on optimal single-user power-control (also known as *minimum energy transmission* or *optimal scheduling*) with the goal of minimizing transmission power subject to queuing delay constraints (e.g. [1] [2]). It is well known that transmitting with more power in good channel states, and with less power in poor channel states, maximizes throughput under a fixed energy budget. For this reason, [1] and [2] use dynamic programming, coupled with a stochastic fading channel model, to determine the optimal power-control policy. Unfortunately, the aforementioned solutions require statistical knowledge of the underlying dynamics (i.e. the channel state and traffic distributions), which is typically not available in practice. When this information is not available, only heuristic solutions, which cannot guarantee optimal performance, are provided. For example, in [2], a heuristic power-control policy is proposed that is only optimal for asymptotically large buffer delays. Moreover, [1] and [2] are information-theoretic in nature, so they ignore transmission errors, which commonly occur in practical wireless transmission scenarios.

Other PHY-centric solutions are based on adaptive modulation, adaptive coding, or AMC. Although these techniques are typically used to tradeoff throughput and error-robustness in fading channels [22], they can also be exploited to tradeoff delay and energy as in [8], where they are referred to as dynamic modulation scaling, dynamic code scaling, and dynamic modulation-code scaling, respectively. Offline and online techniques for determining optimal scaling policies are proposed in [8], however, these techniques cannot be extended in a manner that tightly integrates PHY-centric and system-level power management techniques.

Although PHY-centric solutions are effective at minimizing transmission power, they ignore the fact that it costs power to keep the wireless card on and ready to transmit; therefore, a significant amount of power can be wasted even when there are no packets being transmitted. System-level solutions address this problem.

System-level solutions: System-level solutions rely on DPM, which enables system components such as the wireless network card to be put into low-power states when they are not needed [3]-[6]. A lot of work has been done on DPM, ranging from rigorous work based on the theory of Markov decision processes [3]-[5], to low-complexity work based on heuristics [6]. In [3], the optimal DPM policy is determined offline under the assumption that the traffic arrival distribution is known a priori. In [5], an online approach using maximum likelihood estimation to estimate the traffic arrival distribution is proposed. This approach requires using the complex value iteration algorithm to update the DPM policy to reflect the current estimate of the traffic distribution. In [4], a supervised learning approach is taken to avoid the complex value iteration algorithm. However, this approach incurs large memory overheads because it requires many policies to be computed offline and stored in memory to facilitate the online decision making process. Unfortunately, due to their high computational and memory complexity, the aforementioned online approaches are impractical for optimizing more complex, resource-constrained, and delay-sensitive communication systems.

Our contributions are as follows:

- **Unified power management framework:** We propose a rigorous and unified framework, based on Markov decision processes (MDPs) and reinforcement

learning (RL), for simultaneously utilizing both PHY-centric and system-level techniques to achieve the minimum possible energy consumption, under delay constraints, in the presence of stochastic traffic and channel conditions. This is in contrast to existing work that only utilizes power-control [1] [2], AMC [8], or DPM [3]-[6] to manage power.[2]

- **Generalized post-decision state:** We propose a decomposition of the (offline) value iteration and (online) RL algorithms based on factoring the system's dynamics into a priori known and a priori unknown components. This is achieved by generalizing the concept of a *post-decision state* (PDS, or *afterstate* [11]), which is an intermediate state that occurs after the known dynamics take place but before the unknown dynamics take place. Similar decompositions have been used for modeling games such as tic-tac-toe and backgammon [11], for dynamic channel allocation in cellular telephone systems [11], and for delay-constrained scheduling over fading channels (i.e. power-control) [12] [16]. However, we provide a more general formulation of the PDS concept than existing literature. Specifically, existing literature introduces the PDS concept for very specific problem settings, where the PDS is a deterministic function of the current state and action, and where the cost function is assumed to be known. In general, however, the PDS can be a non-deterministic function of the current state and action, and it can be used in any MDP in which it is possible to factor the transition probability and cost functions into known and unknown components. The advantages of the proposed PDS

[2] More accurately, prior research on power-control implicitly considers adaptive coding because it assumes that optimal codes, which achieve information-theoretic capacity, are used.

learning algorithm are that it exploits partial information about the system, so that less information needs to be learned than when using conventional RL algorithms and, under certain conditions, it obviates the need for action exploration, which severely limits the adaptation speed and run-time performance of conventional RL algorithms.

- **Virtual experience:** Unlike existing literature, we take advantage of the fact that the unknown dynamics are independent of certain components of the system's state. We exploit this property to perform a batch update on multiple PDSs in each time slot. We refer to this batch update as virtual experience learning. Prior to this work, it was believed that the PDS learning algorithm must necessarily be performed one state at a time because one learns only about the current state being observed, and can, therefore, update only the corresponding component [12]. Importantly, our experimental results illustrate that virtual experience dramatically improves performance over "one state at a time" PDS learning.
- **Application of reinforcement learning to dynamic power management (DPM):** We believe that this chapter is the first to apply RL to the DPM problem. This is non-trivial because naïve application of RL can lead to very poor performance. This is because conventional RL solutions (e.g. Q-learning) require frequent action exploration, which lead to significant power and delay penalties when a suboptimal power management action is taken. Using PDS learning in conjunction with virtual experience learning, however, obviates the need for action exploration and allows the algorithm to quickly adapt to the dynamics and learn the optimal policy.

The remainder of the chapter is organized as follows. In Section 4.2, we introduce the system model and assumptions. In Section 4.3, we describe the transmission buffer and traffic models. In Section 4.4, we formulate the power management problem as an MDP. In Section 4.5, we discuss how to solve the problem online using RL. In Section 4.6, we present our simulation results. Section 4.7 concludes the chapter.

4.2 Preliminaries

In this section, we introduce the time-slotted system model used in this chapter. We assume that time is slotted into discrete-time intervals of length Δt such that the nth time slot is defined as the time interval $[n\Delta t,(n+1)\Delta t)$. Transmission and power management decisions are made at the beginning of each interval and the system's state is assumed to be constant throughout each interval. Figure 4.1 illustrates the considered wireless transmission system. We describe the deployed PHY-centric power management techniques in subsection 4.2.1 and the deployed system-level power management technique in subsection 4.2.2.

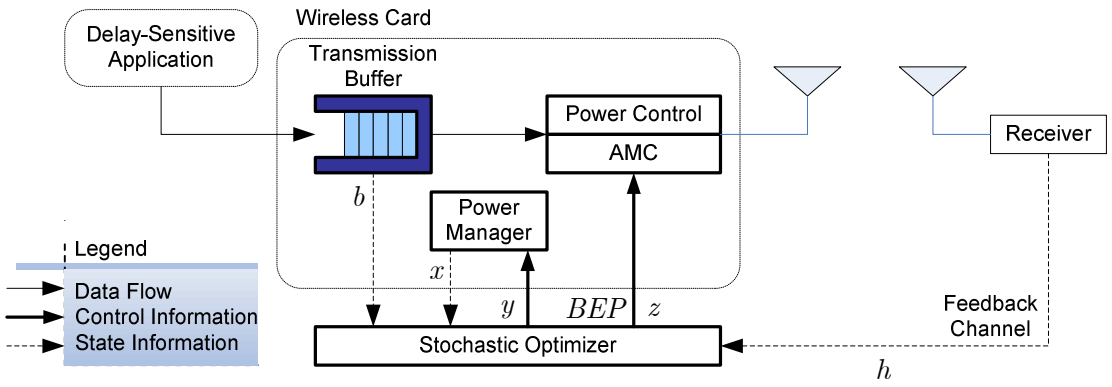


Figure 4.1. Wireless transmission system. The components outlined in bold are the focus of this chapter.

4.2.1 Physical Layer: Adaptive Modulation and Power-Control

We consider the case of a frequency non-selective channel, where h^n denotes the fading coefficient over the point-to-point link (i.e. from the transmitter to the receiver) in time slot n. As in [1], [2], [12], and [13], we assume that the channel state is constant for the duration of a time slot, that it can be estimated perfectly, and that the sequence of channel states $\{h^n \in \mathcal{H} : n = 0,1,\ldots\}$ can be modeled as a Markov chain with transition probabilities $p^h\left(h' \mid h\right)$.

The proposed framework can be applied to any modulation and coding schemes. Our only assumptions are that the bit-error probability (BEP) at the output of the maximum likelihood detector of the receiver, denoted by BEP^n, and the transmission power, denoted by P_{tx}^n, can be expressed as

$$BEP^n = BEP(h^n, P_{\text{tx}}^n, z^n) \text{ and} \tag{4.1}$$

$$P_{\text{tx}}^n = P_{\text{tx}}(h^n, BEP^n, z^n), \tag{4.2}$$

where z^n is the *packet throughput* in packets per time slot and each packet has size ℓ (bits). Assuming independent bit-errors, the packet loss rate (PLR) for a packet of size ℓ

can be easily computed from the BEP as $PLR^n = 1 - (1 - BEP^n)^\ell$.

We will use the packet throughput as a decision variable in our optimization problem. We are then free to either select the transmission power, which determines the BEP through (4.1), or select the BEP, which determines the transmission power through (4.2). Without loss of generality, we select the BEP as our decision variable.

For illustration, throughout this chapter we assume that (4.1) and (4.2) are known; however, the proposed learning framework is general and can be directly extended to the case when (4.1) and (4.2) are unknown a priori. We also select a fixed coding scheme for simplicity; however, the proposed framework can still be applied using adaptive modulation and coding with an appropriate modification to the BEP and transmission power functions. Note that if both modulation and coding can be adapted, then (4.1) and (4.2) are often unattainable analytically. Thus, if (4.1) and (4.2) are assumed to be known, it is necessary to resort to models based on fitting parameters to measurements as in [19]-[21].

4.2.2 System-Level: Dynamic Power Management Model

In addition to adaptive modulation and power-control, which determine the active transmission power, we assume that the wireless card can be put into low power states to reduce the power consumed by the wireless card's electronic circuits. Specifically, the wireless card can be in one of two power management states in the set $\mathcal{X} = \{\text{on}, \text{off}\}$ and can be switched on and off using one of the two power management actions in the set

$\mathcal{Y} = \{\text{s\_on}, \text{s\_off}\}$.[3] As in [3], the notation s_on and s_off should be read as "switch on" and "switch off," respectively.

If the channel is in state h, the wireless card is in state x, the maximum BEP is BEP^n, the power management action is y, and the packet throughput is z, then the required power is

$$\rho([h,x], BEP, y, z) = \begin{cases} [P_{\text{on}} + P_{\text{tx}}(h^n, BEP^n, z^n)], & \text{if } x = \text{on}, y = \text{s\_on} \\ P_{\text{off.}}, & \text{if } x = \text{off}, y = \text{s\_off} \\ P_{\text{tr}}, & \text{otherwise}, \end{cases} \tag{4.3}$$

where P_{tx} (watts) is the transmission power defined in (4.2), P_{on} and P_{off} (watts) are the power consumed by the wireless card in the "on" and "off" states, respectively, and P_{tr} (watts) is the power consumed when it transitions from "on" to "off" or from "off" to "on". Similar to [3], we assume that $P_{\text{tr}} \geq P_{\text{on}} > P_{\text{off}} \geq 0$ such that there is a large penalty for switching between power states, but remaining in the "off" state consumes less power than remaining in the "on" state.

As in [3], we model the sequence of power management states $\{x^n \in \mathcal{X} : n = 0,1,\ldots\}$ as a controlled Markov chain with transition probabilities $p^x(x' \mid x, y)$. Let $\mathbf{P}^x(y)$ denote the transition probability matrix conditioned on the power management action y, such that $\mathbf{P}^x(y) = \left[p^x(x' \mid x, y)\right]_{x,x'}$. Due to the high level of abstraction of the model, it is possible that there is a non-deterministic delay associated with the power management state transition such that

[3] This can be easily extended to multiple power management states as in [4], but we only consider two for ease of exposition.

$$\mathbf{P}^{x}(\mathrm{s\_on}) = \begin{array}{c} \\ \text{on} \\ \text{off} \end{array} \begin{array}{c} \begin{array}{cc} \text{on} & \text{off} \end{array} \\ \begin{pmatrix} 1 & 0 \\ \theta & 1-\theta \end{pmatrix} \end{array}$$
$$\mathbf{P}^{x}(\mathrm{s\_off}) = \begin{array}{c} \\ \text{on} \\ \text{off} \end{array} \begin{array}{c} \begin{array}{cc} \text{on} & \text{off} \end{array} \\ \begin{pmatrix} 1-\theta & \theta \\ 0 & 1 \end{pmatrix} \end{array} \tag{4.4}$$

where the row and column labels represent the current (x^n) and next (x^{n+1}) power management states, respectively, and $\theta \in (0,1]$ (resp. $1-\theta$) is the probability of a successful (resp. an unsuccessful) power state transition. For simplicity of exposition, we will assume that the power state transition is deterministic (i.e. $\theta = 1$); however, our framework applies to the case with $\theta < 1$ (with θ known or unknown). We note that the packet throughput z can be non-zero only if $x = \text{on}$ and $y = \mathrm{s\_on}$; otherwise, the packet throughput $z = 0$.

4.3 Transmission Buffer and Traffic Model

We assume that the transmission buffer is a first-in first-out (FIFO) queue. The source injects l^n packets into the transmission buffer in each time slot, where l^n has distribution $p^l(l)$. Each packet is of size L bits and the arrival process $\{l^n : n = 0,1,\ldots\}$ is assumed to be independent and identically distributed (i.i.d) with respect to the time slot index n. The arriving packets are stored in a finite-length buffer, which can hold a maximum of B packets. The buffer state $b \in \mathcal{B} = \{0,1,\ldots,B\}$ evolves recursively as follows:

$$\begin{aligned} b^0 &= b_{\text{init}} \\ b^{n+1} &= \min(b^n - f^n(BEP^n, z^n) + l^n, B), \end{aligned} \tag{4.5}$$

where b_{init} is the initial buffer state and $f^n(BEP^n, z^n) \le z^n$ is the *packet goodput* in

packets per time slot (i.e. the number of packets transmitted without error), which depends on the throughput z^n and the BEP BEP^n. Recall that the packet throughput z, and therefore the packet goodput f, can be non-zero only if $x = \text{on}$ and $y = \text{s\_on}$; otherwise, they are both zero (i.e. $z = f = 0$). Also, note that the packets arriving in time slot n cannot be transmitted until time slot $n+1$ and that any unsuccessfully transmitted packets stay in the transmission buffer for later (re)transmission. For notational simplicity, we will omit the arguments of the packet goodput and instead write $f^n = f^n(BEP^n, z^n)$. Assuming independent packet losses, f^n is governed by a binomial distribution, i.e.

$$p^f(f^n \mid BEP^n, z^n) = \text{bin}(z^n, 1 - PLR^n), \tag{4.6}$$

and has expectation $E[f^n] = (1 - PLR^n)z^n$.

Based on the buffer recursion in (4.5), the distribution of the arrival process $p^l(l)$, and the distribution of the departure process $p^f(f \mid BEP, z)$, the sequence of buffer states $\{b^n : n = 0,1,\ldots\}$ can be modeled as a controlled Markov chain with transition probabilities

$$p^b\left(b' \mid [b,h,x], BEP, y, z\right) = \begin{cases} \sum_{f=0}^{z} p^l\left(b' - [b-f]\right) p^f(f \mid BEP, z), & \text{if } b' < B \\ \sum_{f=0}^{z} \sum_{l=B-[b-f]}^{\infty} p^l(l)\, p^f(f \mid BEP, z), & \text{if } b' = B. \end{cases} \tag{4.7}$$

Eq. (4.7) is similar to the buffer state transition probability function in [9], except that the model in [9] assumes that the packet goodput f is equal to the packet throughput z (i.e. there are no packet losses).

We also define a *buffer cost* to reward the system for minimizing queuing delays,

thereby protecting against overflows that may result from a sudden increase in transmission delay due to a bad fading channel or traffic burst. Formally, we define the buffer cost as the expected sum of the *holding cost* and *overflow cost* with respect to the arrival and goodput distributions: i.e.,

$$g([b,x],BEP,y,z) = \sum_{l=0}^{\infty}\sum_{f=0}^{z} p^l(l)\,p^f(f \mid BEP,z)\left\{\underbrace{[b-f]}_{\text{holding cost}} + \underbrace{\eta\max([b-f]+l-B,0)}_{\text{overflow cost}}\right\}. \quad (4.8)$$

In (4.8), the holding cost represents the number of packets that were in the buffer at the beginning of the time slot, but were not transmitted (and hence must be held in the buffer to be transmitted in a future time slot). If the buffer is stable (i.e. there are no buffer overflows), then the holding cost is proportional to the queuing delay by Little's theorem [10]. If the buffer is not stable (i.e. there is a finite probability of overflow such that Little's theorem does not apply), then the overflow cost imposes a penalty of η for each dropped packet. With the correct choice of η, the overflow cost ensures that it is suboptimal to drop packets while simultaneously transmitting with low power or shutting off the wireless card. In Appendix B, we discuss how to define the per-packet penalty η and how it relates to the delay.

4.4 Wireless Power Management Problem Formulation

In this section, we formulate the wireless power management problem as a constrained MDP. We define the joint state (hereafter *state*) of the system as a vector containing the buffer state b, channel state h, and power management state x, i.e. $s \triangleq (b,h,x) \in \mathcal{S}$. We also define the joint action (hereafter *action*) as a vector containing the BEP BEP,

power management action y, and packet throughput z, i.e. $a \triangleq (BEP, y, z) \in \mathcal{A}$. The sequence of states $\{s^n : n = 0,1,\ldots\}$ can be modeled as a controlled Markov chain with transition probabilities that can be determined from the conditionally independent buffer state, channel state, and power management state transitions as follows:

$$p(s' \mid s,a) = p^b(b' \mid [b,h,x], BEP, y, z)\, p^h(h' \mid h)\, p^x(x' \mid x,y). \tag{4.9}$$

The objective of the wireless power management problem is to minimize the *infinite horizon discounted power cost* subject to a constraint on the *infinite horizon discounted delay*. Let $\pi : S \mapsto \mathcal{A}$ denote a stationary *policy* mapping states to actions such that $a = \pi(s)$. Starting from state s and following policy π, the expected discounted power cost and expected discounted delay are defined as

$$\bar{P}^{\pi}(s) = E\left[\sum_{n=0}^{\infty} (\gamma)^n \rho(s^n, \pi(s^n)) \mid s^0 = s\right], \text{ and} \tag{4.10}$$

$$\bar{D}^{\pi}(s) = E\left[\sum_{n=0}^{\infty} (\gamma)^n g(s^n, \pi(s^n)) \mid s^0 = s\right], \tag{4.11}$$

where the expectation is over the sequence of states $\{s^n : n = 0,1,\ldots\}$, $\gamma \in [0,1)$ is the *discount factor*, and $(\gamma)^n$ denotes the discount factor to the nth power. Formally, the objective of the constrained wireless power management problem is

$$\text{Minimize } \bar{P}^{\pi}(s) \text{ subject to } \bar{D}^{\pi}(s) \leq \delta, \forall s \in \mathcal{S}, \tag{4.12}$$

where δ is the discounted delay constraint.

We minimize the infinite horizon *discounted* cost instead of, for example, the *time average* cost, because (i) typical traffic and channel dynamics only have "stationary" behavior over short time intervals such that the future dynamics cannot be easily

predicted without error; (ii) delay-sensitive traffic is more valuable the earlier it is transmitted; and, (iii) we may indeed want to optimize the time average cost, but because the application's lifetime is not known a priori (e.g. a video conference may end abruptly), we assume that the application will end with probability $1-\gamma$ in any time slot (i.e., in any state, with probability $1-\gamma$, the MDP transitions to a zero cost absorbing state [23]). For all of these reasons, costs should be minimized sooner rather than later, and therefore immediate costs should be weighted more heavily than future costs.

We can reformulate the constrained optimization in (4.12) as an unconstrained MDP by introducing a Lagrange multiplier associated with the delay constraint. We can then define a Lagrangian cost function:

$$c^{\mu}(s,a) = \rho(s,a) + \mu g(s,a), \tag{4.13}$$

where $\mu \geq 0$ is the Lagrange multiplier, $\rho(s,a)$ is the power cost defined in (4.3), and $g(s,a)$ is the buffer cost defined in (4.8). For a fixed λ, the unconstrained problem is to determine the optimal policy π^* that minimizes

$$L^{\pi,\mu}(s) = E\Big[\sum_{n=0}^{\infty}(\gamma)^n c^{\mu}(s^n, \pi(s^n)) \mid s^0 = s\Big],\ \forall s \in \mathcal{S} \tag{4.14}$$

Solving (4.14) is equivalent to solving the following dynamic programming equation:

$$V^{*,\mu}(s) = \min_{a\in\mathcal{A}}\Big\{c^{\mu}(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s' \mid s,a)V^{*,\mu}(s')\Big\},\ \forall s \in \mathcal{S} \tag{4.15}$$

where $V^{*,\mu} : \mathcal{S} \mapsto \mathbb{R}$ is the *optimal state-value function*.

We also define the *optimal action-value function* $Q^{*,\mu} : \mathcal{S}\times\mathcal{A} \mapsto \mathbb{R}$, which satisfies:

$$Q^{*,\mu}(s,a) = c^{\mu}(s,a) + \gamma\sum_{s'\in\mathcal{S}} p(s' \mid s,a)V^{*,\mu}(s'), \tag{4.16}$$

where $V^{*,\mu}(s') = \min_{a \in \mathcal{A}} Q^{*,\mu}(s', a)$. In words, $Q^{*,\mu}(s,a)$ is the infinite horizon discounted cost achieved by taking action a in state s and then following the optimal policy $\pi^{*,\mu}$ thereafter, where

$$\pi^{*,\mu}(s) = \arg\min_{a \in \mathcal{A}} Q^{*,\mu}(s,a), \ \forall s \in \mathcal{S}. \tag{4.17}$$

Assuming that the cost and transition probability functions are *known* and *stationary*, (4.15), (4.16), and (4.17) can be computed numerically using the well-known *value iteration* algorithm [11].

For notational simplicity, we drop the Lagrange multiplier from the notation in the remainder of the chapter unless it is necessary; hence, we will write $c(s,a)$, $V^*(s)$, $Q^*(s,a)$, $\pi^*(s)$ instead of $c^\mu(s,a)$, $V^{*,\mu}(s)$, $Q^{*,\mu}(s,a)$, and $\pi^{*,\mu}(s)$, respectively.

4.5 Learning the Optimal Policy

In practice, the cost and transition probability functions are (partially) *unknown* a priori. Consequently, V^* and π^* cannot be computed using value iteration; instead, they must be learned online, based on experience. To this end, we adopt a *model-free* RL approach, which can be used to learn Q^* and π^* online, without first estimating the unknown cost and transition probability functions.

We begin in Section 4.5.1 by introducing a well-known model-free RL algorithm called *Q-learning*, which directly estimates Q^* under the assumption that the system's dynamics are completely unknown a priori. In subsection 4.5.2, we develop a framework that allows us to integrate known information about the cost and transition probability functions into the learning process. Exploiting this partially known information

dramatically improves run-time performance compared to the Q-learning algorithm.

4.5.1 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) $\sigma^n = \left(s^n, a^n, c^n, s^{n+1}\right)$: i.e.,

$$Q^{n+1}(s^n, a^n) \leftarrow (1-\alpha^n)Q^n(s^n, a^n) + \alpha^n\left[c^n + \gamma \min_{a' \in \mathcal{A}} Q^n(s^{n+1}, a')\right], \tag{4.18}$$

where s^n and a^n are the state and performed action in time slot n, respectively; c^n is the corresponding cost with expectation $c(s^n, a^n)$; s^{n+1} is the resulting state in time slot $n+1$, which is distributed according to $p(s^{n+1} \mid s^n, a^n)$; a' is the *greedy action* in state s^{n+1}, which minimizes the current estimate of the action-value function; $\alpha^n \in [0,1]$ is a time-varying learning rate parameter; and, $Q^0(s,a)$ can be initialized arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

Q-learning essentially uses a sample average of the action-value function, i.e. Q^n, to approximate Q^*. It is well known that if (i) the instantaneous cost and transition probability functions are stationary, (ii) all of the state-action pairs are visited infinitely often, and (iii) α^n satisfies the *stochastic approximation conditions* $\sum_{n=0}^{\infty} \alpha^n = \infty$ and $\sum_{n=0}^{\infty} (\alpha^n)^2 < \infty$, then Q^n converges with probability 1 to Q^* as $n \to \infty$ (for example, $\alpha^n = 1/(n+1)$ or $\alpha^n = (1/n)^{0.7}$ satisfy the stochastic approximation conditions). Subsequently, the optimal policy can be calculated using (4.17).

Using Q-learning, it is not obvious what the best action is to take in each state during the learning process. On the one hand, Q^* can be learned by randomly *exploring* the

available actions in each state. Unfortunately, unguided randomized exploration cannot guarantee acceptable run-time performance because suboptimal actions will be taken frequently. On the other hand, taking greedy actions, which *exploit* the available information in Q^n, can guarantee a certain level of performance, but exploiting what is already known about the system prevents the discovery of better actions. Many techniques are available in the literature to judiciously trade off exploration and exploitation. In this chapter, we use the so-called ε*-greedy* action selection method [11], but other techniques such as Boltzmann exploration can also be deployed [11]. Importantly, the only reason why exploration is required is because Q-learning assumes that the unknown cost and transition probability functions depend on the action. In the remainder of this section, we will describe circumstances under which exploiting partial information about the system can obviate the need for action exploration.

Q-learning is an inefficient learning algorithm because it only updates the action-value function for a single state-action pair in each time slot and does not exploit known information about the system's dynamics. Consequently, it takes a long time for the algorithm to converge to a near-optimal policy, so run-time performance suffers [17] [18]. In the remainder of this section, we introduce two techniques that improve learning performance by exploiting known information about the system's dynamics and enabling more efficient use of the experience in each time slot.

4.5.2 Proposed Post-Decision State Learning

4.5.2.1 Partially Known Dynamics

The conventional Q-learning algorithm learns the value of state-action pairs under the assumption that the environment's dynamics (i.e. the cost and transition probability functions) are completely unknown a priori. In many systems, however, we have partial knowledge about the environment's dynamics. Table 4.1 highlights what is assumed known, what is assumed unknown, what is stochastic, and what is deterministic in this chapter. Note that this is just an illustrative example and the dynamics may be classified differently under different conditions. For example, if the parameter θ in (4.4) is known and in the interval $(0,1)$, then the power management state transition can be classified as stochastic and known as in [3]; if θ is unknown, then the power management state transition can be classified as stochastic and unknown; if the BEP function defined in (4.1) is unknown, then the goodput distribution and holding cost can be classified as stochastic and unknown; or, if the transmission power function in (4.2) is unknown, then the power cost can be classified as unknown. Importantly, the proposed framework can be applied regardless of the specific classification.

We will exploit the known information about the dynamics to develop more efficient learning algorithms than Q-learning, but first we need to formalize the concepts of known and unknown dynamics in our MDP-based framework.

Table 4.1. Classification of dynamics.

	Known	**Unknown**
Deterministic	• Power management state transition [(4.4), $\theta = 1$] • Power cost (4.3)	N/A
Stochastic	• Goodput distribution (4.6) (Section 4.3) • Holding cost (4.8)	• Traffic arrival distribution $p^l(l)$ • Channel state distribution $p^h(h' \mid h)$ • Overflow cost (4.8)

4.5.2.2 Post-Decision State Definition

We define a *post-decision state* (PDS) to describe the state of the system *after* the known dynamics take place, but *before* the unknown dynamics take place. We denote the PDS as $\tilde{s}$ and the set of possible PDSs as $\tilde{\mathcal{S}}$. Based on the discussion in the previous subsection, the PDS at time n is related to the state $s^n = (b^n, h^n, x^n)$, action $a^n = (BEP^n, y^n, z^n)$, and the state at time $n+1$, as follows:

- **PDS at time** n**:** $\tilde{s}^n = \left(\tilde{b}^n, \tilde{h}^n, \tilde{x}^n\right) = \left([b^n - f^n], h^n, x^{n+1}\right)$.
- **State at time** $n+1$**:** $s^{n+1} = \left(b^{n+1}, h^{n+1}, x^{n+1}\right) = \left([b^n - f^n] + l^n, h^{n+1}, x^{n+1}\right)$.

The buffer's PDS $\tilde{b}^n = b^n - f^n$ characterizes the buffer state *after* the packets are transmitted, but before new packets arrive; the channel's PDS is the same as the channel state at time n; and, the power management PDS is the same as the power management state at time $n+1$. In other words, the PDS incorporates all of the known information about the transition from state s^n to state s^{n+1} after taking action a^n. Meanwhile, the next state incorporates all of the unknown dynamics that were not included in the PDS (i.e. the number of packet arrivals l^n and next channel state h^{n+1}). Note that the buffer

state at time $n+1$ can be rewritten in terms of the buffer's PDS at time n as $b^{n+1} = \tilde{b}^n + l^n$.

These relationships are illustrated in Figure 4.2, where it is also shown that several state-action pairs can lead to the same PDS. A consequence of this is that acquiring information about a *single* PDS provides information about the *many* state-action pairs that can potentially precede it. Indeed, this is the foundation of PDS-based learning, which we describe in Section 4.5.2.4.

By introducing the PDS, we can factor the transition probability function into known and unknown components, where the known component accounts for the transition from the current state to the PDS, i.e. $s \to \tilde{s}$, and the unknown component accounts for the transition from the PDS to the next state, i.e. $\tilde{s} \to s'$. Formally,

$$p(s' \mid s,a) = \sum_{\tilde{s}} p_{\mathrm{u}}(s' \mid \tilde{s},a)\, p_{\mathrm{k}}(\tilde{s} \mid s,a), \tag{4.19}$$

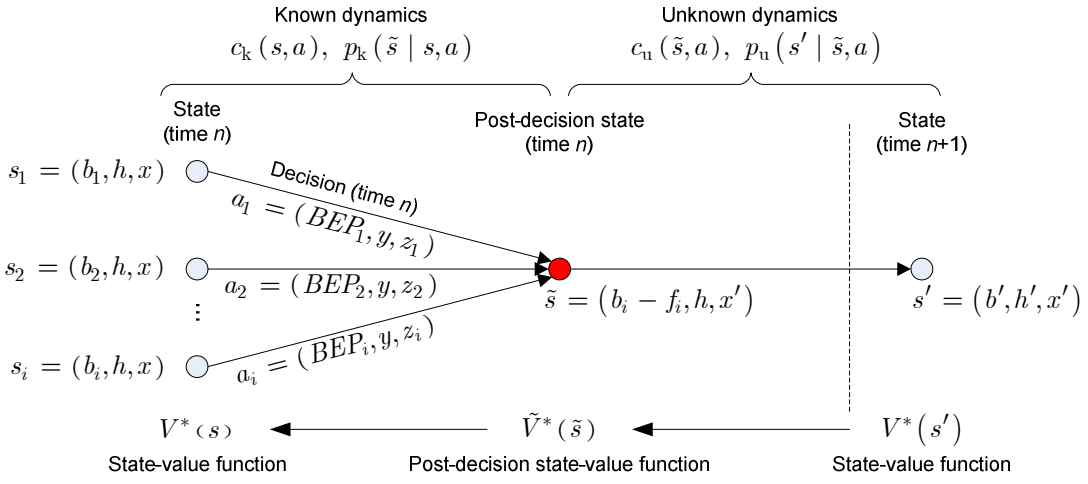


Figure 4.2. Relationship between the state-action pair at time n, PDS at time n, and state at time $n+1$. State action pairs $(s_1,a_1),\ldots,(s_i,a_i)$ potentially lead to the same PDS.

where the subscripts k and u denote the known and unknown components, respectively. We can factor the cost function similarly:

$$c(s,a) = c_{\mathrm{k}}(s,a) + \sum_{\tilde{s}} p_{\mathrm{k}}(\tilde{s} \mid s,a)\, c_{\mathrm{u}}(\tilde{s},a). \tag{4.20}$$

Notice that, in general, both the known and unknown components may depend on the action and the PDS. The proposed framework can be extended to this scenario, but action exploration will be necessary to learn the optimal policy. In the system under study, however, the unknown components only depend on the PDS (i.e. $p_{\mathrm{u}}(s' \mid \tilde{s},a) = p_{\mathrm{u}}(s' \mid \tilde{s})$ and $c_{\mathrm{u}}(\tilde{s},a) = c_{\mathrm{u}}(\tilde{s})$). The implication of this, discussed further in Section 4.5.2.4, is that action exploration is not needed to learn the optimal policy. We will focus on second scenario in the remainder of the chapter. Specifically, the known and unknown transition probability functions are defined as

$$p_{\mathrm{k}}(\tilde{s} \mid s,a) = p^{x}(\tilde{x} \mid x,y)\, p^{f}\left(b - \tilde{b} \mid BEP, z\right) I\left(\tilde{h} = h\right) \text{ and} \tag{4.21}$$

$$p_{\mathrm{u}}(s' \mid \tilde{s}) = p^{h}\left(h' \mid \tilde{h}\right) p^{l}\left(b' - \tilde{b}\right) I\left(x' = \tilde{x}\right), \tag{4.22}$$

where $I(\cdot)$ is the indicator function, which takes value 1 if its argument is true and 0 otherwise. Meanwhile, the known and unknown cost functions are defined as

$$c_{\mathrm{k}}(s,a) = \underbrace{\rho([h,x], BEP, y, z)}_{\text{power cost}} + \mu \sum_{f=0}^{z} p^{f}(f \mid BEP, z) \underbrace{[b-f]}_{\text{holding cost}}, \text{ and} \tag{4.23}$$

$$c_{\mathrm{u}}(\tilde{s}) = \mu\eta \sum_{l=0}^{\infty} p^{l}(l) \underbrace{\max\left(\tilde{b} + l - B, 0\right)}_{\text{overflow cost}}. \tag{4.24}$$

4.5.2.3 Post-Decision State-Based Dynamic Programming

Before we can describe the PDS learning algorithm, we need to define the *PDS value*

function, which is a value function defined over the PDSs. In the PDS learning algorithm, the PDS value function plays a similar role as the action-value function does in the conventional Q-learning algorithm. The optimal PDS value function, denoted by $\tilde{V}^*$, can be expressed as a function of the optimal state-value function and vice-versa:

$$\tilde{V}^*(\tilde{s}) = c_{\mathrm{u}}(\tilde{s}) + \gamma \sum_{s'} p_{\mathrm{u}}(s' \mid \tilde{s}) V^*(s'), \tag{4.25}$$

$$V^*(s) = \min_{a \in \mathcal{A}} \left\{ c_{\mathrm{k}}(s,a) + \sum_{\tilde{s}} p_{\mathrm{k}}(\tilde{s} \mid s,a) \tilde{V}^*(\tilde{s}) \right\}. \tag{4.26}$$

Given the optimal PDS value function, the optimal policy can be computed as

$$\pi^*_{\mathrm{PDS}}(s) = \min_{a \in \mathcal{A}} \left\{ c_{\mathrm{k}}(s,a) + \sum_{\tilde{s}} p_{\mathrm{k}}(\tilde{s} \mid s,a) \tilde{V}^*(\tilde{s}) \right\}. \tag{4.27}$$

The following proposition proves that π^*_{PDS}, defined in (4.27), and π^*, defined in (4.17), are equivalent.

Proposition 4.1: *π^*_{PDS} and π^* are equivalent.*

Proof: The proof is provided in Appendix C.

Proposition 4.1 is important because it enables us to use the PDS value function to learn the optimal policy.

4.5.2.4 The Post-Decision State Learning Algorithm

While Q-learning uses a sample average of the action-value function to approximate Q^*, PDS learning uses a sample average of the PDS value function to approximate $\tilde{V}^*$. However, because the value function V can be directly computed from the PDS value function $\tilde{V}$ using the known dynamics and (4.26), the PDS learning algorithm *only needs*

to learn the unknown dynamics in order to learn the optimal value function V^* and the optimal policy π^*. The PDS learning algorithm is summarized in Table 4.2. The following theorem shows that the PDS learning algorithm converges to the optimal PDS value function $\tilde{V}^{*,\mu}(\tilde{s})$ based on which the optimal scheduling policy can be determined.

Theorem 4.1 (Convergence of PDS learning algorithm): *The post-decision state learning algorithm converges to the optimal post-decision state value function* $\tilde{V}^{*,\mu}(\tilde{s})$ *when the sequence of learning rates* α^n *satisfies* $\sum_{n=0}^{\infty}\alpha^n = \infty$ *and* $\sum_{n=0}^{\infty}(\alpha^n)^2 < \infty$.

Proof: The proof is provided in Appendix D.

Table 4.2.Post-decision state-based learning algorithm.

1.	**Initialize:** At time $n = 0$, initialize the PDS value function $\tilde{V}^0$ as described in Appendix E.
2.	**Take the greedy action:** At time n, take the greedy action $$a^n = \arg\min_{a\in\mathcal{A}}\left\{c_{\mathrm{k}}(s^n,a) + \sum_{\tilde{s}} p_{\mathrm{k}}(\tilde{s} \mid s^n,a)\tilde{V}^n(\tilde{s})\right\}. \quad (4.28)$$
3.	**Observe experience:** Observe the PDS experience tuple $\tilde{\sigma}^n = \left(s^n, a^n, \tilde{s}^n, c_{\mathrm{u}}^n, s^{n+1}\right)$.
4.	**Evaluate the state-value function:** Compute the value of state s^{n+1}: $$V^n\left(s^{n+1}\right) = \min_{a\in\mathcal{A}}\left\{c_{\mathrm{k}}(s^{n+1},a) + \sum_{\tilde{s}} p_{\mathrm{k}}\left(\tilde{s} \mid s^{n+1},a\right)\tilde{V}^n(\tilde{s})\right\}. \quad (4.29)$$
5.	**Update the PDS value function:** At time n, update the PDS value function using the information from steps 3 and 4: $$\tilde{V}^{n+1}(\tilde{s}^n) \leftarrow (1-\alpha^n)\tilde{V}^n(\tilde{s}^n) + \alpha^n\left[c_{\mathrm{u}}^n + \gamma V^n\left(s^{n+1}\right)\right] \quad (4.30)$$
6.	**Lagrange multiplier update:** Update the Lagrange multiplier μ using (4.31).
7.	**Repeat:** Update the time index, i.e. $n \leftarrow n+1$. Go to step 2.

The optimal value of the Lagrange multiplier μ in (4.13), which depends on the

infinite horizon discounted average delay constraint, can be learned online using stochastic subgradients as in [12]: i.e.,

$$\mu^{n+1} = \Lambda[\mu^n + \beta^n(g^n - (1-\gamma)\delta)], \tag{4.31}$$

where Λ projects μ onto $[0, \mu_{\max}]$, β^n is a time-varying learning rate with the same properties as α^n, g^n is the buffer cost with expectation $g(s^n, a^n)$, and the $(1-\gamma)\delta$ term converts the discounted delay constraint to an average delay constraint. The following additional conditions must be satisfied by β^n and α^n to ensure convergence of (4.31) to μ^*:

$$\sum_{n=0}^{\infty}(\alpha^n + \beta^n) < \infty \text{ and } \lim_{n\to\infty}\frac{\beta^n}{\alpha^n} \to 0. \tag{4.32}$$

There are several ways in which the PDS learning algorithm uses information more efficiently than Q-learning. First, PDS learning exploits partial information about the system so that less information needs to be learned. This is evident from the use of the known information (i.e. $c_{\mathrm{k}}(s^n, a)$ and $p_{\mathrm{k}}(\tilde{s}^n \mid s^n, a)$) in (4.28) and (4.29). Second, updating a single PDS using (4.30) provides information about the state-value function at many states. This is evident from the expected PDS value function on the right-hand-side of (4.29). Third, because the unknown dynamics (i.e. $c_{\mathrm{u}}(\tilde{s}^n)$ and $p_{\mathrm{u}}\left(s^{n+1} \mid \tilde{s}^n\right)$) do not depend on the action, there is no need for randomized exploration to find the optimal action in each state. This means that the latest estimate of the value function can always be exploited and therefore non-greedy actions never have to be tested.

4.5.3 Proposed Virtual Experience Learning

The PDS learning algorithm described in the previous subsection only updates one PDS in each time slot. In this subsection, we discuss how to update multiple PDSs in each time slot in order to accelerate the learning rate and improve run-time performance. The key idea is to realize that the traffic arrival and channel state distributions are independent of the buffer and power management states. Consequently, the statistical information obtained from the PDS at time n about the a priori unknown traffic arrival and channel state distributions can be extrapolated to other PDSs with different post-decision buffer states and different post-decision power management states. Specifically, given the PDS experience tuple in time slot n, i.e. $\tilde{\sigma}^n = \left(s^n, a^n, \tilde{s}^n, c_{\mathrm{u}}^n, s^{n+1}\right)$, the PDS value function update defined in (4.30) can be applied to every *virtual experience tuple* in the set

$$\sum(\tilde{\sigma}^n) = \left\{\left(s^n, a^n, \left[\tilde{b}, \tilde{h}^n, \tilde{x}\right], c_{\mathrm{u}}\left(\tilde{b}; l^n\right), \left[\tilde{b} + l^n, h^{n+1}, \tilde{x}\right]\right) \mid \forall\left(\tilde{b}, \tilde{x}\right) \in \mathcal{B} \times \mathcal{X}\right\}, \qquad (4.33)$$

where $c_{\mathrm{u}}\left(\tilde{b}; l\right) = \eta \max\left(\tilde{b} + l - B, 0\right)$. We refer to elements of $\sum(\tilde{\sigma})$ as *virtual* experience tuples because they do not actually occur in time slot n. As shown in (4.33), the virtual experience tuples in $\sum(\tilde{\sigma}^n)$ are the same as the actual experience tuple $\tilde{\sigma}^n$ except that they have a different post-decision buffer state and/or a different post-decision power management state, and a different cost.

By performing the PDS update in (4.30) on every virtual experience tuple in $\sum(\tilde{\sigma}^n)$, the proposed virtual experience learning algorithm improves adaptation speed at the expense of a $|\mathcal{B} \times \mathcal{X}|$-fold increase in computational complexity ($|\mathcal{B} \times \mathcal{X}| = |\sum(\tilde{\sigma})|$). We show in the results section that we can trade off performance and learning complexity by

only performing the complex virtual experience updates every T time slots.

4.5.4 Conceptual Comparison of Learning Algorithms

Figure 4.3 illustrates the key differences between Q-learning (described in Section 4.5.1), PDS learning (proposed in subsection 4.5.2), and virtual experience learning (proposed in subsection 4.5.3). To facilitate graphical illustration, Figure 4.3 uses a simplified version of the system model defined in Section 4.3 in which the state is fully characterized by the buffer state b, the only action is the throughput z, and there are no packet losses. Figure 4.3(a) illustrates how one Q-learning update on state-action pair (b,z) only provides information about the buffer-throughput pair (b,z). Figure 4.3(b) illustrates how one PDS learning update on PDS $b-z$ provides information about every state-action pair that can potentially lead to the PDS, which in this example corresponds to all buffer-throughput pairs (b',z') such that $b'-z'=b-z$. Finally, Figure 4.3(c) illustrates how performing a PDS learning update on every virtual experience tuple associated with PDS $b-z$ provides information about every buffer state.

Table 4.3 illustrates both the greedy action selection complexity and learning update complexity for Q-learning, PDS learning, and virtual experience learning. It is clear from Table 4.3 that as more information is integrated into the learning algorithm, it becomes more complex to implement.

Table 4.3. Learning algorithm complexity (in each time slot). In the system under study $\left|\tilde{\mathcal{S}}\right| = |\mathcal{B}|$ and $|\sum| = |\mathcal{B} \times \mathcal{X}|$.

	Action Selection Complexity	Learning Update Complexity
Q-learning	$O(\|\mathcal{A}\|)$	$O(\|\mathcal{A}\|)$
PDS learning	$O\left(\left\|\tilde{\mathcal{S}}\right\|\|\mathcal{A}\|\right)$	$O\left(\left\|\tilde{\mathcal{S}}\right\|\|\mathcal{A}\|\right)$
Virtual experience learning	$O\left(\left\|\tilde{\mathcal{S}}\right\|\|\mathcal{A}\|\right)$	$O\left(\|\sum\|\left\|\tilde{\mathcal{S}}\right\|\|\mathcal{A}\|\right)$

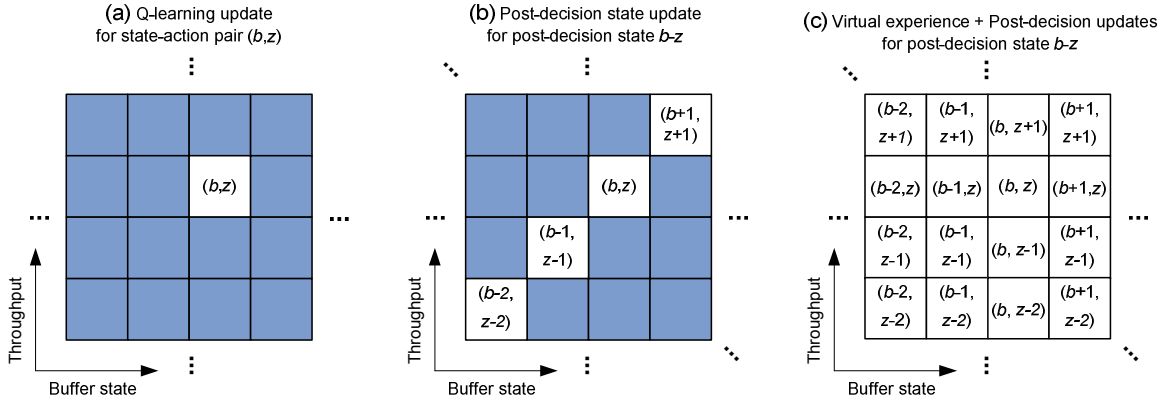


Figure 4.3. State-action pairs that are impacted in one time slot when using different learning updates (highlighted in white). (a) Q-learning update for state-action pair (b, z); (b) PDS update for PDS $b - z$; (c) Virtual experience updates for PDS $b - z$.

4.6 Simulation Results

4.6.1 Simulation Setup

Table 4.4 summarizes the parameters used in our MATLAB-based simulator. We assume that the PHY layer is designed to handle QAM rectangular constellations and that a Gray code is used to map the information bits into QAM symbols. The corresponding bit-error probability and transmission power functions can be found in [7]. We evaluate the performance of the proposed algorithm for several values of P_{on}: at 320 mW, P_{on} is

Table 4.4. Simulation parameters.

Parameter	Value	Parameter	Value
Arrival rate λ	200 packets/second	Packet loss rates PLR	{1, 2, 4, 8, 16} %
Bits per symbol β	{1, 2, … , 10}	Packet size ℓ	5000 bits
Buffer size B	25 packets	Power management actions $y \in \mathcal{Y}$	$\{\mathrm{s\_on}, \mathrm{s\_off}\}$
Channel states $h \in \mathcal{H}$	{-18.82, -13.79, -11.23, -9.37, -7.80, -6.30, -4.68, -2.08} dB	Power management states $x \in \mathcal{X}$	$\{\mathrm{on}, \mathrm{off}\}$
Discount factor γ	0.98	Symbol rate $1/T_s$	500×10^3 symbols/s
Holding cost constraint	4 packets	Time slot duration Δt	10 ms
Noise power spectral density N_0	2×10^{-11} watts/Hz	Transition power P_{tr}	Set equal to P_{on}
"Off" power P_{off}	0 watts	Transmission actions $z \in \mathcal{Z}$	{0, 1, 2, … , 10} packets/time slot
"On" power P_{on}	80 mW, 160mW, or 320 mW		

much larger than P_{tx} as in typical 802.11a/b/g wireless cards [15]; at 80 mW, P_{on} is closer to P_{tx} as in cellular networks. The channel transition distribution $p^h(h' \mid h)$ and packet arrival distribution $p^l(l)$ are assumed to be unknown a priori in our simulations. Lastly, we choose a discount factor of $\gamma = 0.98$ in the optimization objective (γ closer to 1 yields better performance after convergence, but requires more time to converge).

The signal-to-noise ratio (SNR) used for the tests is $SNR^n = \frac{P_{\mathrm{tx}}^n}{N_0 W}$, where P_{tx}^n is the transmission power in time slot n, N_0 is the noise power spectral density, and W is the bandwidth. We assume that the bandwidth is equal to the symbol rate, i.e. $W = \frac{1}{T_s}$, where T_s is the symbol duration.

4.6.2 Learning Algorithm Comparison

Simulation results using the parameters described in Section 4.6.1 and Table 4.4 are presented in Figure 4.4 for numerous simulations with duration 75,000 time slots (750 s) and $P_{\text{on}} = 320$ mW. Figure 4.4(a) illustrates the cumulative average cost versus time, where the cost is defined as in (4.13); Figure 4.4(b) illustrates the cumulative average power versus time, where the power is defined as in (4.3); Figure 4.4(c) and Figure 4.4(d) illustrate the cumulative average holding cost and cumulative average packet overflows versus time, respectively, as defined in (4.8); Figure 4.4(e) illustrates the cumulative average number of time slots spent in the off state (with both $x = \text{off}$ and $y = \text{s\_off}$), denoted by θ_{off}, versus time; and Figure 4.4(f) illustrates the windowed average of the Lagrange multiplier versus time (with a window size of 1000 time slots).

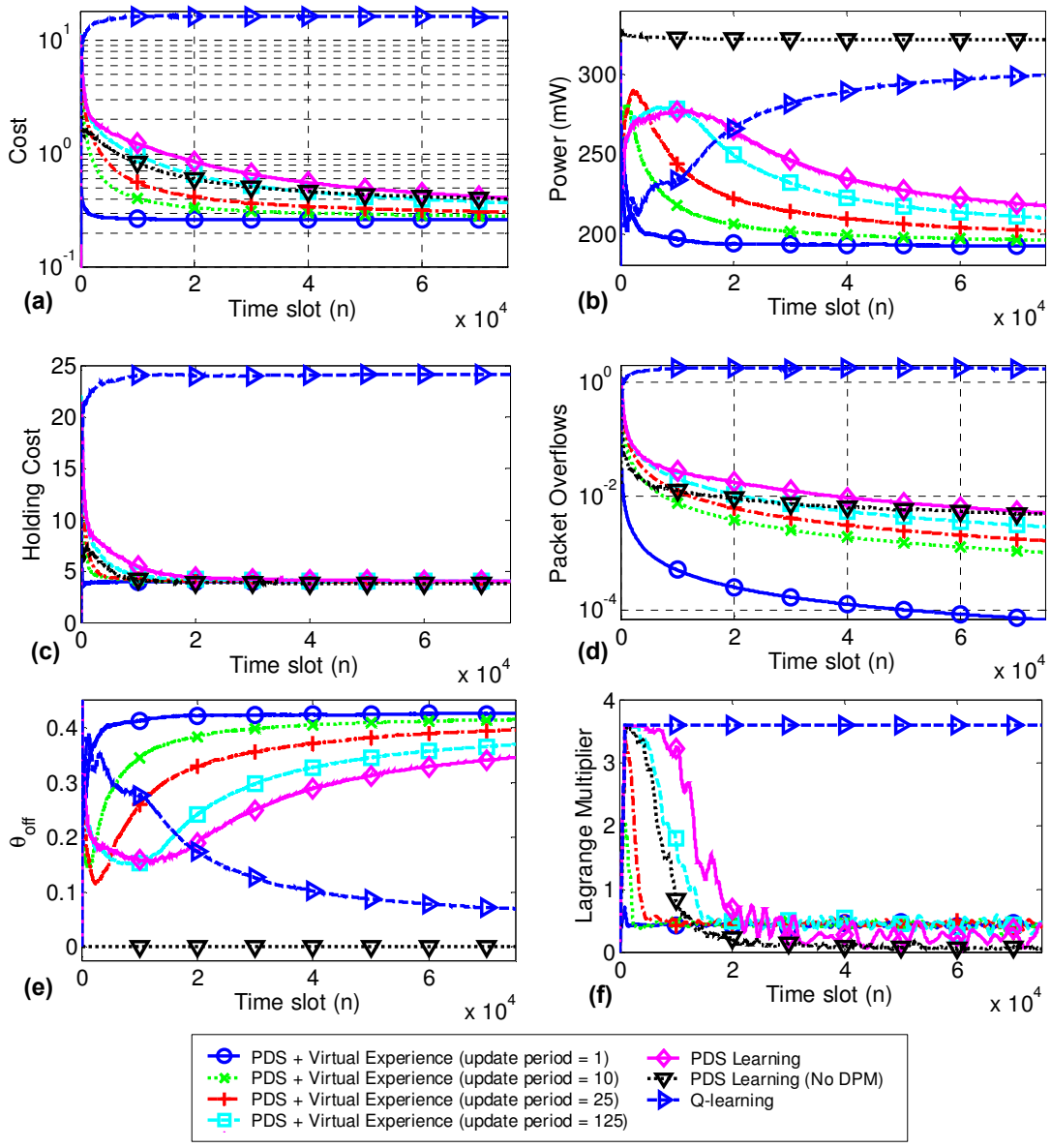


Figure 4.4. Quantitative learning algorithm comparison with $P_{\text{on}} = 320$ mW. (a) Cumulative average cost vs. time slot. The y-axis is in log-scale due to the high dynamic range. (b) Cumulative average power vs. time slot. (c) Cumulative average holding cost vs. time slot. (d) Cumulative average packet overflows vs. time slot. The y-axis is in log-scale due to the high dynamic range. (e) Cumulative average θ_{off} vs. time slot. (f) Windowed average of Lagrange multiplier μ vs. time slot (window size = 1000 time slots).

Each plot in Figure 4.4 compares the performance of the proposed PDS learning algorithm (with virtual experience updates every T = 1, 10, 25, 125 time slots) to the conventional Q-learning algorithm [11] and to an existing PDS learning-based solution [12]. The curves labeled "PDS Learning (No DPM)" are obtained by adapting the algorithm in [12] to use a discounted MDP formulation, rather than a time-average MDP formulation, and to include an adaptive BEP. This algorithm does not consider system-level power management (i.e. no DPM) and does not exploit virtual experience. Meanwhile, the curves labeled "PDS Learning" are obtained by further modifying the algorithm in [12] to include system-level power management.

In Figure 4.4, the PDS learning algorithms are initialized as described in Appendix E assuming that the arrival distribution is deterministic with 5 packet arrivals per time slot and that the channel transition matrix is the identity matrix.

As expected, using a virtual experience update period of 1 leads to the best performance [see "PDS + Virtual Experience (update period = 1)" in Figure 4.4]; in fact, this algorithm achieves approximately optimal performance after 3,000 time slots, at which point the cumulative average cost and Lagrange multiplier settle to nearly steady-state values [Figure 4.4(a) and Figure 4.4(f), respectively]. Thus, if sufficient computational resources are available, the jointly optimal power-control, AMC, and DPM policies can be quickly learned online by smartly exploiting the structure of the problem using PDS learning combined with virtual experience. In existing wireless communications systems (especially in mobile devices), however, there are not enough computational resources to use virtual experience in every time slot. Hence, we also

illustrate the performance of the proposed algorithm for different update periods [see "PDS + Virtual Experience (update period = 10, 25, 125)" in Figure 4.4] and without doing any virtual experience updates (see "PDS Learning" in Figure 4.4). As the number of updates in each time slot is decreased, the learning performance also decreases.

From the plots with different update periods, we can identify how the system adapts over time. Initially, the holding cost and packet overflows rapidly increase because the initial policy is not tuned to the specific traffic and channel conditions. This increasing delay drives the Lagrange multiplier to its predefined maximum value (resulting in a cost function that weighs delay more heavily and power less heavily), which leads to increased power consumption, which in turn drives the holding cost back down toward the holding cost constraint. As the cumulative average holding cost decreases towards the holding cost constraint, the Lagrange multiplier also begins to decrease, which eventually leads to a decrease in power consumption (because the cost function begins to weigh delay less heavily and power more heavily). Clearly, this entire process proceeds quicker when there are more frequent virtual experience updates, thereby resulting in better overall performance. This process also explains why near optimal delay performance is achieved sooner than near optimal power consumption (because the minimum power consumption can only be learned *after* the holding cost constraint is satisfied).

We now compare our proposed solution to the "PDS Learning (No DPM)" curves in Figure 4.4, which were obtained using the PDS learning algorithm introduced in [12] (modified slightly as described at the beginning of this subsection). It is clear that this

algorithm performs approximately the same as our proposed "PDS + Virtual Experience (update period = 125)" algorithm in terms of delay (i.e. holding cost and overflows) but not in terms of power. The reason for the large disparity in power consumption is that the PDS learning algorithm introduced in [12] does not take advantage of system-level DPM, which is where the large majority of power savings comes from when P_{on} is significantly larger than P_{tx}. This is corroborated by the fact that the θ_{off} curves in Figure 4.4(e) and the power consumption curves in Figure 4.4(b) are very similarly shaped. Thus, solutions that ignore system-level power management achieve severely suboptimal power in practice. (Note that PHY-centric power management solutions are still important for achieving the desired delay constraint!) Interestingly, despite using the same number of learning updates in each time slot, the "PDS Learning (No DPM)" algorithm performs better than "PDS Learning" algorithm in terms of delay (i.e. holding cost and overflow). This can be explained by the fact that DPM is ignored, so the learning problem is considerably simpler (i.e. there are less states and actions to learn about).

Lastly, we observe that the conventional Q-learning algorithm (see "Q-learning" in Figure 4.4) performs very poorly. Although Q-learning theoretically converges to the optimal policy, and therefore, should eventually approach the performance of the other algorithms, its cost in Figure 4.4(a) goes flat after its buffer overflows in Figure 4.4(d). It turns out that Q-learning has trouble finding the best actions to reduce the buffer occupancy because it requires action exploration and it only updates one state-action pair in each time slot. Nevertheless, it is clear from the increasing power cost in Figure 4.4(b) and the decreasing value of θ_{off} in Figure 4.4(e) that Q-learning is slowly learning the

optimal policy: indeed, all of the learning algorithms must learn to keep the wireless card on before they can transmit enough packets to drain the buffer and satisfy the buffer constraint.

4.6.3 Comparison to Optimal Policy With Imperfect Statistics

In Figure 4.5, we compare the performance of the best learning algorithm in Figure 4.4 ("PDS + Virtual Experience (update period = 1)" to the expected performance of a per-step suboptimal algorithm. The per-step suboptimal algorithm uses value iteration to compute the optimal policy with respect to imperfect statistics, which are estimated from previous channel transition and packet arrival realizations. The reason the PDS learning algorithm does not track the per-step suboptimal algorithm more closely during the early stages of learning is because it not only needs to learn the imperfect statistics (like the per-step suboptimal algorithm), but it also needs to learn the value function (unlike the per-step suboptimal algorithm, which can compute the value function directly from the statistics without regard for complexity).

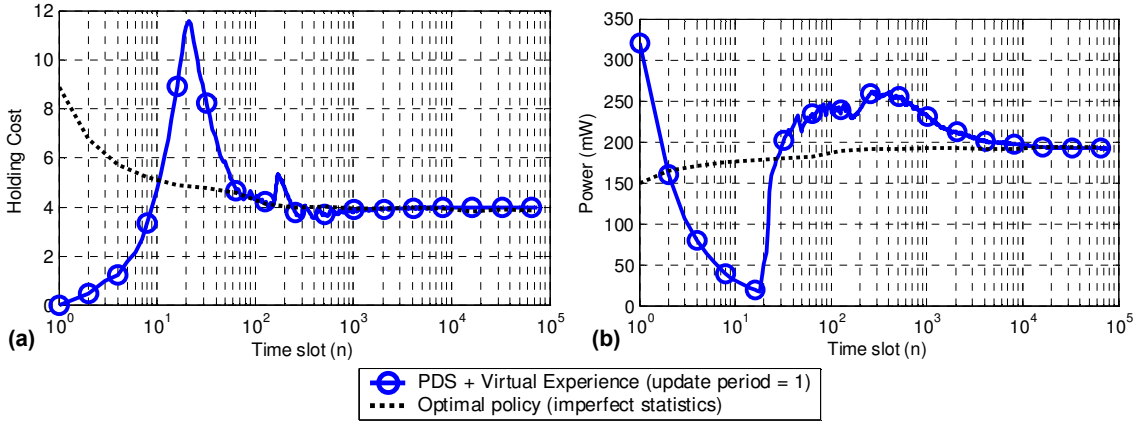


Figure 4.5. Comparison of the optimal policy with imperfect statistics to the PDS learning algorithm with virtual experience and update period = 1. The x-axis is in log-scale to best highlight the learning process over time. (a) Holding cost vs. time slot. (b) Power vs. time slot.

4.6.4 Performance Under Non-Stationary Dynamics

In this subsection, we compare our proposed solution to the network energy saving solution in [6], which combines transmission scheduling with DPM, but uses a fixed BEP. Like our proposed solution, the DPM and packet scheduling solution in [6] uses a buffer to enable power-delay trade-offs. Unlike our proposed solution, however, the solution in [6] ignores the channel and traffic dynamics and is based on a simple *threshold-k policy*: That is, if the buffer backlog exceeds k packets, then the wireless card is turned on and all backlogged packets are transmitted as quickly as possible; then, after transmitting the packets, the wireless card is turned off again.

Figure 4.6 illustrates the average power-delay performance achieved by the proposed PDS learning algorithm (with virtual experience updates every $T = 50$ time slots) and the threshold-k policy for three values of P_{on} after 75,000 time slots. All curves are generated assuming a fixed PLR of 1% and are obtained under non-stationary arrival dynamics[4] and non-stationary channel dynamics[5]. The proposed algorithm performs between 10 and 30 mW better than the threshold-k policy across the range of holding cost constraints despite the fact that RL algorithms can only converge to optimal solutions in stationary environments. Interestingly, the results in Figure 4.6 show that, for low values of P_{on}, the average power can actually *increase* as the delay increases under the threshold-k policy. This is because, after the buffer backlog exceeds the predefined

[4] Non-stationary arrival dynamics are generated by a 5-state Markov chain with states (0, 100, 200, 300, 400) packets/s and corresponding stationary distribution (0.0188, 0.3755, 0.0973, 0.4842, 0.0242). The state indicates the expected arrival rate for a Poisson arrival distribution. Importantly, the MDP does not know the traffic state so it simply assumes that the arrival distribution is i.i.d.

[5] Non-stationary channel dynamics are generated by randomly perturbing the channel state transition probability $p(h' \mid h)$ from time slot to time slot.

threshold k, the wireless card is turned on and transmits as many packets as possible regardless of the channel quality. Hence, a significant amount of transmit power is wasted in bad channel states. However, as P_{on} increases, the results in Figure 4.6 show that the performance begins to improve and more closely approximate the optimal power-delay trade-off. Indeed, a more aggressive scheduling policy becomes necessary as P_{on} increases so that the wireless card may be in the "off" state more frequently. The threshold-k policy takes this observation to the extreme by transmitting the maximum number of packets possible in order to maximize the time spent in the "off" state. Unfortunately, because it ignores the transmission power, the threshold-k policy cannot perform optimally, especially for smaller values of P_{on}.

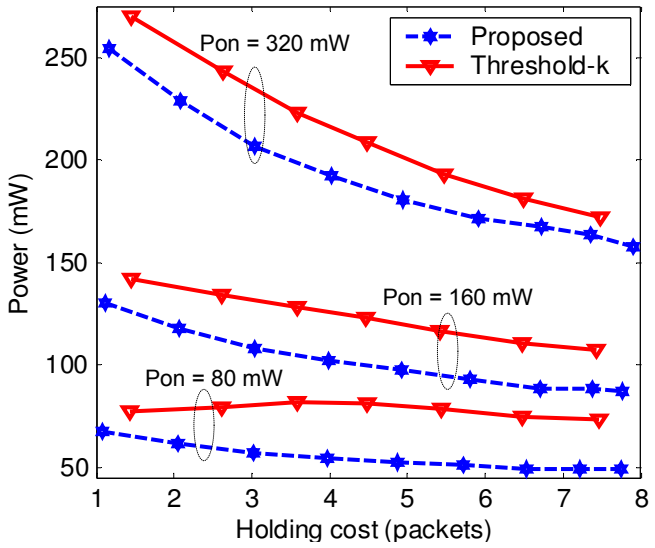


Figure 4.6. Power-delay performance of proposed solution (with virtual experience updates every T = 50 time slots) compared to the power-delay performance of the threshold-k policy. Traffic and channel dynamics are non-stationary.

4.7 Conclusion

In this chapter, we considered the problem of energy-efficient point-to-point transmission of delay-sensitive multimedia data over a fading channel. We proposed a unified reinforcement learning solution for finding the jointly optimal power-control, AMC, and DPM policies when the traffic arrival and channel statistics are unknown. We exploited the structure of the problem by introducing a post-decision state, eliminating action-exploration, and enabling virtual experience to dramatically improve performance compared to conventional RL algorithms. Our experimental results demonstrate that the proposed solution significantly outperforms existing solutions, even under nonstationary traffic and channel conditions. The results also strongly support the use of system-level power management solutions in conjunction with PHY-centric solutions to achieve the minimum possible power consumption, under delay constraints, in wireless communication systems.

Importantly, the proposed framework can be applied to any network or system resource management problem involving controlled buffers (e.g. multi-user uplink and downlink transmission; delay-sensitive data transmission over multi-hop wired/wireless networks; DPM for other system components such as processors and hard-drives; and job scheduling and dynamic-voltage-scaling for energy-efficient processing).

Appendix B: Overflow Cost

In this appendix, we discuss how to set the penalty η in (4.8). Recall that, with the correct choice of η, the overflow cost ensures that it is suboptimal to drop packets while

simultaneously transmitting with low power or shutting off the wireless card. Intuitively, this means that η should be at least as large as the maximum infinite horizon discounted holding cost that can be incurred by a single packet, i.e. the total discounted holding cost incurred by holding a packet in the buffer forever. Therefore, since a packet arriving in time slot n_0 does not incur cost until time slot $n_0 + 1$, we have

$$\eta \geq \sum_{n=n_0+1}^{\infty} (\gamma)^{n-n_0} 1 = \frac{\gamma}{1-\gamma}. \tag{4.34}$$

We choose $\eta = \frac{\gamma}{1-\gamma}$ because of its relationship to the delay. Specifically, $\frac{\gamma}{1-\gamma}$ can be interpreted as the maximum possible infinite horizon discounted delay that can be incurred by a single packet.

Appendix C: Proof of Proposition 4.1

To show that π^*_{PDS} and π^* are equivalent we substitute (4.25) into (4.27) as follows:

$$\begin{aligned}
\pi^*_{\text{PDS}}(s) &= \min_{a\in\mathcal{A}} \left\{ c_{\text{k}}(s,a) + \sum_{\tilde{s}} p_{\text{k}}(\tilde{s} \mid s,a)\tilde{V}^*(\tilde{s}) \right\} \\
&= \min_{a\in\mathcal{A}} \left\{ c_{\text{k}}(s,a) + \sum_{\tilde{s}} p_{\text{k}}(\tilde{s} \mid s,a) \left[c_{\text{u}}(\tilde{s}) + \gamma \sum_{s'} p_{\text{u}}(s' \mid \tilde{s}) V^*(s') \right] \right\} \\
&= \min_{a\in\mathcal{A}} \left\{ c_{\text{k}}(s,a) + \sum_{\tilde{s}} p_{\text{k}}(\tilde{s} \mid s,a) c_{\text{u}}(\tilde{s}) + \gamma \sum_{\tilde{s}} \sum_{s'} p_{\text{k}}(\tilde{s} \mid s,a) p_{\text{u}}(s' \mid \tilde{s}) V^*(s') \right\} \\
&= \min_{a\in\mathcal{A}} \left\{ c(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V^*(s') \right\} \\
&= \min_{a\in\mathcal{A}} \{ Q^*(s,a) \} \\
&= \pi^*(s).
\end{aligned}$$

where the fourth equality follows from (4.19) and (4.20). □

Appendix D: Proof of Theorem 4.1

The post-decision state learning algorithm defined in Table 4.2 can be written using the recursion

$$\tilde{V}^{n+1}(\tilde{s}) \leftarrow \tilde{V}^{n}(\tilde{s}) + \alpha^{n}\left[c_{\mathrm{u}}(\tilde{s}) + \gamma \min_{a}\left\{\begin{array}{l} c_{\mathrm{k}}(\Psi^{n+1}(\tilde{s}), a) + \\ \sum_{\tilde{s}'} p_{\mathrm{k}}\left(\tilde{s}' \mid \Psi^{n+1}(\tilde{s}), a\right)\tilde{V}^{n}\left(\tilde{s}'\right)\end{array}\right\} - \tilde{V}^{n}(\tilde{s})\right]$$

where $\tilde{s}, \tilde{s}' \in \mathcal{S}$, $a \in \mathcal{A}$, and $\Psi^{n+1}(\tilde{s}) \in \mathcal{S}$ is an independently simulated random variable drawn from $p_{\mathrm{u}}(\cdot \mid \tilde{s})$.

Let $h : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ be a map such that $h(\tilde{V}) = \left[h_{\tilde{s}}\left(\tilde{V}\right)\right]_{\tilde{s}}$ with

$$h_{\tilde{s}}\left(\tilde{V}\right) = c_{\mathrm{u}}(\tilde{s}) + \gamma \sum_{s'} p_{\mathrm{u}}\left(s' \mid \tilde{s}\right) \min_{a}\left\{c_{\mathrm{k}}(s', a) + \sum_{\tilde{s}'} p_{\mathrm{k}}\left(\tilde{s}' \mid s', a\right)\tilde{V}^{n}\left(\tilde{s}'\right)\right\} - \tilde{V}^{n}(\tilde{s}),$$

where $\tilde{s}, \tilde{s}' \in \mathcal{S}$, $a \in \mathcal{A}$. Define $F(\tilde{V}) = \left[F_{\tilde{s}}\left(\tilde{V}\right)\right]_{\tilde{s}}$ with

$$F_{\tilde{s}}\left(\tilde{V}\right) = c_{\mathrm{u}}(\tilde{s}) + \gamma \sum_{s'} p_{\mathrm{u}}\left(s' \mid \tilde{s}\right) \min_{a}\left\{c_{\mathrm{k}}(s', a) + \sum_{\tilde{s}'} p_{\mathrm{k}}\left(\tilde{s}' \mid s', a\right)\tilde{V}^{n}\left(\tilde{s}'\right)\right\}.$$

Then, $h\left(\tilde{V}\right) = F\left(\tilde{V}\right) - \tilde{V}$. As in [24], it can be shown that the convergence of the online learning algorithm is equivalent to the convergence of the associated O.D.E.

$$\dot{\tilde{V}} = F\left(\tilde{V}\right) - \tilde{V} := h\left(\tilde{V}\right).$$

Since the map $F : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ is a maximum norm γ-contraction [25], the asymptotic stability of the unique equilibrium point of the above O.D.E. is guaranteed [24]. This unique equilibrium point corresponds to the optimal post-decision state value function $\tilde{V}^{*,\mu}$. □

Appendix E: Initializing The PDS Value Function

Given the known and unknown cost functions (i.e. $c_{\mathrm{k}}(s,a)$ and $c_{\mathrm{u}}(\tilde{s})$, respectively) and the known and unknown transition probability functions (i.e. $p_{\mathrm{k}}(\tilde{s} \mid s,a)$ and $p_{\mathrm{u}}(s' \mid \tilde{s})$), the optimal PDS value function can be computed using PDS value iteration as follows:

$$\tilde{V}_k(\tilde{s}) = c_{\mathrm{u}}(\tilde{s}) + \gamma \sum_{s'} p_{\mathrm{u}}(s' \mid \tilde{s}) V_k(s'), \text{ and} \tag{4.35}$$

$$V_{k+1}(s) = \min_{a \in \mathcal{A}} \left\{ c_{\mathrm{k}}(s,a) + \sum_{\tilde{s}} p_{\mathrm{k}}(\tilde{s} \mid s,a) \tilde{V}_k(\tilde{s}) \right\}, \tag{4.36}$$

where k denotes the iteration index; $V_0(s)$ can be initialized arbitrarily; and $\tilde{V}_k$ converges to $\tilde{V}^*$ as $k \to \infty$. If we make assumptions about $c_{\mathrm{u}}(\tilde{s})$ and $p_{\mathrm{u}}(s' \mid \tilde{s})$, then we can use PDS value iteration to initialize the PDS value function in step 1 of the PDS learning algorithm described in Table 4.2.

Under arbitrary assumptions about the initial PDS value function, selecting the greedy action using (4.28) in each time slot can cause the PDS learning algorithm to get stuck in an undesirable state (because, under certain policies, the transition probability function is not irreducible). In particular, in the system under study, the PDS learning algorithm can get stuck in the "off" state (i.e. $x = \mathrm{off}$) if the greedy action is always "switch off" (i.e. $y = \mathrm{s\_off}$). To prevent this, we need to initialize the PDS value function offline by performing the PDS value iteration algorithm with the unknown cost function $c_{\mathrm{u}}(\tilde{s})$ and unknown transition probability function $p_{\mathrm{u}}(s' \mid \tilde{s})$ replaced with reasonable estimates $\widehat{c_{\mathrm{u}}}(\tilde{s})$ and $\widehat{p_{\mathrm{u}}}(s' \mid \tilde{s})$, respectively. In the system under study, estimates can be derived from past measurements of the traffic arrival and channel state distributions. Most

importantly, the estimate of the traffic arrival distribution should be designed to force overflows to occur in the "off" state; in this way, the greedy action in the "off" state will not always be "switch off."

Figure 4.7 illustrates the sensitivity of the PDS learning algorithm to the initialized arrival rate which impacts $p_{\mathrm{u}}\left(s' \mid \tilde{s}\right)$ through the buffer transition and impacts $c_{\mathrm{u}}(\tilde{s})$ through the overflow cost. In Figure 4.7, the operating points under stationary dynamics use the same arrival and channel dynamics as in Section 4.6.2 and 4.6.3, and the operating points under non-stationary dynamics use the same arrival and channel dynamics as in Section 4.6.4. These empirical results demonstrate that initializing the traffic arrival distribution as deterministic leads to good performance that is relatively insensitive to the initialized arrival rate in both stationary and non-stationary environments. These results also demonstrate that poorly selecting the arrival distribution (e.g. assuming arrivals are uniformly distributed over $\{0,1,\ldots,B\}$) can adversely impact the power-delay performance. In all of the PDS learning results in this chapter, we assume that the channel transition matrix is initialized as the identity matrix.

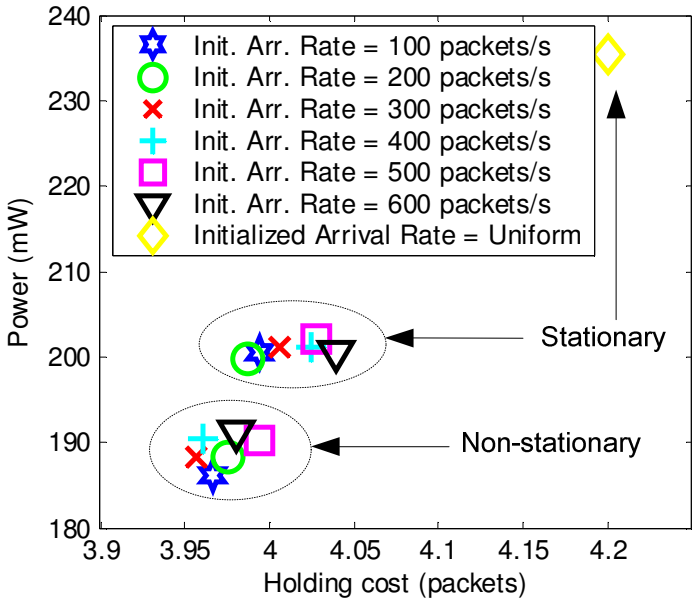


Figure 4.7. Sensitivity to the arrival distribution used to initialize the PDS value function. The arrival distribution is assumed to be deterministic or uniformly distributed over $\{0,1,\ldots,B\}$. Virtual experience updates are used every $T = 25$ time slots.

References

[1] D. Rajan, A. Sabharwal, and B. Aazhang, "Delay-bounded packet scheduling of bursty traffic over wireless channels," *IEEE Trans. on Information Theory*, vol. 50, no. 1, Jan. 2004.

[2] R. Berry and R. G. Gallager, "Communications over fading channels with delay constraints," *IEEE Trans. Inf. Theory*, vol 48, no. 5, pp. 1135-1149, May 2002.

[3] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. on computer-aided design of integrated circuits*, vol. 18, no. 6, June 1999.

[4] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. on Computers*, vol. 51, no. 11, Nov. 2002.

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

[6] K. Nahrstedt, W. Yuan, S. Shah, Y. Xue, and K. Chen, "QoS support in multimedia wireless environments," in Multimedia Over IP and Wireless Networks, ed. M. van der Schaar and P. Chou, Academic Press, 2007.

[7] J. G. Proakis, Digital Communications. New York: McGraw-Hill, 2001.

[8] C. Schurgers, Energy-aware wireless communications. Ph.D. dissertation, University of California at Los Angeles, 2002.

[9] D. V. Djonin and V. Krishnamurthy, "MIMO transmission control in fading channels – a constrained Markov decision process formulation with monotone randomized policies," *IEEE Trans. on Signal Processing*, vol. 55, no. 10, Oct. 2007.

[10] D. Bertsekas, and R. Gallager, "Data networks," Prentice Hall, Inc., Upper Saddle River, NJ, 1987.

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

[12] N. Salodkar, A. Bhorkar, A. Karandikar, V. S. Borkar, "An on-line learning algorithm for energy efficient delay constrained scheduling over a fading channel," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 732-742, Apr. 2008.

[13] M. H. Ngo and V. Krishnamurthy, "Monotonocity of constrained optimal transmission policies in correlated fading channels with ARQ," *IEEE Trans. on Signal Processing*, vol. 58, No. 1, pp. 438-451, Jan. 2010.

[14] N. Mastronarde and M. van der Schaar, "Online reinforcement learning for dynamic multimedia systems," *IEEE Trans. on Image Processing*, vol. 19, no. 2, pp. 290-305, Feb. 2010.

[15] Cisco, Cisco Aironet 802.11a/b/g Wireless CardBus Adapter Data Sheet, Available online:
http://www.cisco.com/en/US/prod/collateral/wireless/ps6442/ps4555/ps5818/product_data_sheet09186a00801ebc29.html

[16] F. Fu and M. van der Schaar, "Structural-aware stochastic control for transmission scheduling," Technical Report. Available online: http://medianetlab.ee.ucla.edu/papers/UCLATechReport_03_11_2010.pdf

[17] E. Evan-Dar and Y. Mansour, "Learning rates for Q-learning," *Journal of Machine Learning Research*, vol. 5, pp. 1-25, 2003.

[18] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 21, pp. 89-96, 2009.

[19] Q. Liu, S. Zhou, and G. B. Giannakis, "Queuing with adaptive modulation and coding over wireless links: cross-layer analysis and design," *IEEE Trans. on Wireless Communications*, vol. 4, no. 3, pp. 1142-1153, May 2005.

[20] D. Krishnaswamy, "Network-assisted link adaptation with power control and channel reassignment in wireless networks," in *Proc. 3G Wireless Conf.*, 2002, pp. 165–170.

[21] K.-B. Song and S. A. Mujtaba, "On the code-diversity performance of bit-interleaved coded OFDM in frequency-selective fading channels," in *Proc. IEEE Veh. Technol. Conf.*, 2003, vol. 1, pp. 572–576.

[22] A. Goldsmith and S.-G. Chua, "Adaptive coded modulation for fading channels," *IEEE Trans. on Communications*, vol. 46, no. 5, pp. 595-602, May 1998.

[23] E. Altman, Constrained Markov Decision Processes. Boca Raton, FL: Chapman and Hall/CRC Press, 1999.

[24] V. S. Borkar, S. P. Meyn, ”The ODE method for convergence of stochastic approximation and reinforcement learning,” *SIAM J. Control Optim*, vol 38, pp. 447-469, 1999.

[25] D. P. Bertsekas, “Dynamic programming and optimal control,” 3rd, Athena Scientific, Massachusetts, 2005.

Chapter 5

Multi-User Cooperative Video Transmission

We investigate the impact of cooperative relaying on uplink and downlink multi-user (MU) wireless video transmission. The objective is to maximize the long-term sum of utilities across the video terminals in a decentralized fashion, by jointly optimizing the packet scheduling, the resource allocation, and the cooperation decisions, under the assumption that some nodes are willing to act as cooperative relays. A pricing-based distributed resource allocation framework is adopted, where the price reflects the expected future congestion in the network. Specifically, we formulate the wireless video transmission problem as an MU Markov decision process (MDP) that explicitly considers the cooperation at the physical layer and the medium access control sub-layer, the video users' heterogeneous traffic characteristics, the dynamically varying network conditions, and the coupling among the users' transmission strategies across time due to the shared wireless resource. Although MDPs notoriously suffer from the curse of dimensionality, our study shows that, with appropriate simplications and approximations, the complexity of the MU-MDP can be significantly mitigated. Our simulation results demonstrate that integrating cooperative decisions into the MU-MDP optimization increases the resource price in congested networks and decreases

the price in uncongested ones. Additionally, our results show that cooperation allows users with feeble direct signals to achieve improvements in video quality on the order of $5 - 10$ dB peak signal-to-noise ratio, with less than 0.8 dB quality loss by users with strong direct signals.

5.1 Introduction

Existing wireless networks provide dynamically varying resources with only limited support for the Quality of Service (QoS) required by delay-sensitive, bandwidth-intense, and loss-tolerant multimedia applications. This problem is further exacerbated in multi-user (MU) settings because they require multiple video streams, with heterogeneous traffic characteristics, to share the scarce wireless resources. To address these challenges, a lot of research has focused on MU wireless communication [1, 2, 3, 4, 5] and, in particular, MU video streaming over wireless networks [6, 7, 8, 9, 10]. The majority of this research relies on cross-layer adaptation to match available system resources (e.g., bandwidth, power, or transmission time) to application requirements (e.g., delay or source rate), and vice versa. In MU video streaming applications [6, 7, 8, 9, 10], for example, cross-layer optimization is deployed to strike a balance between scheduling the *easy customers*, i.e., those users who experience very good fades, and serving the *most important customers*, i.e., those users who have the highest priority video data to transmit. This tradeoff is important because rewarding a few lucky participants, as opportunistic multiple access policies do [2, 3, 4], does not translate to providing good quality to the application (APP)

layer. Unfortunately, with the exception of [5, 11], the aforementioned research assumes that wireless users are *noncooperative*. This leads to a basic inefficiency in the way that the network resources are assigned: indeed, good fades experienced by some nodes can go to waste because users with higher priority video data, but worse fades, get access to the shared wireless channel.

A way to not let good fades go to waste is to enlist the nodes that experience good fades as cooperative helpers, using a number of techniques available for cooperative coding [12, 13, 14]. As mentioned above, this idea has been considered in [5, 11]. In [11], for example, a cross-layer optimization is proposed involving the physical (PHY) layer, the medium access control (MAC) sublayer, and the APP layer, where layered video coding is integrated with randomized cooperation to enable efficient video multicast in a cooperative wireless network. Because it is a multicast system, there is no need for an optimal multiple-access strategy, and no need to worry about heterogeneous traffic characteristics. In [5], a centralized network utility maximization (NUM) framework is proposed for jointly optimizing relay strategies and resource allocations in a cooperative orthogonal frequency-division multiple-access (OFDMA) network. The cross-layer problem is decomposed into two subproblems using a set of dual variables coordinating the APP layer demand and PHY layer supply of rates. The PHY layer subproblem is then solved by introducing another set of dual variables to relax each users' power constraint. In both [5, 11], it is assumed that each user has a static utility function of the average transmission rate, where the utility derived by each user in [11] is a function of the average received rate of the base and enhancement

layer video bitstreams.

Unlike the abovementioned solutions, we take a dynamic optimization approach to the cooperative MU video streaming problem. Our solution is inspired by the cross-layer resource allocation and scheduling solution in [10], in which the MU wireless video streaming problem is modeled and solved as an MU Markov decision process (MDP) that allows the users, via a uniform resource pricing solution, to obtain long-term optimal video quality in a distributed fashion. Unlike [5, 11], the solution that we adopt from [10] explicitly considers packet-level video traffic characteristics (instead of flow-level) and dynamic network conditions (instead of average case conditions). However, as recently shown in [15], augmenting the framework developed in [10] to also account for cooperation is challenging because of the complexity of the resulting cross-layer MU-MDP optimization.

The contributions of this chapter are threefold. First, we formulate the cooperative wireless video transmission problem as an MU-MDP using a time-division multiple-access (TDMA)-like network, randomized space-time block coding (STBC) [16], and a decode-and-forward cooperation strategy. To the best of our knowledge, we are the first to consider cooperation in a dynamic optimization framework. We show analytically that the decision to cooperate can be made opportunistically, independently of the MU-MDP. Consequently, each user can determine its optimal scheduling policy by only keeping track of its experienced cooperative transmission rates, rather than tracking the channel statistics throughout the network. Second, in light of the fact that opportunistic cooperation is optimal, we propose a low complexity opportunistic

cooperative strategy for exploiting good fades in an MU wireless network. The key idea is that nodes can, in a distributed manner, self-select themselves to act as cooperative relays. The proposed self-selection strategy requires a number of message exchanges that is linear in the number of video sources, and selects sets of cooperative relays in such a way that cooperation can be guaranteed to be better than direct transmission. Third, we show experimentally that users with feeble direct signals to the access point (AP) are conservative in their resource usage when cooperation is disabled. In contrast, when cooperation is enabled, users with feeble direct signals to the AP use cooperative relays and utilize resources more aggressively. Consequently, the uniform resource price that is designed to manage resources in the network tends to increase when cooperation is enabled in a congested network.

The remainder of the chapter is organized as follows. We introduce the system and application models in Section 5.2 and Section 5.3, respectively. In Section 5.4, we present the proposed MU cross-layer PHY/MAC/APP optimization. In Section 5.5, we compute the transmission and packet error rates for both direct and cooperative transmission modes, and propose a distributed protocol for opportunistically recruiting cooperative relays. Finally, we report numerical results in Section 5.6 and conclude in Section 5.7.

5.2 System Model

We consider a network composed of M users streaming video content over a shared wireless channel to a single AP (see Fig. 5.1). Such a scenario is typical of many

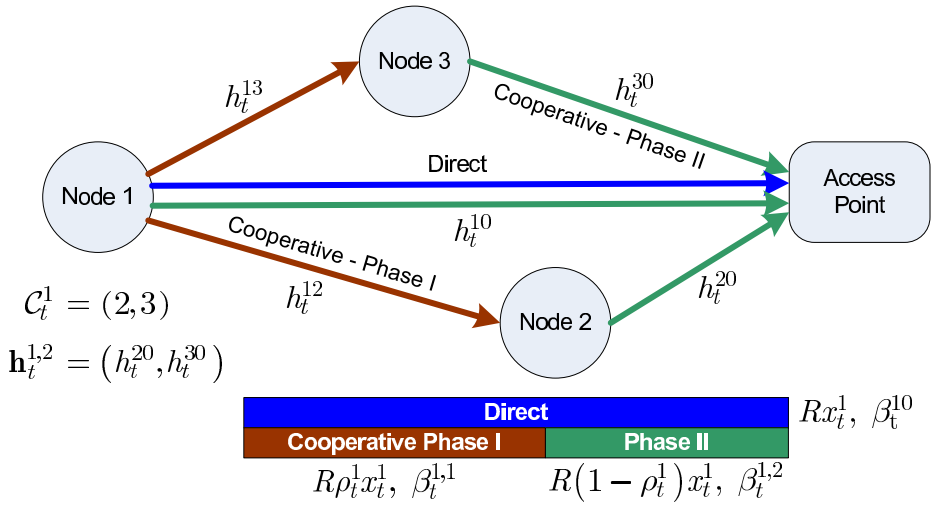


Figure 5.1: An uplink wireless video network with cooperation. A downlink wireless video network with cooperation can be visualized by switching the positions of node 1 and the access point.

uplink media applications, such as remote monitoring and surveillance, wireless video sensors, and mobile video cameras. The proposed optimization framework can also be used for downlink applications, where the relays can be recruited for streaming video to a certain user in the network in exactly the same way that they can be recruited to transmit to the AP in the uplink scenario.

We assume that time is slotted into discrete time-intervals of length $R > 0$ seconds and each time slot is indexed by $t \in \mathbb{N}$.[1] At the MAC layer, the users access the shared channel using a TDMA-like protocol. In each time slot t, the AP endows

[1] The fields of complex, real, and nonnegative integer numbers are denoted with $\mathbb{C}$, $\mathbb{R}$, and $\mathbb{N}$, respectively; matrices [vectors] are denoted with upper [lower] case boldface letters (e.g., $\mathbf{A}$ or $\mathbf{x}$); the field of $m \times n$ complex [real] matrices is denoted as $\mathbb{C}^{m\times n}$ [$\mathbb{R}^{m\times n}$], with $\mathbb{C}^m$ [$\mathbb{R}^m$] used as a shorthand for $\mathbb{C}^{m\times 1}$ [$\mathbb{R}^{m\times 1}$]; the superscript T denotes the transpose of a vector; $|\cdot|$ denotes the magnitude of a complex number; $\|\mathbf{x}\|_1$ is the l_1 norm of the vector $\mathbf{x} \in \mathbb{C}^n$, which for positive real-valued vectors is simply the sum of the components, whereas $\|\mathbf{x}\|_2$ is the Euclidean norm of $\mathbf{x} \in \mathbb{C}^n$; $\{\mathbf{A}\}_{ij}$ indicates the $(i+1, j+1)$th element of the matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$, with $i \in \{0, 1, \ldots, m-1\}$ and $j \in \{0, 1, \ldots, n-1\}$; a circular symmetric complex Gaussian random variable X with mean μ and variance σ^2 is denoted as $X \sim \mathcal{CN}(\mu, \sigma^2)$; $\lfloor\cdot\rfloor$ denotes flooring-integer; E[$\cdot$] stands for ensemble averaging; and, finally, $[\cdot]^+ = \max(\cdot, 0)$.

the ith user, for $i \in \{1, \cdots, M\}$ with the resource fraction x_t^i, where $0 \leq x_t^i \leq 1$, such that the user can use the amount of channel time $R\,x_t^i$ for transmission. Let $\mathbf{x}_t \triangleq (x_t^1, x_t^2, \ldots, x_t^M)^T \in \mathbb{R}^M$ denote the resource allocation vector at time slot t, which must satisfy the stage resource constraint $\|\mathbf{x}_t\|_1 = \sum_{i=1}^M x_t^i \leq 1$, where the inequality accounts for possible signaling overhead. Each node's PHY layer is assumed to be a single-carrier single-input single-output system designed to handle quadrature amplitude modulation (QAM) square constellations, with a (fixed) symbol rate of $1/T_s$ symbols per second and with the average transmitter energy fixed for all the nodes and data rates. We consider a frequency non-selective block fading model, where $h_t^{i\ell} \in \mathbb{C}$ denotes the fading coefficient over the $i \to \ell$ link in time slot t, with $i \neq \ell \in \{0, 1, 2, \ldots, M\}$, and $i = 0$ or $\ell = 0$ corresponding to the AP. It is assumed that all the channels are dual, i.e., $|h_t^{i\ell}| = |h_t^{\ell i}|$, and that the fading coefficients $h_t^{i\ell}$ are i.i.d. with respect to t. Moreover, we define $\mathbf{H}_t \in \mathbb{C}^{M \times M}$ as the matrix collecting the fading coefficients among all of the nodes and the AP, i.e., $\{\mathbf{H}_t\}_{i\ell} = h_t^{i\ell}$, for $i \neq \ell \in \{0, 1, 2, \ldots, M\}$.

At the PHY layer, there are two transmission modes to choose from: direct and cooperative. In the *direct* transmission mode, as shown in Fig. 5.1, the ith source node transmits directly to the AP at the data rate (per unit of bandwidth) β_t^{i0}, measured in bits/second/Hz, for the assigned transmission time $R\,x_t^i$. In the *cooperative* transmission mode, some nodes serve as decode-and-forward relays. Specifically, in the cooperative mode, the assigned transmission time is divided into two phases as illustrated in Fig. 5.1: in *Phase I*, the ith source node directly broadcasts its own data to all the

nodes in the network at the data rate $\beta_t^{i,1}$ (bits/second/Hz) for $R\,\rho_t^i\,x_t^i$ seconds, where $0 < \rho_t^i < 1$ is the Phase I time fraction; in *Phase II*, some of the nodes overhearing the source transmission, belonging to a certain subset $\mathcal{C}_t^i \subseteq \{1, 2, \ldots, M\} - \{i\}$, demodulate the data received in Phase I, re-modulate the original source bits, and then cooperatively transmit towards the AP, along with the original source i, at the data rate $\beta_t^{i,2}$ (bits/second/Hz) for the remaining $R\,(1-\rho_t^i)\,x_t^i$ seconds. Thus, the *cooperative data rate* $\beta_t^{i,\text{coop}}$ (bits/second/Hz) over the two phases is a convex combination of the data rates attainable in each of these two hops, i.e., $\beta_t^{i,\text{coop}} = \rho_t^i\,\beta_t^{i,1} + (1-\rho_t^i)\,\beta_t^{i,2}$.

The decision to transmit in the direct or cooperative transmission mode depends on fading coefficients throughout the network in time slot t and on the target packet error rate (PER). Thus, the actual transmission rate of the ith source in time slot t is dictated by cooperation decision $z_t^i \in \{0, 1\}$, where $z_t^i = 1$ if cooperation is chosen, and $z_t^i = 0$ if direct transmission is chosen. In Section 5.5, we compute the transmission parameters β_t^{i0}, $\beta_t^{i,1}$, and $\beta_t^{i,2}$ as functions of a subset of the elements in the channel matrix $\mathbf{H}_t$, and describe how to determine the set of cooperative relays $\mathcal{C}_t^i$ and the cooperation decision z_t^i.

5.3 Application Model

Due to the video application's real-time requirements and the fact that not all video bits are equally important, cooperative throughput gains alone do not translate to good video quality at the APP layer. For this reason, we develop herein a dynamic video traffic model that we will use to optimize the resource acquisition and packet

scheduling in the face of stochastically varying wireless resources. Before proceeding further, we define the characteristics of the ith user's video data, for $i \in \{1, 2, \ldots, M\}$.

5.3.1 Video Data Attributes

We assume that the ith user's video is encoded using a fixed, periodic, group of pictures (GOP) structure that contains N^i frames and lasts a period of T^i time slots. Each video frame is characterized by five attributes: size, arrival time, deadline, distortion impact, and dependency. We denote with $l_j^{i,\max}$ the maximum video frame *size* in packets. Each frame $j \in \{1, 2, \ldots, N\}$ is packetized into $l_j^i \in \{1, 2, \ldots, l_j^{i,\max}\}$ packets of P bits, where l_j^i is an i.i.d. random variable with respect to i and j. The *arrival time* t_j^i (in time slots) denotes the earliest time at which frame j can be transmitted. The *deadline* d_j^i (in time slots) denotes the time at which frame j must be decoded by the receiver. The *distortion impact* q_j^i represents the distortion reduction observed when a packet in frame j is received before its deadline d_j^i. The packets must be decoded at the receiver in decoding order, which is dictated by the *dependencies* introduced by predictive coding (e.g., motion-compensation). In general, the dependencies among frames for different GOP structures can be described by a directed acyclic graph (DAG) [17] with the nodes representing frames and the edges representing the dependencies among frames. We borrow the notation $k \prec j$ from [17] to indicate that frame j depends on frame k (i.e., there exists a path directed from k to j) and cannot be decoded until k is decoded.

5.3.2 Traffic Model and Packet Scheduling

For $i \in \{1, 2, \ldots, M\}$, the *traffic state* $\mathcal{T}_t^i$ of the ith user represents the video data that the ith user can potentially transmit in time slot t, and comprises the following two components: the schedulable frame set $\mathcal{F}_t^i$ and the buffer state $\mathbf{b}_t^i$. In time slot t, we assume that the ith user can transmit the set of frames whose deadlines are within the scheduling time window (STW) $[t, t+W]$. Thus, the schedulable frame set is defined as $\mathcal{F}_t^i \triangleq \left\{ j \,|\, d_j^i \in \{t, t+1, \ldots, t+W\} \right\}$. The STW is chosen such that if frame j directly depends on frame k (i.e., there is a directed arc from k to j in the DAG), then $W > d_j^i - d_k^i$. This condition guarantees that frame j and k are within the same STW in at least one time slot, and ensures that the traffic state transition is Markovian.[2] Because the GOP structure is fixed and periodic, $\mathcal{F}_t^i$ is periodic with some period T^i. Each frame's arrival time t_j^i and deadline d_j^i is fully determined by the periodic GOP structure. Specifically, it turns out that $t_j^i \triangleq \min_{j \in \mathcal{F}_t^i} t$ and $d_j^i \triangleq \max_{j \in \mathcal{F}_t^i} t$. The buffer state $\mathbf{b}_t^i \triangleq (b_{t,j}^i \,|\, j \in \mathcal{F}_t^i)^T$ represents the number of packets of each frame in the STW that are awaiting transmission at time t. The jth component $b_{t,j}^i$ of $\mathbf{b}_t^i$ denotes the number of packets of frame $j \in \mathcal{F}_t^i$ remaining for transmission at time t. By definition, $b_{t,j}^i \leq l_j^i$, where l_j^i is the total number of packets associated with frame j. Thus, the traffic state is defined as $\mathcal{T}_t^i \triangleq \{\mathcal{F}_t^i, \mathbf{b}_t^i\}$. Fig. 5.2

[2]Larger STWs allow more packets to be transmitted in each time slot, but the number of buffer states $\mathbf{b}_t^i$ is exponential in the cardinality of $\mathcal{F}_t^i$, which increases with W. To mitigate the size of the state-space and the complexity of the optimization in Section 5.4, we set the STW to the minimum value of W that satisfies $W > d_j^i - d_k^i$. In the congested operating regime for which the proposed MU-MDP is most beneficial, it is not necessary to have a large STW because there are not enough resources to transmit all of the packets in the STW.

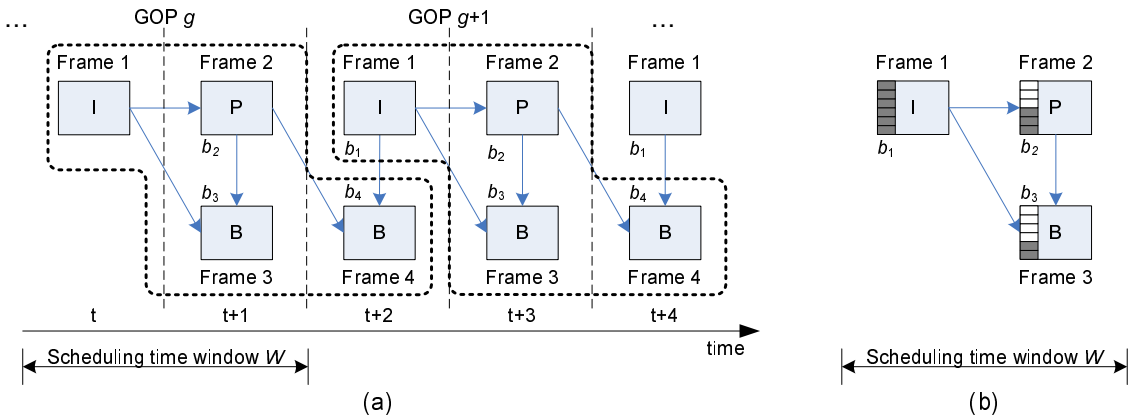


Figure 5.2: (a) Illustrative DAG dependencies and scheduling time window using IBPB GOP structure. The schedulable frame sets defined by the scheduling time window W are $\mathcal{F}_t = \{1, 2, 3\}$, $\mathcal{F}_{t+1} = \{2, 3, 4, 1\}$, $\mathcal{F}_{t+2} = \{4, 1, 2, 3\}$, $\mathcal{F}_{t+3} = \{2, 3, 4, 1\}$, etc. Clearly, $\mathcal{F}_t$ is periodic with period $T = 2$ excluding the initial time t, and each GOP contains $N = 4$ frames. (b) Traffic state detail for schedulable frame set $\mathcal{F}_t = \{1, 2, 3\}$. b_j denotes the state of the jth frame's buffer, where $j \in \mathcal{F}_t = \{1, 2, 3\}$.

illustrates how the traffic states are defined for a simple IBPB GOP structure.[3]

We now define the packet scheduling action, and show how this action impacts the traffic state transition. In each time slot t, the ith user takes scheduling action $\mathbf{y}_t^i \triangleq (y_{t,j}^i \,|\, j \in \mathcal{F}_t^i)^T$, which determines the number of packets to transmit out of $\mathbf{b}_t^i$. Specifically, the jth component $y_{t,j}^i$ of $\mathbf{y}_t^i$ represents the number of packets of the jth frame within the STW that are scheduled to be transmitted in time slot t. Importantly, the scheduling action $\mathbf{y}_t^i$ is constrained to be in the feasible scheduling action set $\mathcal{P}^i(\mathcal{T}_t^i, \mathbf{H}_t)$, which depends on the traffic state $\mathcal{T}_t^i$ and network state $\mathbf{H}_t$. In particular, the following three constraints must be met:

1. *Buffer*: Every component of $\mathbf{y}_t^i$ must satisfy $0 \leq y_{t,j}^i \leq b_{t,j}^i$.

[3]In a typical hybrid video coder like H.264/AVC or MPEG-2, I, P, and B indicate the type of motion prediction used to exploit temporal correlations between video frames. I-frames are compressed independently of the other frames, P-frames are predicted from previous frames, and B-frames are predicted from previous and future frames.

2. *Packet*: The total number of transmitted packets must satisfy

$$\|\mathbf{y}_t^i\|_1 = \sum_{j \in \mathcal{F}_t^i} y_{t,j}^i \leq \frac{R\,\beta_t^i}{P\,T_s}, \tag{5.1}$$

where $\beta_t^i = \beta_t^{i0}$ in the direct transmission mode, i.e., when $z_t^i = 0$, and $\beta_t^i = \beta_t^{i,\text{coop}}$ in the cooperative transmission mode, i.e., when $z_t^i = 1$. Note that β_t^i depends on a subset of the elements in $\mathbf{H}_t$ as described later in Section 5.5.

3. *Dependency*: If there exists a $k \prec j$ that has not been transmitted, then $\left(b_{t,k}^i - y_{t,k}^i\right) y_{t,j}^i = 0$. In other words, all packets associated with k must be transmitted before transmitting any packets associated with j.

The transition of the schedulable frame set from time t (i.e., $\mathcal{F}_t^i$) to time $t+1$ (i.e., $\mathcal{F}_{t+1}^i$) is independent of the scheduling action. In fact, it is deterministic and periodic for a fixed GOP structure, and can therefore be modeled as a deterministic Markov process with transition probability function $p(\mathcal{F}_{t+1}^i \,|\, \mathcal{F}_t^i) = I\left[\mathcal{F}_{t+1}^i = \text{next}(\mathcal{F}_t^i)\right]$, where $I[\cdot]$ is an indicator function that takes value 1 if $\mathcal{F}_{t+1}^i$ is the schedulable frame set that proceeds $\mathcal{F}_t^i$ in the GOP structure, denoted by $\mathcal{F}_{t+1}^i = \text{next}(\mathcal{F}_t^i)$, and takes value 0 otherwise.

Unlike the schedulable frame set transition, the buffer state transition depends on the scheduling action. Under the assumption that l_j^i is i.i.d. with respect to i and j, the sequence of buffer states $\{\mathbf{b}_t^i : t \in \mathbb{N}\}$ can be modeled as a controllable Markov process with transition probabilities $p(\mathbf{b}_{t+1}^i \,|\, \mathbf{b}_t^i, \mathbf{y}_t^i)$, where the controlling variable is the scheduling action $\mathbf{y}_t^i$. Before we can compute the transition from $\mathbf{b}_t^i$ to $\mathbf{b}_{t+1}^i$, we

need to define a partition of the schedulable frame set $\mathcal{F}_{t+1}^i$. The partition divides $\mathcal{F}_{t+1}^i$ into two sets: a set of frames that persist from time t to $t+1$ because they have deadlines $d_j^i > t$, i.e., $\mathcal{F}_t^i \cap \mathcal{F}_{t+1}^i$; and, a set of newly arrived frames with arrival times $t_j^i = t+1$, i.e., $\mathcal{F}_{t+1}^i \setminus \mathcal{F}_t^i \triangleq \mathcal{F}_{t+1}^i - \mathcal{F}_t^i \cap \mathcal{F}_{t+1}^i$. Based on this partition, $b_{t+1,j}^i$ can be determined from $b_{t,j}^i$ and $y_{t,j}^i$ as follows

$$b_{t+1,j}^i = \begin{cases} b_{t,j}^i - y_{t,j}^i \,, & \text{if } j \in \mathcal{F}_t^i \cap \mathcal{F}_{t+1}^i \,; \\ l_j^i \,, & \text{if } j \in \mathcal{F}_{t+1}^i \setminus \mathcal{F}_t^i \,, \end{cases} \tag{5.2}$$

where $\mathbf{y}_t^i$ satisfies the buffer, packet, and dependency constraints described above. Because both $\mathcal{F}_t^i$ and $\mathbf{b}_t^i$ are Markov processes, the sequence of traffic states $\{\mathcal{T}_t^i : t \in \mathbb{N}\}$ can be modeled as a controllable Markov process with transition probabilities $p(\mathcal{T}_{t+1}^i \,|\, \mathcal{T}_t^i, \mathbf{y}_t^i) = p(\mathcal{F}_{t+1}^i \,|\, \mathcal{F}_t^i)\, p(\mathbf{b}_{t+1}^i \,|\, \mathbf{b}_t^i, \mathbf{y}_t^i)$.

5.4 Cooperative Multi-User Video Transmission

Recall that $\mathcal{T}_t^i$ denotes the ith user's traffic state and $\mathbf{H}_t$ collects the channel coefficients among all the nodes and the AP. Hence, the global state can be defined as $\mathbf{s}_t \triangleq \{\mathcal{T}_t^1, \mathcal{T}_t^2, \ldots, \mathcal{T}_t^M, \mathbf{H}_t\} \in \mathcal{S}$, where $\mathcal{S}$ is a discrete set of all possible states.[4] Since: (i) the ith user's traffic state evolves as a Markov process controlled by its scheduling action $\mathbf{y}_t^i$; (ii) the ith user's traffic state transition is conditionally independent of the other users' traffic state transitions given $\mathbf{y}_t^i$; and (iii) the state of each $i \to \ell$ link $h_t^{i\ell}$ is assumed to be i.i.d. with respect to time; the sequence of global states $\{\mathbf{s}_t : t \in \mathbb{N}\}$

[4]To have a discrete set of network states, the individual link states in $\mathbf{H}_t$ are quantized into a finite number of bins (see [24] for details).

can be modeled as a controlled Markov process with transition probability function

$$p(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, \mathbf{y}_t) = p\left(\mathbf{H}_{t+1}\right) \prod_{i=1}^{M} p(\mathcal{T}_{t+1}^i \,|\, \mathcal{T}_t^i, \mathbf{y}_t^i)\,, \tag{5.3}$$

where $\mathbf{y}_t \triangleq (\{\mathbf{y}_t^1\}^T, \{\mathbf{y}_t^2\}^T, \ldots, \{\mathbf{y}_t^M\}^T)^T$ collects the scheduling actions of all the video users. Moreover, under the scheduling action $\mathbf{y}_t^i$, the ith user obtains the immediate utility $u^i(\mathcal{T}_t^i, \mathbf{y}_t^i) \triangleq \sum_{j \in \mathcal{F}_t^i} q_j^i \, y_{t,j}^i$, which is the total video quality improvement experienced by the ith user by taking scheduling action $\mathbf{y}_t^i$ in traffic state $\mathcal{T}_t^i$ under the assumption that quality is incrementally additive [17].

The objective of the MU optimization is the maximization of the *expected discounted sum of utilities* with respect to the joint scheduling action $\mathbf{y}_t$ and the cooperation decision vector $\mathbf{z}_t \triangleq (z_t^1, z_t^2, \ldots, z_t^M)^T$ taken in each state $\mathbf{s}_t$. Due to the stationary Markovian transition probability function, the optimization can be formulated as an MDP that satisfies the following dynamic programming equation[5]

$$U^*(\mathbf{s}) = \max_{\mathbf{y},\mathbf{z}} \left\{ \sum_{i=1}^{M} u^i(\mathcal{T}^i, \mathbf{y}^i) + \alpha \sum_{\mathbf{s}' \in \mathcal{S}} p(\mathbf{H}') \prod_{i=1}^{M} p(\mathcal{T}^{i\prime} \,|\, \mathcal{T}^i, \mathbf{y}^i)\, U^*(\mathbf{s}') \right\}, \forall \mathbf{s}, \tag{5.4}$$

subject to

$$\mathbf{y}^i \in \mathcal{P}^i(\mathcal{T}^i, \mathbf{H}) \quad \text{and} \quad \sum_{i=1}^{M} x^i \leq 1, \tag{5.5}$$

where x^i is the time-fraction allocated to the ith user given its scheduling action $\mathbf{y}^i$

[5]In this section, since we model the problem as a stationary MDP, we omit the time index when it does not create confusion. In place of the time index, we use the notation $(\cdot)'$ to denote a state variable in the next time step (e.g. $\mathcal{T}^{i\prime}$, $\mathbf{H}'$, $\mathbf{s}'$).

and transmission rate β^i, i.e.,

$$x^i = \frac{P\,T_s}{R\,\beta^i}\,\|\mathbf{y}^i\|_1\,, \tag{5.6}$$

the parameter $\alpha \in [0, 1)$ is the "discount factor", which accounts for the relative importance of the present and future utility, and $\mathcal{P}^i(\mathcal{T}^i, \mathbf{H})$ is the set of feasible scheduling actions given the traffic state $\mathcal{T}^i$ and channel state matrix $\mathbf{H}$.

Given the distributions $p(\mathbf{H})$ and $p(\mathcal{T}^{i\prime} \,|\, \mathcal{T}^i, \mathbf{y}^i)$ for all i, the above MU-MDP can be solved by the AP using value iteration or policy iteration [18]. However, there are two challenges associated with solving the above MU-MDP. First, the complexity of solving an MDP is proportional to the cardinality of its state-space $\mathcal{S}$, which, in the above MU-MDP, scales exponentially with the number of users, i.e., M, and with the number of links in $\mathbf{H}$, i.e., M^2. Hence, even for moderate sized networks, it is impractical to compute, or even to encode, $U^*(\mathbf{s})$. In subsection 5.4.1, we show that the exponential dependence on the number of links in $\mathbf{H}$ can be eliminated. Second, in the uplink scenario, the traffic state information is local to the users, so neither the AP nor the users have enough information to solve the above MU-MDP. In subsection 5.4.2, we summarize the findings in [10] that show that the considered optimization can be approximated to make it amenable to a distributed solution. Additionally, this distributed solution eliminates the exponential dependence on the number of users.

5.4.1 Reformulation with Simplified Network State

The only reason to include the detailed network state information $\mathbf{H}$ and the cooperation decision $\mathbf{z}$ in the MU-MDP is to make foresighted cooperation decisions, which take into account the impact of the immediate cooperation decision on the expected future utility of the users. However, if we can show that the optimal opportunistic (i.e., myopic) cooperation decision is also long-term optimal, then the detailed network state information does not need to be included in the MU-MDP. The following theorem shows that the optimal opportunistic cooperation decision, which maximizes the immediate transmission rate, is also long-term optimal.

Theorem 5.1 (Opportunistic cooperation is optimal) *If utilizing cooperation incurs zero cost to the source and relays, then the optimal opportunistic cooperation decision, which maximizes the immediate throughput, is also long-term optimal.*

Proof: See Appendix F.

To intuitively understand why maximizing the immediate transmission rate at the PHY layer is long-term optimal, consider what happens when a user chooses not to maximize its immediate transmission rate (i.e., does not utilize the optimal opportunistic cooperation decision). Two things can happen: either less packets are transmitted overall because of packet expirations; or, the same number of packets are transmitted overall, but their transmission incurs additional resource costs because transmitting the same number of packets at a lower rate requires more resources [see (5.6)]. In either case, the long-term utility is suboptimal. A consequence of

Theorem 5.1 is that the cooperation decision vector $\mathbf{z}$ does not need to be included in the MU-MDP. Instead, it can be determined opportunistically by selecting $\mathbf{z}$ to maximize the immediate transmission rate. Most importantly, this means that the MU-MDP does not need to include the high-dimensional network state.

We now make two remarks regarding Theorem 5.1 so that its consequences are not misinterpreted. First, in the introduction, we noted that maximizing throughput is a suboptimal multiple access strategy for wireless video. This does not contradict Theorem 5.1 because it only states that the *cooperation decision* should be made opportunistically to maximize the immediate transmission rate. Indeed, myopic (opportunistic) resource allocation and scheduling is suboptimal because it does not take into account the dynamic video data attributes (i.e., deadlines, priorities, and dependencies). Second, although the users' MDPs do not need to include the high-dimensional network state, the optimal resource allocation and scheduling strategies still depend on it; however, instead of tracking $\mathbf{H}_t$, it is sufficient to track the users' optimal opportunistic transmission rates provided by the PHY layer, i.e., β_t^i for all i. Under the assumption that the channel coefficients are i.i.d. random variables with respect to t, β_t^i can also be modeled as an i.i.d. random variable with respect to t. We let $p(\beta^i)$ denote the probability mass function (pmf) from which β_t^i is drawn. We note that $p(\beta^i)$ depends on $p(\mathbf{H})$ and the deployed PHY layer cooperation algorithm.

Based on the second remark, we can simplify the maximization problem in (5.4). Let us define the ith user's state as $s^i \triangleq (\mathcal{T}^i, \beta^i) \in \mathcal{S}^i$ and redefine the global state as $\mathbf{s} \triangleq (s^1, \ldots, s^M)^T$. In Section 5.5, we describe how β^i is determined, but for now

we will take for granted that it is known. Because the optimization does not need to include the cooperation decision, the maximization of the expected sum of discounted utilities in (5.4) can be simplified by only maximizing with respect to the scheduling action $\mathbf{y}$ in each state $\mathbf{s}$, that is,

$$U^*(\mathbf{s}) = \max_{\mathbf{y}} \left\{ \sum_{i=1}^{M} u^i(\mathcal{T}^i, \mathbf{y}^i) + \alpha \sum_{\mathbf{s}' \in \mathcal{S}} \prod_{i=1}^{M} p(s^{i\prime} \mid s^i, \mathbf{y}^i)\, U^*(\mathbf{s}') \right\}, \forall \mathbf{s}, \tag{5.7}$$

subject to

$$\mathbf{y}^i \in \mathcal{P}^i(\mathcal{T}^i, \beta^i), \qquad \sum_{i=1}^{M} x^i \leq 1, \tag{5.8}$$

where $p(s^{i\prime} \mid s^i, \mathbf{y}^i) = p(\beta^{i\prime})\, p(\mathcal{T}^{i\prime} \mid \mathcal{T}^i, \mathbf{y}^i)$. Although we have eliminated the exponential dependence on the number of links in the network, the complexity of solving (5.7) in both the uplink and downlink scenarios still scales exponentially with the number of users. Moreover, (5.7) must be solved in a centralized fashion by the AP, which does not have direct access to the users' local states in the uplink scenario. Thus, significant information exchange overheads are required because, in every time slot, the users must communicate their states to the AP and the AP must communicate their optimal actions back to them.

5.4.2 Distributed Solution

The optimization problem in (5.7) is equivalent to the so-called multiuser primary problem with stage resource constraints (MUP/SRC) considered in [10]. In [10], it is shown that (5.7) can be reformulated as an unconstrained MDP using Lagrangian relaxation. The key idea is to introduce a Lagrange multiplier λ associated with the

stage resource constraint $\sum_{i=1}^{M} x^i \leq 1$ in each global state $\mathbf{s}$ because every global state has a different resource-quality tradeoff. The resulting dual solution has zero duality gap compared to the primary problem [i.e., (5.7)], but it still depends on the global state so it is not amenable to a distributed solution. However, by imposing a uniform resource price λ, which is independent of the multi-user state, the resulting MU-MDP can be decomposed into M MDPs, one for each user [10].[6] These local MDPs satisfy the following dynamic programming equation

$$U^{i,*}(s^i, \lambda) = \max_{\mathbf{y}^i} \left[u^i(\mathcal{T}^i, \mathbf{y}^i) - \lambda \left(x^i - \frac{1}{M} \right) + \alpha \sum_{s^{i\prime} \in \mathcal{S}} p(s^{i\prime} \mid s^i, \mathbf{y}^i)\, U^{i,*}(s^{i\prime}, \lambda) \right], \quad (5.9)$$

$$\hat{U}^{\lambda^*}(\mathbf{s}) = \min_{\lambda \geq 0} \sum_{i=1}^{M} U^{i,*}(s^i, \lambda)\,, \quad (5.10)$$

subject to $\mathbf{y}^i \in \mathcal{P}^i(\mathcal{T}^i, \beta^i)$. Importantly, the ith user's dynamic programming equation defines the optimal scheduling action as a function of the ith user's state, rather than the global state $\mathbf{s}$. In this chapter, the ith user solves (5.9) offline using value iteration; however, it can be easily solved online using reinforcement learning as in [10] and [19].

Although the optimization can be decomposed across the users, the optimal resource price λ still depends on all of the users' resource demands. Hence, λ must be determined by the AP in both the uplink and downlink scenarios. Specifically, the resource price can be numerically computed by the AP using the subgradient method. The subgradient with respect to λ is given by $\sum_{i=1}^{M} X^i - \frac{1}{1-\alpha}$, where

[6] We note that the resource price is only used to efficiently allocate the limited wireless resources among the users; it is not used to generate revenue for the AP. In other words, it is a congestion price rather than a real price.

$X^i = E\left[\sum_{t=0}^{+\infty} \alpha^t x_t^i \,|\, s_0^i\right]$ is the ith user's expected discounted accumulated resource consumption, which can be calculated as described in [10]. Importantly, X^i can be computed locally by the ith user in the uplink scenario and by the AP in the downlink scenario. Using the subgradient method, the resource price is updated as

$$\lambda^{k+1} = \left[\lambda^k + \mu^k \left(\sum_{i=1}^{M} X^i - \frac{1}{1-\alpha}\right)\right]^+, \tag{5.11}$$

where μ^k is a diminishing step size. Since the focus of this chapter is on the interaction between the multiuser video transmission and the cooperative PHY layer, we refer the interested reader to [10] for complete details on the dual decomposition outlined in this subsection, and the derivation of the subgradient with respect to λ.

We note that a similar decomposition has recently been proposed for energy-efficient uplink scheduling with delay constraints in multiuser wireless networks using a different MU-MDP framework [19]. Besides the fact that [19] does not consider physical layer cooperation or heterogeneous traffic characteristics, there is one significant difference between the decomposition in [19] and the one adopted in this chapter. Specifically, the TDMA-like protocol in [19] assumes that only one user can transmit in each time slot, whereas we consider a TDMA-like protocol in which each time slot is divided into different length transmission opportunities for each user. As a result, in [19], every user has a unique Lagrange multiplier associated with its average buffer delay constraint. In contrast, in our decomposition, all users have the same Lagrange multiplier, which regulates the resource division among the users, rather than their individual delay constraints. Note that, in this chapter, delay constraints are included

in the application model.

5.5 Cooperative PHY Layer Transmission

The ith user's packet scheduling decision $\mathbf{y}_t^i$ is subject to the packet constraint $\|\mathbf{y}_t^i\|_1 \leq \frac{R\,\beta_t^i}{P\,T_s}$, where $\beta_t^i = \beta_t^{i,\text{coop}}$ when $z_t^i = 1$ and the nodes cooperate, and $\beta_t^i = \beta_t^{i0}$ when $z_t^i = 0$ and they do not. Until now, we have assumed that the ith video user knows the opportunistically optimal β_t^i, which, from Theorem 5.1, we know to be long-term optimal. Herein, with reference to the uplink scenario, we describe how β_t^{i0} and $\beta_t^{i,\text{coop}}$ depend on a subset of the elements in the channel state matrix $\mathbf{H}_t$. Then, we define our opportunistic cooperative strategy to select distributively the set of cooperative relays $\mathcal{C}_t^i$ and make the decision z_t^i at the AP. The downlink case is a minor variation.

5.5.1 Data Rate for a Direct Transmission

Let us consider the direct $i \to \ell$ link with instantaneous channel gain $h_t^{i\ell}$ and data rate $\beta_t^{i\ell} \geq 1$ (bits/second/Hz) corrupted by additive white Gaussian noise (AWGN). The bit error probability (BEP) $P_t^{i\ell}(h_t^{i\ell}, \beta_t^{i\ell})$ at the output of the maximum likelihood (ML) detector of the receiving node ℓ, under the assumption that a Gray code is used to map the information bits into QAM symbols and the signal-to-noise ratio (SNR) is sufficiently high, can be upper bounded as (see [20])

$$P_t^{i\ell}(h_t^{i\ell}, \beta_t^{i\ell}) \leq 4\,\exp\left[-\frac{3\,\gamma\,|h_t^{i\ell}|^2}{2\left(2^{\beta_t^{i\ell}}-1\right)}\right], \tag{5.12}$$

where γ is the average SNR per symbol expended by the transmitter. In our framework, each direct transmission is subject to a PER threshold at the MAC sublayer, which leads to a BEP constraint $P_t^{i\ell}(h_t^{i\ell}, \beta_t^{i\ell}) \leq BEP$ at the PHY layer. Consequently, the achievable data rate $\beta_t^{i\ell}$ under the BEP constraint is

$$\beta_t^{i\ell} = \lfloor \log_2 \left(1 + \Gamma \, |h_t^{i\ell}|^2\right) \rfloor \,, \quad \text{where} \quad \Gamma \triangleq \frac{3\,\gamma}{2\,\left|\log_e\left(\frac{BEP}{4}\right)\right|} \,. \tag{5.13}$$

The data rate β_t^{i0} over the link between the source and the AP is obtained using (5.13) by setting $\ell = 0$.

5.5.2 Data Rates for the First and Second Hops of a Cooperative Randomized Transmission

Because of possible error propagation, the end-to-end BEP for a two-hop cooperative transmission is in general cumbersome to calculate exactly with decode-and-forward relays; therefore, the relationship that ties the rates $\beta_t^{i,1}, \beta_t^{i,2}$ and the relevant channel state information, and that guarantees a certain reliability of the overall link, is not as simple as (5.13). To significantly simplify the computation of $\beta_t^{i,1}$ and $\beta_t^{i,2}$, we use two different BEP thresholds BEP_1 and BEP_2 for the first and second hops, respectively, where BEP_1 is a large percentage of the total error rate budget BEP, say $BEP_1 = 0.9\,BEP$, and $BEP_2 = BEP - BEP_1$. Moreover, we reasonably assume that the end-to-end BEP at the output of the ML detector of the AP is dominated by the BEP over the worst source-to-relay link. Under this assumption, accounting

for (5.13), we can estimate $\beta_t^{i,1}$ in Phase I as

$$\beta_t^{i,1} = \left\lfloor \log_2 \left(1 + \Gamma_1 \min_{\ell \in \mathcal{C}_t^i} |h_t^{i\ell}|^2 \right) \right\rfloor , \tag{5.14}$$

where Γ_1 is obtained from Γ by replacing BEP with BEP_1. Since the AP and the relays cannot estimate the channel coefficients $h_t^{i\ell}$, for all $\ell \in \mathcal{C}_t^i$, they cannot determine $\beta_t^{i,1}$ alone. We will deal with this problem in Subsection 5.5.3.

Supposing that a subset $\mathcal{C}_t^i$ of the available nodes are recruited to serve as relays in Phase II, these nodes, along with the ith user, cooperatively forward the source message by using a randomized STBC rule [16]. More specifically, assuming error-free demodulation at the decode-and-forward relays, if $\mathbf{a}_t^i \in \mathbb{C}^K$ denotes the block of i.i.d. QAM symbols transmitted by source i during Phase I of time slot t, then at the ℓth node, for each $\ell \in \{i\} \cup \mathcal{C}_t^i$, the vector $\mathbf{a}_t^i$ is mapped onto an *orthogonal* space-time code matrix $\boldsymbol{\mathcal{G}}(\mathbf{a}_t^i) \in \mathbb{C}^{Q \times L}$ [21], where Q is the block length and L denotes the number of antennas in the underlying space-time code. During Phase II, the ℓth node transmits a linear weighted combination of the columns of $\boldsymbol{\mathcal{G}}(\mathbf{a}_t^i)$, with the weights of the L columns of $\boldsymbol{\mathcal{G}}(\mathbf{a}_t^i)$ contained in the vector $\mathbf{r}_\ell \in \mathbb{C}^L$. We denote with $\mathbf{R} \triangleq (\mathbf{r}_\ell \,|\, \ell \in \mathcal{C}_t^i) \in \mathbb{C}^{L \times N_t^i}$ the weight matrix of all the cooperating nodes, where $N_t^i \leq M$ is the cardinality of $\mathcal{C}_t^i$.[7] Under the randomized STBC rule, the AP observes the space-time coded signal $\boldsymbol{\mathcal{G}}(\mathbf{a}_t^i)$ with *equivalent* channel vector $\widetilde{\mathbf{h}}_t^{i,2} \triangleq h_t^{i0}\,\mathbf{r}_i + \mathbf{R}\,\mathbf{h}_t^{i,2}$, where $\mathbf{h}_t^{i,2} \triangleq (h_t^{\ell 0} \,|\, \ell \in \mathcal{C}_t^i)^T \in \mathbb{C}^{N_t^i}$ collects all the channel coefficients between the relay

[7] One specific code of the STBC matrix is always assigned to the source itself, which transmits over the cooperative link every time cooperation is activated. This can be accounted for by simply setting $\mathbf{r}_i = (1, 0 \ldots, 0)^T$ and replacing the first row of $\mathbf{R}$ with $(0 \ldots, 0)$, whereas the remaining entries of $\mathbf{R}$ are identically and independently generated random variables with zero mean and variance $1/L$.

nodes and the AP (see Fig. 5.1). It is noteworthy that the AP only needs to estimate $\widetilde{\mathbf{h}}_t^{i,2}$ for coherent ML decoding and that the randomized coding is decentralized since the ℓth relay chooses $\mathbf{r}_\ell$ locally.

By capitalizing on the orthogonality of the underlying STBC matrix $\boldsymbol{\mathcal{G}}(\mathbf{a}_t^i)$, the BEP $P_t^{i,2}(\widetilde{\mathbf{h}}_t^{i,2}, \beta_t^{i,2})$ over the second hop at the output of the ML detector of the AP using data rate $\beta_t^{i,2}$ (bits/second/Hz) can be upper bounded as in (5.12) by replacing $|h_t^{i\ell}|^2$ and $\beta_t^{i\ell}$ with $\|\widetilde{\mathbf{h}}_t^{i,2}\|^2$ and $\beta_t^{i,2}$, respectively. By imposing the BEP constraint $P_t^{i,2}(\widetilde{\mathbf{h}}_t^{i,2}, \beta_t^{i,2}) \leq BEP_2$, the data rate $\beta_t^{i,2}$ attainable on the second hop of the cooperating link is given by

$$\beta_t^{i,2} = \lfloor \log_2[1 + \Gamma_2 \left(|h_t^{i0}|^2 + \|\mathbf{R}\,\mathbf{h}_t^{i,2}\|^2\right)] \rfloor \,, \tag{5.15}$$

where Γ_2 is obtained from Γ in (5.13) by replacing BEP with BEP_2. It is interesting to underline that the AP can exactly evaluate $\beta_t^{i,2}$ because it can estimate h_t^{i0} and $\mathbf{R}\,\mathbf{h}_t^{i,2}$ via training as explained in Subsection 5.5.3.

5.5.3 Recruitment Protocol

Recall from the end of Section 5.2 that the cooperative data rate $\beta_t^{i,\text{coop}}$ over the two phases is a convex combination of the attainable data rates $\beta_t^{i,1}$ and $\beta_t^{i,2}$ in the first and second hops. Since K and Q symbols have to be transmitted in Phase I and II, respectively, it is required that $Q\,\rho_t^i\,\beta_t^{i,1} = K\,(1-\rho_t^i)\,\beta_t^{i,2}$, which means that

$$\rho_t^i = \frac{1}{1 + \beta_t^{i,1}/(\beta_t^{i,2}\,R_c)} \quad \Longrightarrow \quad \frac{R_c+1}{\beta_t^{i,\text{coop}}} = \frac{R_c}{\beta_t^{i,1}} + \frac{1}{\beta_t^{i,2}} \,, \tag{5.16}$$

where $R_c \triangleq K/Q \leq 1$ is the rate of the orthogonal STBC rule. The cooperative mode is activated only if the cooperative transmission is more data-rate efficient than the direct communication, i.e., only if $\beta_t^{i,\text{coop}} > \beta_t^{i0}$, which from (5.16) leads to the following condition

$$\frac{R_c}{\beta_t^{i,1}} + \frac{1}{\beta_t^{i,2}} < \frac{R_c + 1}{\beta_t^{i0}} \,. \tag{5.17}$$

If condition (5.17) is not fulfilled, the source transmits to the AP in direct mode.

The trouble in recruiting relays on-the-fly is that the AP cannot directly compute $\beta_t^{i,1}$ given by (5.14). Some MAC randomized protocols have been recently proposed [22, 23], which get round the problem that the AP and the relays do not have the necessary channel state information to determine $\beta_t^{i,1}$. However, such protocols require the exchange and/or the tracking of a large amount of network parameters that may incur unacceptable delays in a wireless video network. To avoid this problem, we propose a much simpler recruitment scheme that is based on (5.14) and (5.15). Specifically, we adopt a MAC solution similar to that proposed in [23]; however, we underline that only the signaling is the same as [23], whereas the parameters exchanged among the source, the relays, and the AP are different because we use a different recruitment policy. The proposed handshaking is reminiscent of the request-to-send (RTS) and clear-to-send (CTS) control messages used in carrier sense multiple access with collision avoidance (CSMA/CA), where all the control frames are transmitted at a low data-rate such that they can be decoded correctly. The thresholds BEP_1 and BEP_2, as well as L and R_c, are fixed parameters that are known at all the nodes. Fig. 5.3 illustrates the signaling protocol for time slot t, which consists of

the following four steps:

Step 1) The ith source initiates the handshaking by transmitting the RTS frame, which announces the will to transmit K data symbols and also includes training symbols that are used by the other nodes to estimate the link gains. From the RTS message, the AP estimates the channel coefficients h_t^{i0} and, hence, determines β_t^{i0}. At the same time, by passively listening to all the RTS messages occurring in the network, the other nodes estimate their respective channel parameters $h_t^{i\ell}$, for $\ell \in \{1, 2, \ldots, M\} - \{i\}$, and, thus, determine $\beta_t^{i\ell}$.

Step 2) The AP responds with a global cooperative recruitment signal (CRS) that provides feedback on β_t^{i0} to all the candidate cooperative nodes, as well as a second parameter $0 < \xi_t \leq \frac{R_c}{1+R_c}$, which is used to recruit relays.

Step 3) The candidate cooperative nodes can self-select themselves according to the following rule:

$$\mathcal{C}_t^i = \left\{ \ell : \frac{\beta_t^{i0}}{\beta_t^{i\ell}} \leq \frac{R_c + 1}{R_c} \xi_t \right\} , \tag{5.18}$$

where $\beta_t^{i\ell}$ is defined using (5.13) by replacing BEP with BEP_1, and the condition defining $\mathcal{C}_t^i$ assures that

$$\frac{R_c}{\beta_t^{i,1}} \leq \xi_t \frac{R_c + 1}{\beta_t^{i0}} . \tag{5.19}$$

The nodes belonging to the formed group $\mathcal{C}_t^i$ send in unison the *help to send* (HTS) message using randomized STBC of size L as described in Subsection 5.5.2, which piggybacks training symbols that are used by the AP to

estimate the cooperative channel vector $\mathbf{R}\,\mathbf{h}_t^{i,2}$.

Step 4) The AP computes the data rate $\beta_t^{i,2}$ by resorting to (5.15) and verifies the fulfillment of the following condition

$$\frac{1}{\beta_t^{i,2}} < (1-\xi_t)\,\frac{R_c+1}{\beta_t^{i0}}\,. \tag{5.20}$$

If (5.20) holds then, accounting also for (5.19), it can be inferred that cooperation is better than direct transmission, i.e., condition (5.17) is satisfied: in this case, $z_t^i = 1$. Otherwise, cooperation is useless: in this case, $z_t^i = 0$. Therefore, in the end of the handshaking among all participants, the AP responds with a CTS frame, which conveys the following information: (i) the cooperation decision z_t^i; (ii) if $z_t^i = 1$, the data rate $\beta_t^{i,2}$ in Phase II given by (5.15); (iii) the resource price λ computed as explained in Section 5.4.

Fig. 5.3 also depicts the data transmission phase. In Phase I, the source proceeds with sending its data frame at rate $\beta_t^{i,1} = \frac{1}{\xi_t}\,\frac{R_c}{R_c+1}\,\beta_t^{i0}$. In Phase II, along with the source, the self-recruited relays cooperatively transmit the data frame at rate $\beta_t^{i,2}$. Then, the AP finishes the procedure by sending back to the source an acknowledgement (ACK) message.

With reference to the protocol reported above, the key observation is that the selection of the set $\mathcal{C}_t^i$ by virtue of (5.18) is done in a distributed way and, moreover, by simply having access to the channel state from the source i to itself, i.e., $h_t^{i\ell}$, the ℓth candidate cooperative node can *autonomously* determine if, by cooperating,

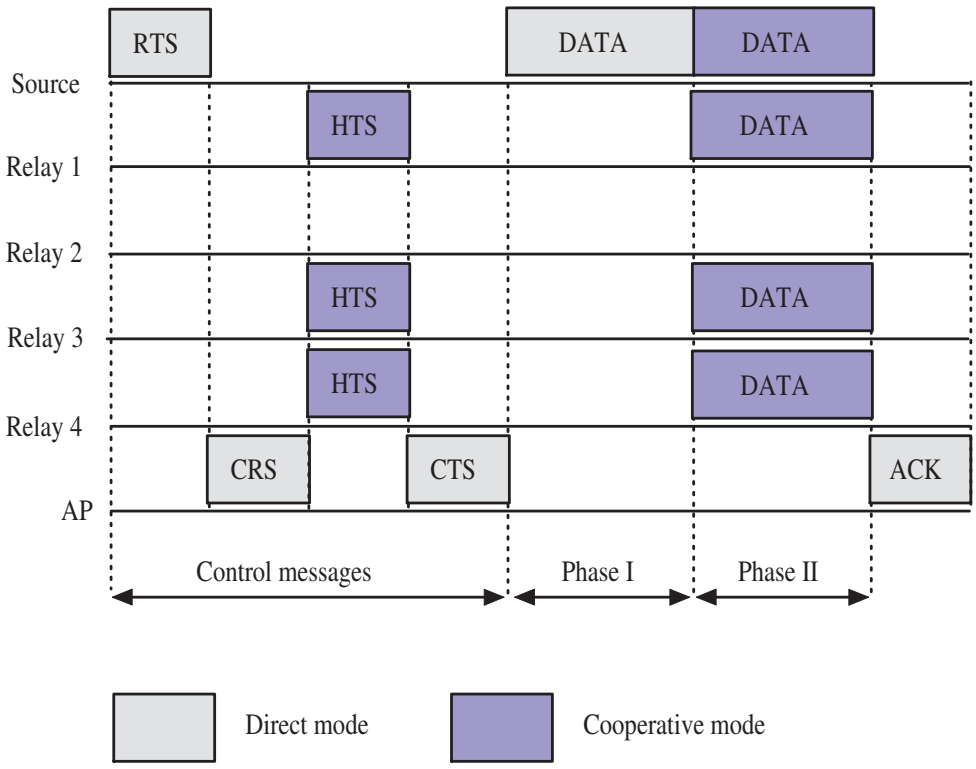


Figure 5.3: Signaling protocol for randomized STBC cooperation.

it can improve the data rate of node i. Another important observation is that the recruitment of the cooperative nodes and the assignment of the data rates requires only four control messages for each source. In particular, the control information exchange is independent of the number of recruited relays thanks to the randomization of the cooperative transmission. Moreover, the two parameters ξ_t and L need to be chosen appropriately. The best choice for ξ_t and L requires global network information. A learning framework would be very appropriate for their selection but we defer the treatment of this aspect to future work. Finally, as for the impact of L on the network performance, it should evidenced that randomized channels tend to behave statistically like their non-randomized counterparts [16], with deep-fade events that

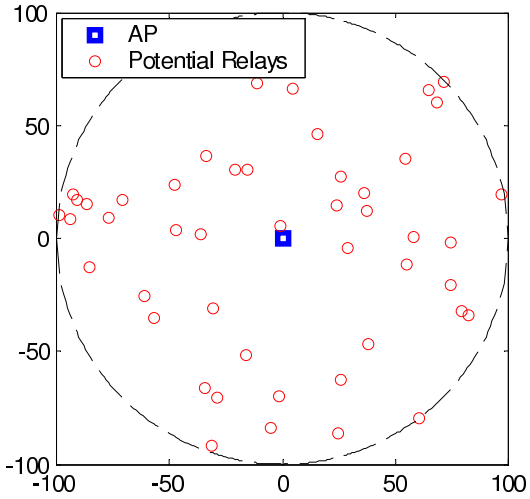


Figure 5.4: Network topology used for numerical results. There are 50 nodes placed randomly and uniformly throughout the AP's 100 m coverage range.

become as frequent as those of L independent channels, as long as the number of cooperative nodes $N_t^i \geq L + 1$.

5.6 Numerical Results

We consider a network with 50 potential relay nodes placed randomly and uniformly throughout the 100 m coverage range of a single AP as illustrated in Fig. 5.4. We specify the placement of the video source(s) separately for each experiment. Let $\eta_t^{i\ell}$ denote the distance in meters between the ith and ℓth nodes. The fading coefficient $h_t^{i\ell}$ over the $i \to \ell$ link is modeled as an i.i.d. $\mathcal{CN}(0, (\eta_t^{i\ell})^{-\delta})$ random variable, where δ is the path-loss exponent. Additionally, we assume that the entries of $\mathbf{R}$, defined in Section 5.5, are i.i.d. $\mathcal{CN}(0, \frac{1}{L})$ random variables, where L is the length of the STBC.

Due to space constraints, and because cooperation has the same impact in both uplink and downlink scenarios, we only present results for cooperative uplink video

transmission. In particular, we consider four uplink scenarios:

1. **Single source**: In this scenario, we assume that a single source node is placed between 10 and 100 m directly to the right of the AP in Fig. 5.4. We use this scenario to evaluate the transmission rates in the direct and cooperative transmission modes at different distances from the AP, and to determine a good self-selection parameter ξ.

2. **Homogeneous video sources**: This scenario mimics a surveillance application in which three cameras capture correlated video content in an outdoor environment and transmit it to the AP. The video sources are placed to the right of the AP as illustrated in Fig. 5.5. To simulate correlated content, we assume that each of the three cameras stream the Foreman sequence (CIF resolution, 30 Hz framerate, encoded at 1.5 Mb/s) offset by several frames. Using homogeneous sources allows us to isolate the impact of cooperation on the video streaming performance by removing the additional layer of complexity introduced by heterogeneous video sources (e.g. different packet priorities and bit-rates among the video users).

3. **Heterogeneous video sources 1**: This scenario mimics a network in which users deploy entertainment applications such as video sharing and video conferencing. To simulate this, we assume that the three video sources illustrated in Fig. 5.5 transmit heterogeneous video content to the AP. Specifically, we assume that video user 1 streams the Coastguard sequence (CIF, 30 Hz, 1.5 Mb/s), video user 2 streams the Mobile sequence (CIF, 30 Hz, 2.0 Mb/s), and

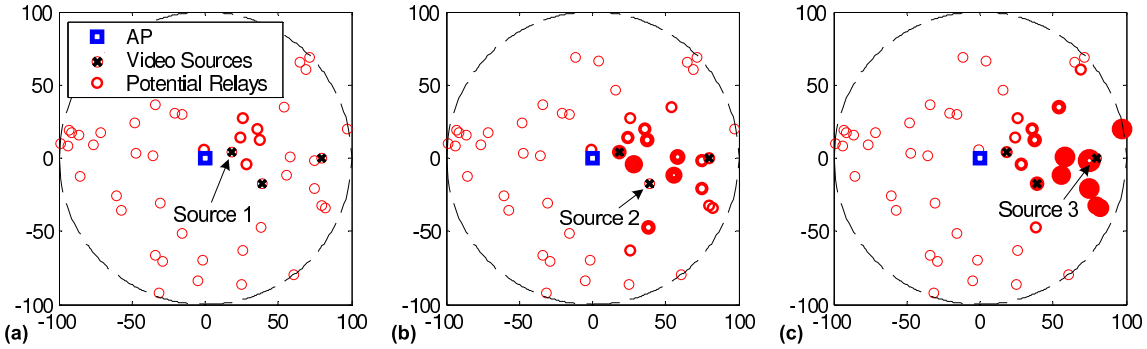


Figure 5.5: Video source placement for homogeneous and heterogeneous streaming scenarios. Three video sources are placed 20 m, 45 m, and 80 m from the AP at angles 25°, −30°, and 0°, respectively. (a,b,c) Relay activation frequencies for video source 1, 2, and 3, respectively. The size of the relay is proportional to the frequency with which it is activated as a helper for the corresponding source.

video user 3 streams the Foreman sequence (CIF, 30 Hz, 1.5 Mb/s).

4. **Heterogeneous video sources 2**: This is the same as the previous scenario, but with video user 2 streaming the Foreman sequence and video user 3 streaming the Mobile sequence.

We note that the proposed framework can be applied using any video coder to compress the video data. However, for illustration, we use a scalable video coding scheme [25], which is attractive for wireless streaming applications because it provides on-the-fly application adaptation to channel conditions, support for a variety of wireless receivers with different resource and power constraints, and easy prioritization of video packets.

In the presented results, we deploy the proposed randomized STBC cooperation protocol outlined in Section 5.5.3 and determine the optimal resource allocation and scheduling decisions using the distributed optimization introduced in Section 5.4.2. The relevant simulation parameters are given in Table 5.1. Note that, in the homoge-

Table 5.1: Simulation parameters.

Parameter	Description	Value
L	Length of the STBC	2
R_c	Rate of orthogonal STBC rule	1
ξ	Self-selection parameter	0.20
P	Packet size	8000 bits
BEP	Bit error probability target	10^{-3}
δ	Path loss exponent	3
R_{cell}	WLAN coverage radius (5 dB SNR at boundary)	100 m
M	Number of nodes (excluding the AP)	50
α	Discount factor	0.80
$1/T_s$	Symbol rate (symbols per second)	625000 or 1250000

neous and heterogeneous scenarios described above, we simulate two levels of network congestion by adjusting the symbol rate. To simulate a network experiencing low congestion, we use the symbol rate $\frac{1}{T_s} = 1250000$ symbols/second; whereas, to simulate high congestion, we use half that symbol rate, i.e., $\frac{1}{T_s} = 625000$ symbols/second.

5.6.1 Cooperation Statistics

In this subsection, we consider the single source scenario described above. Fig. 5.6 illustrates the performance of the proposed cooperation protocol for time-invariant self-selection parameter values $\xi_t = \xi \in \{0.1, 0.2, \ldots, 0.5\}$, given the single source transmitting to the AP. Note that the results hold regardless of the symbol rate because the proposed cooperation protocol only depends on the transmission rate measured in bits/second/Hz.

From Fig. 5.6(a), it is clear that nodes further from the AP utilize cooperation more frequently than nodes closer to the AP. This is because, on average, distant nodes have the feeblest direct signals to the AP due to path-loss and, therefore, have the most to gain from the channel diversity afforded to them by cooperation. It is

also clear from Fig. 5.6(a) that cooperation is utilized more frequently as the self-selection parameter ξ increases. This is because more relays satisfy the self-selection condition in step 3 of the recruitment protocol in Section 5.5.3 for larger values of ξ. However, larger values of ξ yield relay nodes for which $\frac{\beta_t^{i0}}{\beta_t^{i\ell}}$ is large, which leads to a bad transmission rate over the bottleneck hop-1 cooperative link. Due to this poor bottleneck rate, the mean cooperative rate shown in Fig. 5.6(c) declines for $\xi > 0.2$.

We let the self-selection parameter $\xi_t = \xi = 0.2$ because, from Fig. 5.6(d), this value provides a large average transmission rate over the AP's entire coverage range, and achieves approximately double the direct rate when the source is between 60 and 100 m from the AP. Fig. 5.5 illustrates the activation frequencies for different relays in the homogeneous and heterogeneous streaming scenarios with $\xi = 0.2$.

5.6.2 Transmission Rate and Resource Price

Fig. 5.7 illustrates the average transmission rates achieved by the video users in the homogeneous and heterogeneous scenarios under different levels of network congestion. Recall that the resource cost x_t^i incurred by user i is inversely proportional to the transmission rate [see (5.6)], which decreases as the distance to the AP increases due to path loss. Hence, when only direct transmission is available, user 3 tends to resign itself to a low average transmission rate because the cost of using resources is too high. Cooperation increases the average transmission rate, thereby providing user 3 lower cost access to the channel to transmit more data.

In the homogeneous scenario illustrated in Fig. 5.7(a), cooperation tends to equalize the resource allocations to the three users (this is especially evident in the cooper-

ative case with low congestion). This is because the homogeneous users have identical utility functions; thus, when sufficient resources are available, it is optimal for them to all operate at the same point of their resource-utility curves. In contrast, when heterogeneous users with different utility functions are introduced, the transmission rates change to reflect the priorities of the different users' video data. Observing Fig. 5.7(b,c), it is clear that the additional resources afforded by cooperation tend to go to the highest priority video user, who, in our simulations, is the user streaming the Mobile sequence. This is especially true if the highest priority user is farther from the AP.

Recall that users autonomously optimize their resource allocation and scheduling actions given the resource price λ announced by the AP. Table 5.2 illustrates the optimal resource prices in the homogeneous and heterogeneous scenarios. Interestingly, cooperation impacts the resource price differently depending on the level of congestion in the network. In particular, when there is little congestion in the network, cooperation decreases the resource price compared to direct transmission. This is because cooperation increases the available resources without significantly increasing aggregate demand. In contrast, when there is congestion in the network, cooperation increases the resource price compared to direct transmission. This is because, users that resigned themselves to low transmission rates in the direct scenario suddenly demand resources due to a decrease in their resource cost. The resource price increases in our simulations because the enlarged demand pool exceeds the additional resources introduced by cooperation.

Table 5.2: Resource prices in different scenarios.

Streaming Scenario	Transmission Mode	Resource Price (Low / High)
Homogeneous	Direct	45.79 / 42.97
	Cooperative	38.72 / 52.56
	Change	-6.93 / 9.59
Heterogeneous 1	Direct	51.01 / 53.17
	Cooperative	48.02 / 71.94
	Change	-2.99 / 18.77
Heterogeneous 2	Direct	68.24 / 41.48
	Cooperative	62.61 / 72.86
	Change	-5.63 / 31.38

5.6.3 Video Quality Comparison

Table 5.3 compares the video quality obtained in the homogeneous and heterogeneous scenarios, where video quality is measured in terms of peak-signal-to-noise ratio (PSNR in dB) of the luminance channel. In the low congestion scenario, the user furthest from the AP (user 3) benefits on the order of 5-10 dB PSNR from cooperation, while the video user closest to the AP (user 1) is penalized by less than 0.4 dB PSNR. In the high congestion scenario, user 3 goes from transmitting too little data to decode the video (denoted by "$---$") to transmitting enough data to decode at low quality, while penalizing user 1 by less than 0.8 dB PSNR. Note that these PSNR results implicitly reflect the end-to-end delay from the source, through the relays, to the destination. This is because the frames that are not entirely received before their deadlines, and subsequent frames that depend on them, cannot be decoded and therefore do not contribute to the received video quality.

Table 5.3: Average video quality (PSNR) in different scenarios.

Streaming Scenario	Transmission Mode	Video User 1 @ 20 m (Low / High)	Video User 2 @ 45 m (Low / High)	Video User 3 @ 80 m (Low / High)
Homogeneous		Foreman	Foreman	Foreman
	Direct	36.82 dB / 36.51 dB	35.85 dB / 30.20 dB	29.89 dB / --- dB
	Cooperative	36.69 dB / 35.82 dB	36.58 dB / 34.83 dB	36.04 dB / 27.12 dB
	Change	-0.13 dB / -0.69 dB	0.73 dB / 4.63 dB	6.15 dB / --- dB
Heterogeneous 1		Coastguard	Mobile	Foreman
	Direct	32.30 dB / 31.09 dB	26.74 dB / 24.53 dB	25.94 dB / --- dB
	Cooperative	31.94 dB / 30.89 dB	27.14 dB / 25.8 dB	35.69 dB / 27.12 dB
	Change	-0.36 dB / -0.20 dB	0.4 dB / 1.27 dB	9.75 dB / --- dB
Heterogeneous 2		Coastguard	Foreman	Mobile
	Direct	31.91 dB / 31.72 dB	35.16 dB / 32.75 dB	21.85 dB / --- dB
	Cooperative	31.56 dB / 30.97 dB	35.72 dB / 32.39 dB	26.53 dB / 22.03 dB
	Change	0.35 dB / -0.75 dB	0.56 dB / -0.36 dB	4.68 dB / --- dB

5.7 Conclusion

We introduced a cooperative multiple access strategy that enables nodes with high priority video data to be serviced while simultaneously exploiting the diversity of channel fading states in the network using a randomized STBC cooperation protocol. We formulated the dynamic multi-user video transmission problem with cooperation as an MU-MDP and we used Lagrangian relaxation with a uniform resource price to decompose the MU-MDP into local MDPs at each user. We analytically proved that opportunistic (myopic) cooperation strategies are optimal, and therefore the users' local MDPs only need to determine their optimal resource allocation and scheduling policies based on their experienced cooperative transmission rates. Subsequently, we proposed a randomized STBC cooperation protocol that enables nodes to opportunistically and distributively self-select themselves as cooperative relays. Finally, we experimentally showed that the prosed cooperation strategy significantly improves the video quality of nodes with feeble direct links to the AP, without significantly

penalizing other users. We also demonstrated that cooperation increases the resource price in congested networks and decreases the price in uncongested ones.

Appendix F: Proof of Theorem 5.1

The transmission rate β^i is a function of the cooperation decision z^i and the channel state $\mathbf{H}$, i.e., we can write $\beta^i = \beta^i(\mathbf{H}, z^i)$. Thus, the cooperation decision impacts the immediate utility because it constrains the set of feasible scheduling actions $\mathcal{P}^i(\mathcal{T}^i, \beta^i)$ through the packet constraint $\|\mathbf{y}^i\|_1 \leq \frac{R\beta^i}{PT_s}$.

Let $z^{i*}_{\text{opp}} = \arg\max_{z^i}\{\beta^i(\mathbf{H}, z^i)\}$ and $\beta^{i*}_{\text{opp}} = \max_{z^i}\{\beta^i(\mathbf{H}, z^i)\}$ denote the optimal opportunistic cooperation decision and the maximum transmission rate, respectively. Selecting the cooperation decision that maximizes the immediate transmission rate enlarges the set of feasible scheduling actions, i.e., $\mathcal{P}^i(\mathcal{T}^i, \beta^i) \subseteq \mathcal{P}^i\left(\mathcal{T}^i, \beta^{i*}_{\text{opp}}\right)$, for all $\beta^i \leq \beta^{i*}_{\text{opp}}$. We now show that the optimal opportunistic cooperation decision enables a user to maximize its immediate utility when $\alpha = 0$. We then show that the long-term utility with $\alpha > 0$ is maximized by the same optimal opportunistic cooperative decision. Let $u^i_\lambda(\mathcal{T}^i, \beta^i, \mathbf{y}^i) = \sum_{j\in\mathcal{F}^i} q^i_j y^i_j - \lambda\left(x^i - \frac{1}{M}\right)$ denote the utility less the cost, where x^i is given by (5.6). Under the optimal opportunistic cooperation decision, we have

$$\max_{\mathbf{y}^i\in\mathcal{P}^i\left(\mathcal{T}^i,\beta^{i*}_{\text{opp}}\right)} u^i_\lambda\left(\mathcal{T}^i, \beta^i, \mathbf{y}^i\right) \geq \max_{\mathbf{y}^i\in\mathcal{P}^i(\mathcal{T}^i,\beta^i)} u^i_\lambda\left(\mathcal{T}^i, \beta^i, \mathbf{y}^i\right), \tag{5.21}$$

where the inequality is due to the fact that $\mathcal{P}^i(\mathcal{T}^i, \beta^i) \subseteq \mathcal{P}^i\left(\mathcal{T}^i, \beta^{i*}_{\text{opp}}\right)$. Thus, the

optimal opportunistic cooperative decision allows the node to immediately transmit more packets for the same resource cost. By the same argument, the long-term utility with $\alpha > 0$ is also maximized, i.e.,

$$\begin{aligned} U_{\lambda}^{i,*}\left(s^{i}\right) &= \max_{\mathbf{y}^{i}\in\mathcal{P}^{i}\left(\mathcal{T}^{i},\beta_{\text{opp}}^{i*}\right)} \left\{ u_{\lambda}^{i}\left(\mathcal{T}^{i},\beta^{i},\mathbf{y}^{i}\right) + \alpha \sum_{s^{i\prime}} p\left(s^{i\prime}|s^{i},\mathbf{y}^{i}\right) U_{\lambda}^{i,*}\left(s^{i\prime}\right) \right\} \quad (5.22) \\ &\geq \max_{\mathbf{y}^{i}\in\mathcal{P}^{i}\left(\mathcal{T}^{i},\beta^{i}\right)} \left\{ u_{\lambda}^{i}\left(\mathcal{T}^{i},\beta^{i},\mathbf{y}^{i}\right) + \alpha \sum_{s^{i\prime}} p\left(s^{i\prime}|s^{i},\mathbf{y}^{i}\right) \bar{U}_{\lambda}^{i,*}\left(s^{i\prime}\right) \right\} \quad (5.23) \\ &= \bar{U}_{\lambda}^{i}\left(s^{i}\right) , \quad (5.24) \end{aligned}$$

where the inequality is due to the fact that $\mathcal{P}^{i}\left(\mathcal{T}^{i},\beta^{i}\right) \subseteq \mathcal{P}^{i}\left(\mathcal{T}^{i},\beta_{\text{opp}}^{i*}\right)$ for all $\beta^{i} \leq \beta_{\text{opp}}^{i*}$. Thus, the optimal opportunistic cooperative decision also maximizes the long-term utility.

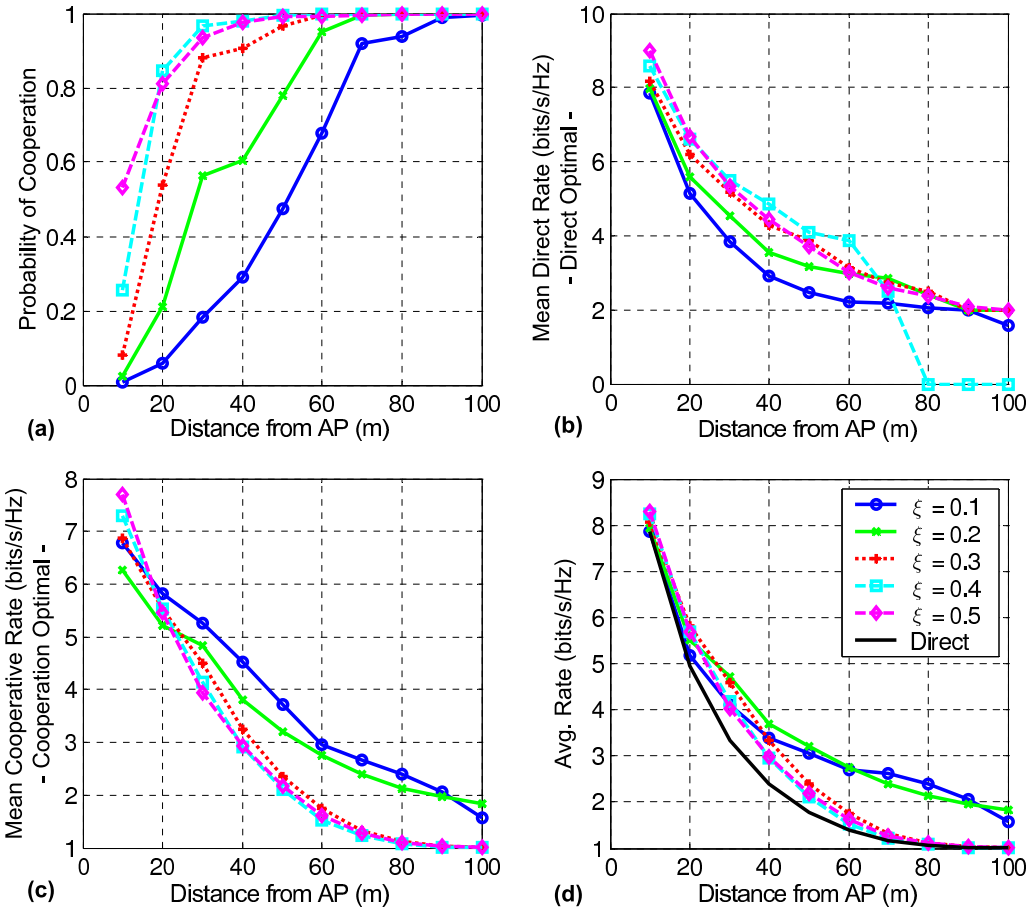


Figure 5.6: Cooperative transmission statistics for different values of the self-selection parameter ξ and for different distances to the AP. (a) Probability of cooperation being optimal. (b) Direct transmission rate conditioned on direct transmission being optimal. (c) Cooperative transmission rate conditioned on cooperative transmission being optimal. (d) Average transmission rate, which depends on the probability of cooperation, the conditional direct rate, and the conditional cooperative rate.

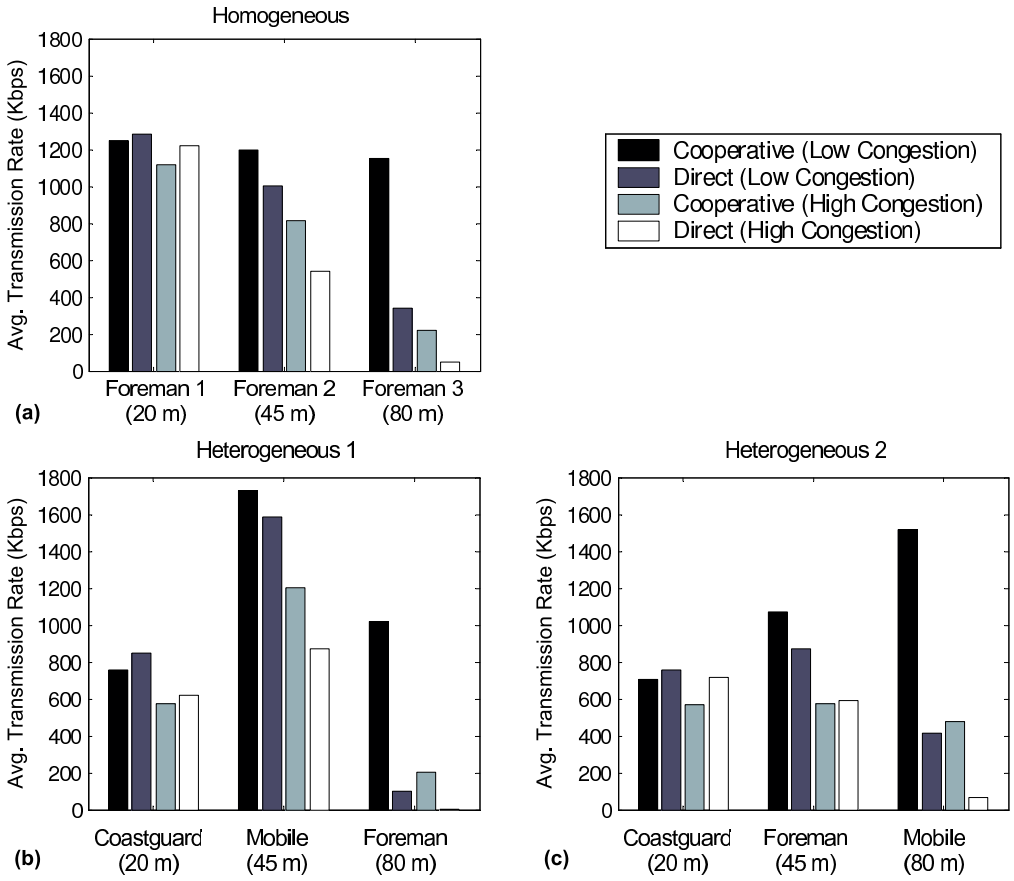


Figure 5.7: Average transmission rates using direct and cooperative transmission in low congestion and high congestion scenarios. (a) Homogeneous video sources. (b,c) Heterogeneous video sources.

References

[1] M. Chiang, S. H. Low, A. R. Caldbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of IEEE*, vol. 95, no. 1, 2007.

[2] R. Knopp and P. A. Humblet, "Information capacity and power control in single-cell multiuser communications," *Proc. IEEE ICC*, 1995.

[3] P. Viswanath, D. N. C. Tse, R. Laroia, "Opportunistic beamforming using dumb antennas," *IEEE Trans. on Information Theory*, vol. 48, no. 6, June 2002.

[4] D. N. C. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[5] T. C.-Y. Ng and W. Yu, "Joint optimization of relay strategies and resource allocations in cooperative cellular networks," *IEEE Trans. on Selected Areas in Communications*, vol. 25, no. 2, Feb. 2007.

[6] X. Zhang and Q. Du, "Cross-Layer Modeling for QoS-Driven Multimedia Multicast/Broadcast Over Fading Channels in Mobile Wireless Networks," *IEEE Communications Magazine*, pp. 62–70, August 2007.

[7] J. Huang, Z. Li, M. Chiang, and A.K. Katsaggelos, "Joint Source Adaptation and Resource Allocation for Multi-User Wireless Video Streaming," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 18, issue 5, 582-595, May 2008.

[8] E. Maani, P. Pahalawatta, R. Berry, T.N. Pappas, and A.K. Katsaggelos, "Resource Allocation for Downlink Multiuser Video Transmission over Wireless Lossy Networks," *IEEE Transactions on Image Processing*, vol. 17, issue 9, 1663-1671, September 2008.

[9] G-M. Su, Z. Han, M. Wu, and K.J.R. Liu, "Joint Uplink and Downlink Optimization for Real-Time Multiuser Video Streaming Over WLANs," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 2, pp. 280-294, August 2007.

[10] F. Fu and M. van der Schaar, "A Systematic Framework for Dynamically Optimizing Multi-User Video Transmission," *IEEE J. Sel. Areas Commun.*, vol. 28, pp. 308-320, Apr. 2010.

[11] O. Alay, P. Liu, Z. Guo, L. Wang, Y. Wang, E. Erkip, and S. Panwar, "Cooperative layered video multicast using randomized distributed space time codes", *IEEE INFOCOM Workshops 2009*, Rio de Janeiro, Brazil, Oct. 2009, pp. 1–6.

[12] J.N. Laneman and G.W. Wornell, "Distributed space-time block coded protocols for exploiting cooperative diversity in wireless networks," *IEEE Trans. Inf. Theory*, vol. 49, pp. 2415–2425, Oct. 2003.

[13] A. Sendonaris, E. Erkip, and B. Aazhang, "User cooperation diversity – Part I & II," *IEEE Trans. Commun.*, vol. 51, pp. 1927–1948, Nov. 2003.

[14] J.N. Laneman, D. Tse, and G.W. Wornell, "Cooperative diversity in wireless networks: efficient protocols and outage behavior," *IEEE Trans. Inf. Theory*, vol. 50, pp. 3062–3080, Sept. 2004.

[15] N. Mastronarde, M. van der Schaar, A. Scaglione, F. Verde, and D. Darsena, "Sailing good radio waves and transmitting important bits: a case for cooperation at the physical layer in wireless video transmission," in *Proc. IEEE International Conf. Acoustics, Speech and Signal Proc.*, Dallas, Texas, USA, Mar. 2010, pp. 5566–5569.

[16] B. Sirkeci-Mergen and A. Scaglione, "Randomized space-time coding for distributed cooperative communication", *IEEE Trans. Signal Process.*, vol. 55, pp. 5003–5017, Oct. 2007.

[17] P. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media", *IEEE Trans. Multimedia*, vol. 8, pp. 390–404, Apr. 2006.

[18] D. P. Bertsekas, "Dynamic programming and optimal control," 3rd, Athena Scientific, Massachusetts, 2005.

[19] N. Salodkar, A. Karandikar, V. S. Borkar, "A stable online algorithm for energy-efficient multiuser scheduling," *IEEE Trans. on Mobile Computing*, vol. 9, no. 10, Oct. 2010.

[20] J.G. Proakis, *Digital Communications*. New York: McGraw-Hill, 2001.

[21] V. Tarokh, H. Jafarkhani, and A. Calderbank, "Space-time block codes from orthogonal designs," *IEEE Trans. Inf. Theory*, vol. 45, pp. 1456-1467, July 1999.

[22] F. Verde, T. Korakis, E. Erkip, and A. Scaglione, "A simple recruitment scheme of multiple nodes for cooperative MAC," *IEEE Trans. on Commun.*, vol. 58, pp. 2667-2682, Sept. 2010.

[23] P. Liu, C. Nie, T. Korakis, E. Erkip, S. Panwar, F. Verde, and A. Scaglione, "STiCMAC: A MAC Protocol for Robust Space-Time Coding in Cooperative Wireless LANs," *IEEE Transactions on Wireless Communications*, in revision.

[24] H. Wang and N. Mandayam, "A Simple Packet Transmission Scheme for Wireless Data over Fading Channels," *IEEE Transcations on Communications*, vol. 52, no. 7, pp. 1055–1059, 2004.

[25] J.R. Ohm, "Three-dimensional subband coding with motion compensation", *IEEE Trans. Image Processing*, vol. 3, no. 5, Sept 1994.

Chapter 6

Conclusion

In this dissertation, we developed a rigorous, formal, and unified energy-efficient resource management framework for supporting heterogeneous multimedia applications in a large class of resource management scenarios including energy-efficient point-to-point wireless communication, multi-user cooperative wireless communication, and energy-efficient cross-layer system optimization. First, we showed that a dynamic centralized cross-layer optimization can be decomposed into multiple single-layer dynamic optimizations, thereby preserving the layered system architecture. Second, we showed that, by exploiting this decomposition and introducing appropriate interlayer communication, the individual layers can learn online, at run-time, how they impact and are impacted by the layers they interact with. This ultimately enables the layers to autonomously, but jointly, optimize the system's performance. Third, we introduced a post-decision state to separate the known an unknown dynamics. Using the post-decision state: (i) the decision making process can be separated from the unknown dynamics; (ii) partial information about the system can be exploited so that less information needs to be learned; and, (iii) action-exploration can be avoided. Forth, we introduced the concept of virtual experience, which can be used to learn about multiple states in each time slot to dramatically accelerate learning. Finally, we showed that some actions, such as the

decision to cooperate in a multi-user cooperative network, can be made myopically without loss of optimality.

The proposed framework can be extended on three fronts. First, it can be applied in new resource management scenarios. For instance, the proposed framework is a promising solution to the intimately related problems of load balancing, cache efficiency, and power management in multi-core and parallel systems, which are used for multimedia compression and processing in data centers, desktop computers, and increasingly in mobile devices such as laptops, tablets, and smart phones. Additionally, the proposed framework may prove useful for solving large-scale emerging problems like intelligent energy distribution in the smart grid. Second, the proposed algorithms can be implemented in real networks and systems to compare them to existing resource management algorithms and to gain a practical understanding of implementation issues and resource management overheads. Third, there is potential to advance reinforcement learning theory by gaining a better theoretical understanding of the dynamic behavior of the proposed learning algorithms (i.e. bounds on the convergence rates). This is in contrast to most existing theoretical results, which focus on the asymptotic behavior of the algorithms, and ignore the speed at which they converge. However, convergence speed is essential to ensure good performance in engineered systems, and in particular for energy-efficient resource management of delay-sensitive applications. Meanwhile, existing theoretical results on convergence rates are for very general settings and can likely be significantly tightened for resource allocation problems with specific structures and learning algorithms that exploit these structures. New theoretical results could be

used to explore design trade-offs in different resource management scenarios, and could be used as benchmarks for real system implementations.