A Reinforcement-Learning Approach to Power Management

Carl Steinbach

AI Technical Report 2002-007 May 2002

A Reinforcement-Learning Approach to Power Management

by

Carl W. Steinbach

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Certified by: Leslie Pack Kaelbling
Professor
Thesis Supervisor

Accepted by: Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Reinforcement-Learning Approach to Power Management

by

Carl W. Steinbach

Submitted to the Department of Electrical Engineering and Computer Science on May 28, 2002, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

We describe an adaptive, mid-level approach to the wireless device power management problem. Our approach is based on *reinforcement learning*, a machine learning framework for autonomous agents. We describe how our framework can be applied to the power management problem in both infrastructure and ad hoc wireless networks. From this thesis we conclude that mid-level power management policies can outperform low-level policies and are more convenient to implement than high-level policies. We also conclude that power management policies need to adapt to the user and network, and that a mid-level power management framework based on reinforcement learning fulfills these requirements.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Acknowledgments

I would first like to thank my advisor, Leslie Kaelbling, for introducing me to reinforcement learning and for supporting my efforts to apply this technique to power management. Her patience was much appreciated. I am also extremely indebted to Terran Lane for meeting with me countless times over the past year to offer advice and support. I would like to thank Kyle Jamieson for showing interest in my project and for the valuable discussions we had about power management in ad hoc networks.

This work was supported in part by NASA award #NCC2-1237.

Contents

1 Introduction 8

1.1 The Wireless Problem 8

1.2 Current Solutions 9

1.3 Our Approach: Adaptive Mid–level Power Management 10

1.4 Contributions 11

1.4.1 Thesis Outline 11

2 Wireless Mobile Networks 12

2.1 Overview 12

2.2 Hardware 13

2.2.1 The Radio 14

2.3 Infrastructure vs. Ad Hoc Networks 15

2.4 Protocols 16

2.5 The IEEE 802.11 Standard 16

2.5.1 Ad Hoc and Infrastructure Modes 17

2.5.2 Beacon Mechanism 17

2.5.3 Power Management 17

3 Reinforcement Learning 19

3.1 The Reinforcement Learning Model 19

3.1.1 Markov Decision Processes 20

3.1.2 The Markov Property 20

3.2 Value Functions 20

3.3 Measures of Reward 21

3.4 Q–learning 22

4 A Reinforcement-Learning Approach to Power Management 24

4.1 The MDP Approach to Power Management 24

4.2 A Simple RL Framework 26

4.2.1 The Problem 26
4.2.2 The Agent's Environment 26
4.2.3 The Reward Signal 27
4.2.4 The Learning Algorithm 28
4.2.5 Assumptions 29
4.2.6 Results 32
4.3 Extensions and Improvements to the Simple Framework 33
4.3.1 Extensions to the State and Action Spaces . . . 33
4.3.2 Multiple Window Sizes 33
4.3.3 Better Learning Algorithms 33
4.4 Specific Applications of the RL Power Management Model 34
4.4.1 Power Management in Infrastructure Networks . 34
4.4.2 Power Management in Ad Hoc Networks 35

5 Conclusions **37**
5.1 Future Work 37

List of Figures

2.1 A schematic overview of a Lucent wireless network interface designed to fit in a mobile computer's PCMCIA slot. 13
2.2 WNIC state transition diagram 14

4.1 Agent actions and their effect on the `mode` state variable. 27
4.2 An arbitration mechanism for integrating actions from multiple agents. 29

List of Tables

2.1 PRISM chipset power consumption by mode. 15

4.1 Q values obtained using the simple model. 32

Chapter 1

Introduction

In the past decade, trends in hardware miniaturization have resulted in the introduction and widespread acceptance of laptops and, more recently, in a burgeoning market for hand-held computing devices. While these devices match their desktop counterparts in computational power, a major drawback is their reliance on battery power. Whereas processor speeds have improved at an exponential rate for the past three decades, battery capacity has only increased by a factor of two to four over the same period. With no reason to expect major advances in battery technology in the near future, current attempts to solve this problem have focused on designing portable devices that consume less energy.

1.1 The Wireless Problem

Most people involved in the hand-held computer industry agree that the major application of these devices will be as mobile email, text messaging, and web browsing platforms. Unfortunately, according to Stemm and Katz [1] the network interface is typically the single largest consumer of power in a hand-held system. Interest in designing low power components for hand-held devices has led to the introduction in the past decade of *dynamic voltage scaling* (DVS) processors, as well as the popularization of new fabrication techniques which result in more efficient chip components. However, attempts to improve the energy efficiency of wireless network interfaces is constrained by the poor power consumption characteristics of radio transmitters and receivers. There is no obvious way of making the radio more power efficient since there is a direct relationship between transmit power and the range of the transmitted signal. Consequently, attempts to save power at the network interface have focused on power management (PM) techniques for switching the radio off during idle periods.

1.2 Current Solutions

To date most research in this area has focused on doing PM either at the hardware-level (low-level) or at the application level (high–level). Hardware-level approaches typically build power management into the *medium access control* (MAC) layer [2] of the network protocol. For example, the IEEE 802.11 Wireless LAN Standard [3] stipulates MAC layer support for power management in both infrastructure and ad hoc modes. This support is provided through *traffic indication map* (TIM) fields contained in broadcast messages sent by the base station, and a polling mechanism that allows mobile hosts to request packets buffered by the base station while they were asleep.

The major drawback of power management schemes that operate at the MAC layer is that they can only view the data contained within frames as opaque objects. For example, the network interface is unable to tell whether a frame it receives contains data corresponding to a UDP packet or a TCP packet, nor can it differentiate between frames which contain different types of application-layer packets such as Secure Shell Protocol (SSH) or Hypertext Transfer Protocol (HTTP) packets.

Proponents of application level power management strategies are currently a minority, but their arguments are important to consider. Kravets and Krishnan [4] describe a transport-level protocol for managing the sleep/active cycle of a mobile host's wireless interface that exposes power management to applications. Application programs that access the network are required to convey their communication needs to the transport-layer power management policy, which in turn decides whether it is advisable to transition the network interface to the sleep or awake mode.

An argument in favor of this strategy is that application programs necessarily have the most information on which to base predictions about their communication needs. For example, a web browser that has just requested a page from a server but that has not yet received a response can direct the power manager to stay awake. Similarly, applications that receive streaming data over the wireless network can tell the power manager to keep the interface in active mode until the application terminates or the stream is closed.

However, it seems unlikely that the software industry will embrace a PM technique that places more work on the application programmer, making it even harder to port applications from desktop to mobile platforms. Furthermore, it remains to be seen whether adding this functionality is as trivial an operation as its proponents assert.

1.3 Our Approach: Adaptive Mid–level Power Management

This thesis proposes a new approach to power management in wireless network interfaces based on an adaptive power manager with access to application-layer and transport-layer information. It operates at a lower level than the technique described by Kravets and Krishnan [4], since it does not require application programs to pass it information, and it operates at a higher level than the MAC layer methods described earlier. This means that it has access to transport-layer and application-layer information, which it can use to make more informed predictions about near term communication needs.

The power manager is based on a machine-learning framework for autonomous agents called *reinforcement learning* (RL) [5]. Reinforcement learning models an agent's sequential decision-making task as a Markov Decision Process (MDP) [6] with unknown parameters. The goal of an agent is to learn a *policy* that maximizes the agent's performance at the task. The policy that the agent learns is a function mapping states to actions, and an agent's performance is quantified by a reward function. In this case the reward function is inversely related to the amount of energy consumed by the network interface.

We believe that an adapative mid-level approach to power management will be able to outperform PMs built into the MAC layer. In most cases, the packets that are transmited and received over a mobile host's wireless interface are the direct result of the user's actions, or are sent in response to the user's actions. We believe we can build a more complete model of this cause-and-effect behavior by placing the agent closer to its source. An accurate model of these high-level processes and their effect on network activity will allow us to make better predictions about when the network interface's services will be needed. We also note that our approach is supported by the widely accepted end-to-end argument [7], which holds that system functions such as power management, implemented at a low level, are likely to be of little value when compared to those same functions implemented at a higher level.

In comparing our mid-level PM scheme to the high-level approaches of Kravets and Krishnan [4], Ellis [8], and Flinn and Satyanarayanan

[9], we note that our method does not place any requirements on the applications. Furthermore, since our method is adaptive it can learn to support the vast array of software that is already in use on mobile computers.

1.4 Contributions

The main contribution of this thesis is its illustration of an adaptive mid–level approach to wireless device power management that avoids the drawbacks associated with hardware level and application level solutions to the problem.

1.4.1 Thesis Outline

Chapter 2 of this thesis describes wireless networks in general with particular attention paid to hardware power consumption issues. Chapter 2 also discusses the 802.11 wireless LAN standard. Chapter 3 gives an overview of reinforcement learning and a detailed description of Q-learning, and then discusses the mid–level power management strategy in detail. Finally, chapter 4 concludes with a discussion of related and future work.

Chapter 2

Wireless Mobile Networks

2.1 Overview

A *wireless mobile network* is a local area network (LAN) where the individual *stations* are mobile computers that communicate with the rest of the network via radio or infrared signals. The stations are most likely laptop or hand-held computers outfitted with a wireless radio network interface. Current wireless LAN standards specify data transfer rates equivalent to those of Ethernet, placing wireless LANs in the same league as their wired counterparts.

While wireless LANs provide a software interface virtually indistinguishable from that provided by wired LANs, there are many differences between the two technologies at the link and MAC layers. The following sections discuss the characteristics that make wireless LANs different from their wired counterparts.

Before we begin, it is important to point out that 802.11 is a wireless LAN standard. While it is currently the dominant wireless standard it does not define wireless LANs, and other standards which would accomplish the same task in different ways are possible. Consequently, the first part of this chapters focuses on the characteristics of wireless LANs in general, and in the later part we consider the 802.11 standard in particular.

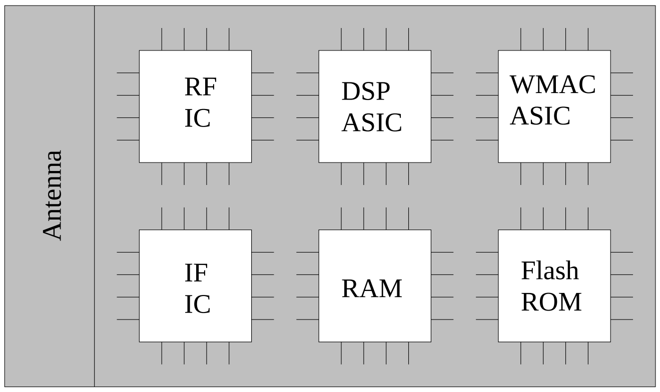

Figure 2.1: A schematic overview of a Lucent wireless network interface designed to fit in a mobile computer's PCMCIA slot.

2.2 Hardware

The basic building block of a wireless LAN is a wireless network interface card. These devices typically come in ISA or PCMCIA form factors.

Figure 2.1 is a block diagram of a Lucent 802.11 wireless network interface designed to fit in a mobile computer's PCMCIA slot. Wireless network interfaces built for other standards would likely look the same. The components shown in the diagram include a radio (RF IC), intermediate frequency chip (IF IC), a digital signal processor (DSP) application-specific integrated circuit (ASIC), a wireless medium access controller (WMAC) ASIC, and RAM and ROM chips.

The modular nature of the device provides manufacturers with two distinct advantages. First, a manufacturer of wireless network interfaces (WNICs) typically purchases the components from third party manufacturers. Many of these components are not specific to any wireless network protocol in particular, so manufacturers can market a general purpose chip with multiple applications. Second, the modular construction of the device makes it possible for the network interface to selectively power down on a component basis. For example, the interface's logic might want to turn the radio off but maintain the supply of power to clocks.

Figure 2.2 shows the canonical WNIC state diagram. In the `transmit` and `receive` states the radio is fully powered and the WNIC is actively

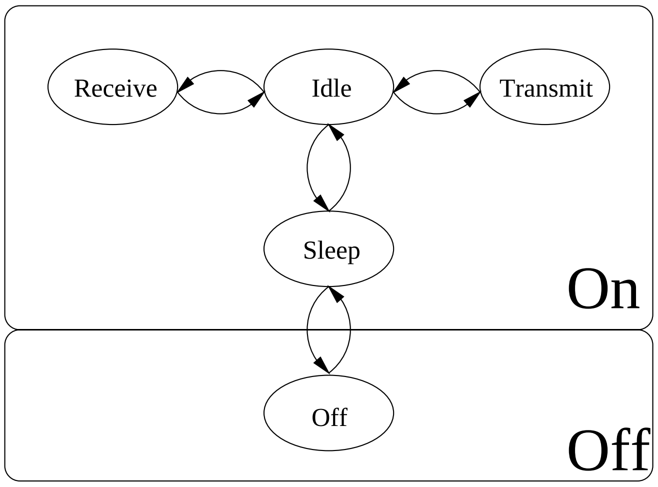

Figure 2.2: A state machine showing the wireless network interface states and the transitions between them. In 802.11 devices the `receive`, `transmit`, `idle` and `sleep` states are not individually visible to the operating system, which views these states collectively as the `on` state.

transmitting and receiving data. In the `idle` state the radio is powered, and the device is listening to the communications medium, but the device does not pass any of the data it receives while in this mode up to the software layer. In the `sleep` state the radio is powered down while other subsystems, such as hardware clocks, remain powered. Finally, in the `off` state the device is completely powered down.

Most of the states illustrated in figure 2.2 are hidden from the software that controls the WNIC. For example, in 802.11 WNICs the `receive`, `transmit`, `idle` and `sleep` states are only visible to the operating system as the `on` state. Another point to note is that the `sleep` state typically has several sub-states. Each substate corresponds to a different power-saving mode. We discuss these modes in detail in the next section.

2.2.1 The Radio

The WNIC's radio deserves special consideration as it is both central to the device's purpose and also its single largest consumer of energy. Several efforts to measure the power consumption of popular WNICs [1, 10]

Mode	Recovery Time	Current
TX Current (continuous)	N/A	488mA
RX Current (continuous	N/A	287mA
Average Current w/o PSM	N/A	290mA
Average Current w/ PSM	N/A	50mA
Power Saving Mode 1	1μs	190mA
Power Saving Mode 2	25μs	70mA
Power Saving Mode 3	2ms	60mA
Power Saving Mode 4	5ms	30mA

Table 2.1: PRISM chipset power consumption by mode.

have resulted in the conclusion that the WNIC, and the radio in particular, is responsible for a significant portion of the energy requirements of portable computers. Consequently, the goal of WNIC power management policies is to keep the WNIC's radio in a low power mode (or off) most of the time without adversely affecting the performance of the WNIC.

To illustrate the different power saving modes available at a device level we now consider the PRISM radio chipset manufactured by the Intersil Corporation. The PRISM chipset has been incorporated into a large number of commodity 802.11 devices, and is just as applicable to other WNIC designs since it is not protocol specific. The chipset includes a radio, modem, baseband processor, dual synthesizer, and MAC module. Andren et al. [11] describe the different device-level power saving modes supported by the chipset, which are summarized in table 2.1. Of particular interest are the tradeoffs between power consumption and recovery time. Deeper sleep modes consume less energy but require more time to transition back to an active state. This is due to the fact that the deeper sleep modes allow capacitors to discharge, and considerable time is required to re-charge these analog components. Similarly, deep sleep modes also shut down oscillators used in the radio, and when power is reapplied, time is required for them to settle.

2.3 Infrastructure vs. Ad Hoc Networks

Wireless LANs can be divided into two separate categories: *infrastructure* networks and *ad hoc* networks. *Infrastructure* networks consist of some number of wireless stations and a wired base station. The base station (or *access point* in 802.11 terminology) is responsible for mediating communication between individual stations in the wireless LAN as well as providing an up-link service over the wire to the rest of the network. In many designs the base station also provides power management support to the mobile stations by offering to temporarily buffer packets bound for sleeping stations.

Ad hoc networks are also composed of individual stations, except that there is no base station. Instead, stations talk directly to each other without the aid of a base station. Ad hoc networks are a convenient alternative to infrastructure networks in environments where it is impossible to install fixed base stations.

Infrastructure and ad hoc wireless networks face very different power management issues, which we discuss in detail in chapter 4.

2.4 Protocols

One of the major differences between wired and wireless networks is that collision detection in wired networks is easy to implement, while reliable collision detection (CD) in wireless networks is impossible to implement. This is a consequence of the fact that situations where stations A and C are both in range of station B but out of range of each other are common. Hence, if A and C begin transmitting at the same time B will be unable to read the data due to a collision which neither A nor C can detect. Because of this physical limitation, wireless protocols are based on collision avoidance (CA) rather than collision detection.

One of the first wireless protocols to use collision avoidance was the Multiple Access with Collision Avoidance (MACA) protocol. The 802.11 standard is a direct descendant of MACA. Collision avoidance was implemented by requiring each station to observe a random discrete-time backoff interval from the time the medium becomes clear until the station begins transmitting. This, along with the use of an RTS/CTS (request to send/clear to send) mechanism between the transmitting station and receiving station eliminates collisions which would otherwise cause problems.

2.5 The IEEE 802.11 Standard

The 802.11 (WaveLAN)[3] standard was developed by the IEEE 802 LAN/MAN Standards Committee for wireless local area networks (LAN). The standard describes operation in both *ad hoc* and *infrastructure* networks with data rates ranging from 1 to 11 Mbits/s. It is currently the dominant wireless LAN standard.

The 802.11 standard shares many similarities with the 802.3 Ethernet standard including media access control (MAC) of the data link layer and the physical layer (PHY) provided through a 48 bit address, and the use of the carrier sense multiple access (CSMA) protocol for data transmission at the physical layer.

2.5.1 Ad Hoc and Infrastructure Modes

As mentioned earlier, the 802.11 standard specifies operation in *ad hoc* and *infrastructure* modes. In *infrastructure* mode 802.11 stations communicate with a fixed 802.11 *access point* (AP). All data is routed through the AP, including packets sent between wireless stations. In many cases the AP provides a gateway between the wireless network and other 802.x networks (Ethernet, Token Ring, etc). In *ad hoc* mode there is no centralized authority. Rather, the task of organizing and running the network is shared by each of the stations.

2.5.2 Beacon Mechanism

The *beacon* mechanism is central to the operation of 802.11 networks in both infrastructure and ad hoc modes. It provides for the periodic communication of system parameters that are required in order for the network to function. Each *beacon frame* includes a timestamp field and a beacon interval field which can be used by 802.11 stations to compute when the next beacon frame will arrive.

2.5.3 Power Management

The 802.11 standard defines two different power states: *awake state* and *doze state*. In the awake state the device is fully powered, whereas in the doze state the device is not able to transmit or receive data and energy consumption is minimized. WNICs in the *doze state* are allowed to transition into the `sleep` state illustrated figure 2.2, whereas WNICs in the *awake state* must remain in the `idle`, `transmit`, or `receive` states. Wireless devices can conserve energy by switching from *active state* to *power-save mode*. In power-save mode the device spends most of its time in the doze state, and transitions periodically to the active mode in order to check for packets.

Power Management in Infrastructure Networks

In infrastructure networks, the access point is responsible for buffering packets that are addressed to dozing stations. A field with an entry for each station is included in the beacon frames transmitted by the AP. If the AP has buffered data for a station, the corresponding entry in this field is set accordingly. Stations in power-save mode are required to transition from the doze state to the awake state at each beacon interval. If the beacon indicates that the AP has buffered data for a station, the station sends a special frame to the AP which causes the AP to transmit the buffered data. Otherwise, the station transitions back to the doze state pending the next beacon frame.

Power Management in Ad Hoc Mode

In ad hoc networks there is no AP that can buffer data destined for a dozing station. Rather, the stations themselves are responsible for buffering packets that are addressed to dozing stations. While the 802.11 standard does specify how stations can announce that they have buffered data destined for other stations, it does not describe how stations should determine the power-saving mode of other stations, and hence does not explain how a station would decide to buffer data. In later chapters we discuss some of the algorithms that have been developed to solve this problem.

Chapter 3

Reinforcement Learning

Reinforcement Learning (RL) [5, 12] is a machine-learning framework for autonomous agents. Agents are trained to accomplish a task through the application of rewards and punishments. The major difference between reinforcement learning and other machine learning techniques is that in reinforcement learning there is no need to specify how the task is to be accomplished. Instead, the agent learns how perform the task through experience.

3.1 The Reinforcement Learning Model

In this thesis we limit our discussion to techniques for solving the reinforcement learning problem using statistical and dynamic programming techniques to estimate the value of state-action pairs in the world.

The standard reinforcement-learning model consists of an agent that interacts with the environment through observations and actions. On each step of interaction with the environment, the agent receives an input from the environment that describes the current state of the environment. This forms the agent's observation of the environment and is used by the agent to determine its internal state. After observing the environment, the agent issues an action as output, which changes the state of the environment, and receives a scalar-valued *reward* from the environment, which indicates the value of the state transition. Formally, a reinforcement learning model consists of

- a set $\mathcal{S}$ of environment states,
- a set $\mathcal{A}$ of agent actions, and

- a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$.

The goal of the agent is to learn a policy π, mapping states to actions, such that some measure of the long-term reward collected by the agent through interactions with the environment is maximized.

3.1.1 Markov Decision Processes

Reinforcement learning systems model the world as *Markov Decision Processes* (MDPs) [6]. Formally, an MDP consists of

- a set $\mathcal{S}$ of states,
- a set $\mathcal{A}$ of actions,
- a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$, and
- a state transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, where each element of $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$.

$T(s, a, s')$ is the probability of ending up in state s' when taking action a in state s. Similarly, $R(s, a)$ is the reward for taking action a in state s.

3.1.2 The Markov Property

A model expressed as an MDP is said to satisfy the *Markov property* and to be *Markov* if the state transition function is independent of the agent's history of environment states and actions given the current state. This is equivalent to saying that the agent has a perfect ability to observe the state of its environment. If a model satisfies the Markov property then it is possible to determine the next state and expected next reward based only on the current state and action. MDP models which have hidden states are called *partially observable Markov decision processes* (POMDPs) [6].

Most of the algorithms used for solving reinforcement learning problems require the existence of an MDP that models the problem, satisfies the Markov property, and is fully observable.

3.2 Value Functions

A *value function* is a function of states, or state-action pairs, that estimates the utility of an agent being in a particular state, or the utility of an agent taking a specific action out of a particular state.

Here "utility" is defined in terms of the future rewards that the agent can expect to receive. Earlier we explained that a policy π is a mapping from each state-action pair $\langle s, a \rangle \in \mathcal{S} \times \mathcal{A}$ to the probability $\pi(s, a)$ of choosing action a when in state s. We now define $V^{\pi}(s)$, the *state-value function for policy* π to be the expected return when starting in state s and following policy π thereafter. We can express $V^{\pi}(s)$ formally as

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}.$$

$E_{\pi}\{\}$ is used to signify the expected return given that the agent follows policy π.

Similarly, we can define $Q^{\pi}(s, a)$, the *action-value function for policy* π, to be the expected return from starting in state s, taking action a, and thereafter following policy π. We can express Q^{π} formally as

$$Q^{\pi}(s, a) = E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} r_{t+k+1} | s_t = s, a_t = a\right\}.$$

We use Q^* and V^* respectively to symbolize the state-value and action-value functions for the optimal policy π^*.

3.3 Measures of Reward

We previously stated that the goal of an agent is to maximize over some period of time a measure of the reward that it collects. Here we describe two frequently used reward models and explain their differences.

First we consider the *receding-horizon* reward model that, on every step causes an agent to think it only has h steps left, and to choose the action that will result in the largest expected reward over the next h steps. Formally, we say that on every step the the agent is trying to maximize the expression

$$E(\sum_{t=0}^{h} r_t).$$

In the receding-horizon model an agent is not concerned about the rewards it will collect that are more than h time steps in the future. Rather, on each step the agent will select what it thinks is the best action given that it has h steps remaining in which to act and gain rewards. The receding-horizon model is most appropriate in cases where the agent should try to maximize its short term expected reward.

In contrast, the *infinite-horizon* reward model causes the agent to optimize its long-term reward. However, rewards in the future are geometrically discounted by a constant factor γ:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t).$$

This causes the agent to emphasize rewards in the near future over those in the far future, but does not require selection of a hard horizon.

3.4 Q–learning

Q–learning [13] is one of several algorithms available for solving reinforcement learning problems. It is a *model–free* algorithm, meaning that, in contrast to *model–based* algorithms, there is no need to know the transition probability function $T(s, a, s')$ and reward function $R(s, a)$ in order to learn a policy. Q–learning derives its name from the *optimal action-value function* $Q^*(s, a)$. The Q-learning algorithm is guaranteed to converge to optimal Q^* values for each state action pair in $\mathcal{S} \times \mathcal{A}$.

The Q–learning algorithm is based on the observation that since $V^*(s) = \max_a Q^*(s, a)$, one can express the optimal action–value function recursively as

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') \max_{a'} Q^*(s', a').$$

This leads to the Q–learning rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right].$$

The Q-learning algorithm repeatedly applies the Q-learning rule to the set of Q values corresponding to the states in the environment. Starting with a state s, the algorithm chooses an action a using the policy derived from Q, takes action a and observes the next state s' and the reward r. The value of $Q(s, a)$ is then updated using the Q-learning rule, s is set to s', and the process repeats.

Many states will never be visited if the algorithm always selects an action a corresponding to the maximum Q value for a given state. Clearly, without visiting all of the actions in $\mathcal{A}$ the algorithm can not guarantee an optimal result. In order to fix this problem we have the algorithm use an ϵ-greedy policy for selecting actions. Given a small ϵ

such that $0 < \epsilon < 1$, the algorithm selects an action corresponding to the maximum Q values with a probabily of $1 - \epsilon$, and selects a random action with probability ϵ.

The ϵ value determines the balance the algorithm makes between exploration and exploitation. Low ϵ values cause the algorithm to neglect exploration and select the greedy action most of the time. This will cause the algorithm to take a longer time to converge to optimal Q values. Similarly, large ϵ values cause the algorithm to spend all of its time exploring. In this case the policy converges very rapidly to optimal Q values, but the agent fails to follow its own policy and performs poorly.

Chapter 4

A Reinforcement-Learning Approach to Power Management

This chapter describes our framework for casting the wireless device power management problem as a reinforcement learning task. In section 1 we discuss an earlier MDP-based approach to power management that inspired our work in this area, and explain why we think an RL approach is superior. In section 2 we introduce a simple RL framework for considering the power management problem and discuss our results. In section 3 we discuss a variety of proposed extensions to the simple model discussed in section 2. Most of these extensions involve extending the action and state spaces to better reflect the actual dynamics of the WNIC. Finally, in section 4 we discuss the various ways that these models could be applied to the WNIC power-management problem in a realistic setting.

4.1 The MDP Approach to Power Management

Our interest in a reinforcement-learning approach to wireless device power management was directly inspired by the work of Šimunić, Benini, Qui et al. [14, 15, 16, 17, 18] on the related topic of using stochastic processes, and Markov decision processes in particular, to solve the power management problem.

People using this approach generally model each component with a separate probability distribution. For example, user behavior is modeled with a request inter-arrival distribution, the behavior of the device is modeled with a service time distribution, and the time taken by the device to transition between its power states is modeled with a transition distribution. Typically these distributions are estimated during actual user interaction with the system, but in some cases artificial distributions such as the Pareto distribution are substituted. A linear-programming algorithm is then used to find an optimal policy based on the distributions and an MDP modeling the dynamics of the system. Šimunić [18] reports that these methods can be used to realize a factor of 5 power savings in WNIC operation if the user is willing to accept a small performance penalty.

While these methods can be used to realize impressive power savings in a laboratory setting, their wider applicability is questionable when you stop to consider the assumptions made by these models. Specifically, these power management models guarantee optimal results only when the actual usage patterns match those of the distributions that were used to compute the policy. It is extremely unlikely that a policy that came with your laptop, designed for the "average" user, will provide a useful approximation of your usage patterns. Consequently, we are left with the option of either devising a system that adapts the policy to the user, or a system that recomputes the policy using probability distributions obtained by monitoring the user.

Chung, Benini, et al. [16] devised a non-stationary, MDP-based power management system that uses sliding windows. Each window is composed of a set number of slots, and records the recent packet arrival and departure history for the WNIC. This data is used to form a variable that ranges between 0 and 1 and which reflects the average packet arrival rate over the period of time monitored by the window. This variable is then used to select a policy from a policy table. However, while the overall policy that results is non-stationary, it is not really adaptive in a general sense. To see why, consider that the policies in the policy table were all computed based on sample distributions which need to match the actual user patterns in order to yield good results. Furthermore, even if the distributions used to compute the policies do match the patterns of the user, the system will still not guarantee optimal results, since it uses a heuristic algorithm to select policies.

The other MDP-based power-management algorithms provide no mechanism for fine or even medium-grained adaptability. Rather, to construct a useful policy, a power manager would need to collect data on a user's usage patterns over the period of a day, a week, or a month, and then use the resulting distributions to recompute the policy. Note that the distributions collected one day will generally not match distributions collected on the next day. Hence, finding an optimal or near-optimal policy with these methods is very unlikely.

It is precisely these considerations that have motivated us to pursue a reinforcement-learning approach to the power-management problem. Reinforcement-learning algorithms have the great advantage of continuously adapting to the usage patterns exhibited by the user and network.

4.2 A Simple RL Framework

This section gives a detailed description of our initial attempts at formulating an adaptive reinforcement-learning framework for solving the wireless-device power-management problem. Our approach is based on a simple model of the system dynamics of the *wireless network interface card* (WNIC). We present initial results obtained using this model and investigate the model's shortcomings. In the next section, we propose strategies for constructing more realistic models. We begin with a discussion of how the agent views and interacts with the WNIC.

4.2.1 The Problem

Our goal is to construct an agent capable of power-managing the WNIC so that some long-term measure of the WNIC's energy consumption is minimized without creating a degradation in network performance that is unacceptable to the user. The agent is supposed to accomplish this task by switching the WNIC either `on` or `off`. When the WNIC is `on` it can receive and transmit packets and communicate with the AP, but it also consumes power. When the WNIC is `off` it cannot transmit or receive traffic and it consumes no power, but users requesting network service will experience latency or service interruption.

4.2.2 The Agent's Environment

The agent's ability to perceive the world is defined by the state space $\mathcal{S}$, the elements of which are ordered tuples of the form $\langle \texttt{mode}, \texttt{time} \rangle$. The `mode` state variable represents the current state of the WNIC and is an element from the set $\{\texttt{on}, \texttt{off}\}$. In our simple model the `transmit`, `receive`, `idle`, and `sleep` states are all lumped into the `on` state.

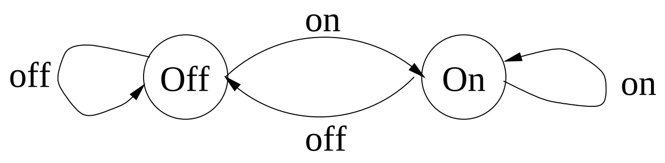

Figure 4.1: Agent actions and their effect on the `mode` state variable.

Our model divides time into discrete *time intervals*, each of which is a second long. Ideally, the `time` state variable would represent the number of time intervals that have elapsed since the last packet was transmitted or received by the WNIC. However, since the WNIC can neither transmit nor receive packets while in the `off` state, this definition becomes impractical. Consequently, we make the assumption that when the WNIC is in the `off` state, packets that would have been received or transmitted had the WNIC been in the `on` state are time-stamped with the current time, and either buffered on the AP or on the WNIC itself. We then use the following rule for setting the `time variable`. As long as the WNIC is in the `off` state, the `time` variable increments one unit per time interval. If the WNIC transitions from the `off` state to the `on` state, the WNIC checks for time-stamped packets buffered in its queue or on the AP. If any packets are found, the `time` variable is set equal to the difference of the current time and the most recent time found on the collection of buffered packets. If no packets are outstanding, then the `time` variable is simply incremented by a unit.

The agent's action space, $\mathcal{A}$, is $\{\texttt{on}, \texttt{off}\}$: on each step of interaction with the WNIC, the agent can either turn the WNIC `on` or `off`. If the card is already in the `on` state, the `on` action has no effect on the `mode`. Similarly, issuing the `off` action when the WNIC is in the `off` state has no effect on the `mode`. Figure 4.1 summarizes these transition rules.

4.2.3 The Reward Signal

On each step of interaction with the WNIC, the agent receives a reward signal. Over the long term, the agent tries to maximize some measure of the incremental reward signals. Ideally, the WNIC's hardware interface would provide a means for measuring the device's power consumption. However, since it does not, we base our estimates of power consumption on data collected by Stemm and Katz [1] and Feeney and Nilsson [10].

If the reward signal reflected only the amount of energy consumed by the WNIC, and if the agent wanted to minimize this quantity, the simple solution would be to stay in the `off` state all the time. In order to avoid this trap, we also include a *latency* parameter in the function used to calculate the reward signal. Consequently, the reward signal takes into account the amount of energy consumed in the last time step, as well as the time that packets spent buffered in the WNIC's output queue if the card was in the `off` state. The introduction of a scalar variable α with values in the range $[0, 1]$ allows us to express the reward function as

$$r = -(\alpha * \mathit{energy}) - ((1 - \alpha) * \mathit{latency}).$$

The variable α expresses the tradeoff between the importance of conserving energy and avoiding latency, and is set by the user. High α values make the reward function insensitive to latency penalties, and will result in an agent that spends most of its time asleep. Similarly, small α values will cause the agent to spend most of its time awake in an effort to avoid packet latency penalties.

4.2.4 The Learning Algorithm

Our initial attempts at solving the WNIC power-management problem with reinforcement learning have focused on offline, trace-based training using the Q-learning algorithm. We used the `tcpdump` [19] packet-sniffing utility to collect our live traces, and have supplemented this collection with a variety of artificially generated traces in an effort to make program verification easier. Each entry in a raw trace corresponds to a packet, and indicates whether the packet was transmitted or received, at what time it was handled by the WNIC, the application-level protocol the packet was associated with, and the size of the packet. Currently we ignore packet fragmentation, instead creating only one entry for each fragmented packet. This means that our packet size records are sometimes incorrect. e plan to fix this error in the near future.

Each raw trace is filtered by protocol in order to create separate traces composed of SSH packets, HTTP packets, etc. The separate traces are then used to train individual agents with the hope that they will learn policies tailored to the unique patterns of the protocol. We plan to eventually integrate the individual agents using an arbitration mechanism similar to the one illustrated in figure 4.2. At each time step the protocol-specific agents illustrated in the figure—π_{SSH}, π_{HTTP}, etc.—submit an action to the arbitration mechanism. The arbiter will turn the WNIC `off` if all of the agents submit `off` actions. However,

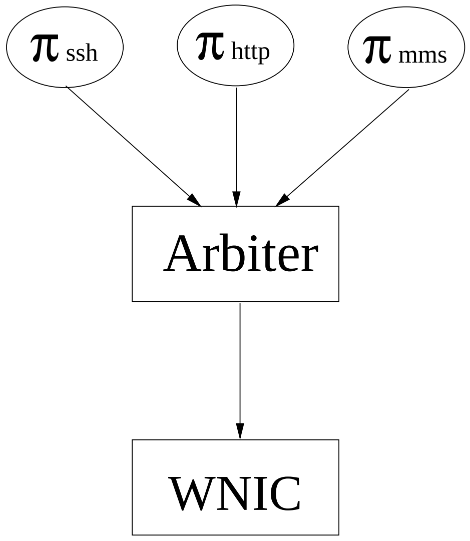

Figure 4.2: An arbitration mechanism for integrating actions from multiple, protocol specific power management agents. The three agents represented as π_{ssh}, π_{http}, and π_{mms} were respectively trained to manage Secure Shell Protocol, Hypertext Transfer Protocol, and the Microsoft Media Server Protocol.

the arbiter will turn the WNIC `on` if any of the agents submits an `on` action.

We based our decision to learn a separate policy for each protocol on several observations. First, we realized that we would have to add multiple state variables to the simple model in order to get the same effect, and that this change would make the state space much larger and cause the agent to take more time to learn an optimal policy. We also realized that our separate policy approach makes it easier to learn new policies for new protocols.

4.2.5 Assumptions

The model we have presented above makes three fundamental assumptions that are questionable in this domain. We list each of these assumptions below along with an example that illustrates why our model violates the assumption. All of the assumptions are based on conditions that must be true in order for the Q-learning algorithm to guarantee convergence to optimal results. It is important to note that these conditions are sufficient but not necessary. Hence, while our model does not satisfy any of these conditions completely, it is not right to conclude that this will make it impossible to obtain useful, or even optimal results using these techniques.

Assumption 1: The environment is Markov

The Q-learning algorithm is guaranteed to eventually converge to optimal Q values only if the environment satisfies the Markov property. Hence, we either need to demonstrate that the environment *is* Markov, or that it is close enough to being Markov that its non-Markovianity will not noticeably affect the results of the Q-learning algorithm.

As explained earlier, the environment is Markov if the probability of being in state s' and receiving reward r at time $t+1$ is dependent only on the state the environment was in at time t and the action the agent took at time t. This is expressed formally as

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}.$$

Unfortunately, we know that this condition does not hold for the system model described above. To see this, consider what happens when the WNIC is in state $\langle \texttt{on}, t \rangle$ and selects action `off`. If the WNIC transmitted or received a packet in the period between t and $t+1$ then the next state will be $\langle \texttt{on}, 1 \rangle$. However, if the WNIC did not transmit or receive a packet between t and $t+1$, then the next state will be $\langle \texttt{on}, t+1 \rangle$. Our model does not include a state variable to describe packet transmission, so from the agent's perspective this state transition will appear to be nondeterministic.

In the reinforcement learning framework, each action of the agent is answered by the environment with a reward signal and a state signal. The degree to which the state signal accurately and completely reflects the current state of the environment determines an environment's *observability*. An environment is *fully observable* if the state signal uniquely distinguishes the current state from all other states.

In our model, the state signal consists of two state variables: `mode` and `time`. The `mode` is either `on` or `off`. In the previous chapter we explained that the WNIC at a minimum has 5 states (`transmit`, `receive`, `idle`, `sleep`, and `off`), and in many cases has even more if we include the different radio power-saving modes. Earlier we mentioned that the `transmit`, `receive`, `idle`, and `sleep` states are all lumped into the `on` state. We can reduce the set of unobservable states to `transmit`, `receive`, and `idle` by disabling the WNIC's power saving mode. At this point, our main concern is that our inability to differentiate between the `transmit`, `receive`, and `idle` states will compromise our ability to accurately gauge the amount of power consumed per time unit. However, we argue that we can closely estimate the amount of time the WNIC spends in `transmit` and `receive` modes by observing the size of the packets that are transmitted and received.

The state signal also fails to account for the state of the network as a whole. For example, whether or not the WNIC receives a packet at time $t + 1$ is dependent on whether another WNIC sent a packet to it at time t. Packets in transit as well as the state of other WNICs on the network are not included in the state signal since acquiring such information is impossible. We believe that we can ignore this hidden state, since mobile computers rarely receive packets that are not in some way related to an earlier request made by the mobile computer.

Assumption 3: Offline Learning is Legal

Our initial tests have been based on an offline, trace-based approach to learning. In an experimental setting, this is advantageous since it allows us to quickly reproduce results as well as to subject our algorithm to traces modeling a wide range of different user behaviors. However, there is one major pitfall involved in learning from traces, illustrated with the following example. If, during an offline training epoch, the Q-learning algorithm turns the simulated WNIC off at time t and then on again at time $t + n$, what happened to the packets that the trace indicates were transmitted and received by the real WNIC during that period? In our offline learning setup, we chose to assume that these packets were buffered either at the AP or within the WNIC, and that these buffers were cleared and the packets sent during the next time slice when the WNIC was in the `on` state.

The problem with this approach is that the behavior documented by the trace after time $t + n$ was affected by what the trace recorded happening before time $t + n$. Consequently, when we let our offline learning algorithm ignore what actually happened between time t and $t + n$, we end up invalidating the rest of the trace. Furthermore, it seems likely that the errors introduced by each one of these disconnects between the trace record and simulator are cumulative, such that after a handful of these episodes the trace is probably useless as a document describing what is going to happen next.

We defend our actions by noting two points. First, using traces offline is a better starting point than trying to learn online from real

State	On		Off	
Action	Off	On	Off	On
1	-1.211	-1.795	-0.756	-1.119
2	-1.216	-1.819	-0.567	-1.189
3	-1.326	-1.952	-0.630	-1.250
4	-1.394	-2.019	-0.699	-1.320
5	-1.471	-2.097	-0.777	-1.402
6	-1.562	-2.181	-0.864	-1.486
7	-1.656	-2.279	-0.961	-1.582
8	-1.765	-2.381	-1.069	-1.682
9	-1.949	-1.875	-1.250	-1.188
10	-1.315	-1.787	-1.634	-1.391
11	-1.213	-1.797	-1.570	-1.483
12	-1.259	-2.134	-1.440	-2.764

Table 4.1: Q values obtained using the simple model.

data. Second, we believe that since the agent's decision period is quite large (100 msec to 1sec) in comparison to the amount of time it takes to transmit a TCP packet (0.5 to 1msec at 11Mbps) our argument that buffered packets can be cleared within the first `on` interval is probably accurate.

4.2.6 Results

In an effort to verify our simple framework we have tested the model described above and the Q-learning algorithm using artificially generated trace data. Our artificial trace, composed of 10 records, describes a situation in which a packet arrives every 10 seconds. Using $\gamma = 0.9$ and $\epsilon = 0.1$, and reward factor $\alpha = 0.7$ we have demonstrated reliable convergence to an optimal policy within 1000 training epochs. The optimal policy in this case, described by the Q values in table 4.1, is for the WNIC to enter the `on` state once every 10 seconds in order to handle the packet arrival, and then spend the next 9 seconds in the `off` state. With this policy the WNIC minimizes power consumption and avoids latency penalties.

4.3 Extensions and Improvements to the Simple Framework

In this section we discuss several important extensions to the model described above that attempt to make it a more realistic representation of the WNIC's dynamics. We believe that by applying these modifications in the future we will be able to realize improved power management performance from the system.

4.3.1 Extensions to the State and Action Spaces

In our discussion of wireless networking hardware in chapter 2 we explained that many of the internal WNIC power states are hidden from the software interface. For example, based on the 802.11 WNIC hardware interface, software is only able to differentiate between the `on` and `off` WNIC states, despite the fact that the device actually has at least five major modes (`transmit`, `receive`, `idle`, `sleep`, `off`) not including the various reduced-power radio modes. We believe that it is important to investigate the consequences of including these hidden states in our model's state space, and then determining through simulation if power management software agents could learn improved policies for managing the transitions between these states.

4.3.2 Multiple Window Sizes

The state space of the simple model described above makes no attempt to record or take into account the recent packet transmission and receipt history of the WNIC. Since the current service load is a good predictor of future near-term service load we expect to be able to improve the performance of our power management policies through the introduction of history state variables similar to those described by Chung et al [16]. For example, we plan to add state variables that record the average packet arrival/departure rate over the past 100 milliseconds, 1 second, and 10 seconds.

4.3.3 Better Learning Algorithms

We have based our initial experiments on Watkins' Q-learning algorithm since its properties are well understood and the algorithm itself is easy to implement. However, the Q-learning algorithm has several drawbacks including a sensitivity to models that have hidden states.

Consequently, we are interested in experimenting with other reinforcement learning algorithms. A natural first step would be to experiment with the Sarsa algorithm [5], an on-policy temporal-difference learning algorithm that is less sensitive to partially observable models than the Q-learning algorithm. Furthermore, modifying our learning algorithm to use the Sarsa algorithm instead of the Q-learning algorithm is simple since the Sarsa algorithm differs from the Q-learning algorithm by only one line of code. We might also improve our results by testing the Sarsa(λ) algorithm. Finally, if all of these algorithms prove to be too sensitive to the hidden states in our state model we will try experimenting with POMDP methods.

4.4 Specific Applications of the RL Power Management Model

In the preceding sections we have described an abstract model that, without modification, is not directly applicable to the problem of WNIC power management. Here we discuss how one would adapt this abstract framework to the task of power management in infrastructure and ad hoc networks.

4.4.1 Power Management in Infrastructure Networks

A WNIC operating in an infrastructure LAN can enter low-power modes and turn its radio off as long as it first notifies the AP. Subsequently, if the AP receives a packet bound for the sleeping WNIC it will buffer the packet and alert the WNIC in the next beacon frame. The 802.11 standard defines the beacon period as a variable controlled by the AP. Despite this fact, the vast majority of access points rely on a 100 millisecond beacon period. This allows WNICs that are based on the PRISM chipset described in chapter 2 to enter the lowest power-saving mode during the inter-beacon periods and still have time to awaken and receive the next beacon frame. It is important to recognize that awakening from a low power state requires a significant amount of energy, since capacitors have to be charged and oscillators stabilized. Consequently, it is advisable to awaken and listen for a beacon frame only if the WNIC knows with high probability that there are packets addressed to it that are buffered on the AP.

Krashinsky [20] notes this problem and proposes an extension to the hardware-level 802.11 power management algorithm that would allow WNICs to change their ListenInterval (the period between listening for a beacon frame) dynamically. He refers to this mechanism as "ListenInterval-backoff" as a way of comparing it to the familiar exponential-backoff contention algorithms that are used at the physical layer in the Ethernet and 802.11 standards. In Krashinsky's heuristic algorithm, the WNIC starts with a ListenInterval equal to the beacon period, but gradually increases this variable in beacon period increments as long as the WNIC does not detect packets buffered at the WNIC.

We believe that we can improve on Krashinsky's heuristic approach by using a reinforcement learning agent to control changes in the ListenInterval variable. Whereas Krashinsky's algorithm is implemented at the hardware level and can see only physical layer frame information, our mid-level RL algorithm would have access to transport-layer and application-layer packet data, which would allow it to select a ListenInterval suited to the needs of the applications running on the WNIC's host.

4.4.2 Power Management in Ad Hoc Networks

In chapter 2 we explained that ad hoc wireless LANs do not have a permanent, wired access point to coordinate traffic and buffer packets for sleeping WNICs. Instead, in ad hoc LANs the duties of the AP are shared by the stations participating in the network. The most common model is for stations to volunteer or be elected to the temporary role of *coordinator*. A station operating as the coordinator functions as the AP for the ad hoc network. It buffers traffic for sleeping nodes and routes traffic to coordinators in other subnetworks. While the 802.11 standard provides MAC layer support for ad hoc networks, it does not define a policy for how to choose coordinators.

How to choose coordinators has been a popular topic in the networking community for the past several years. Most people believe that the primary application of ad hoc networks will be for battlefield communication or as the communication layer in mobile sensor networks. Consequently, people have focused their efforts on designing power-aware algorithms. Span [21] described by Chen et al., and the Power Aware Multi-Access protocol with Signaling for Ad Hoc Networks (PAMAS) [22], described by Singh and Raghavendra, are two of the better known protocols for organizing ad hoc networks in a power-aware manner. Readers interested in an overview of power-aware ad hoc networks should consult Jones et al. [23].

The Span protocol is built on top of the 802.11 standard. In a Span network, stations volunteer to become coordinators using an algorithm that takes into account the station's battery state. Specifically, at regular intervals the current coordinator clears a block of time during which other stations volunteer for coordinator duty. This period is divided into slots. Each station measures its current battery level and uses this reading to determine during which slot it will volunteer for coordinator duty. For example, a station whose batteries are fully charged will volunteer during an earlier slot, whereas a station whose batteries are nearly empty will volunteer during one of the later slots. Each station listens to the channel as its slot approaches. The first station to volunteer typically has the most fully charged batteries, and is consequently elected as the next coordinator. Clearly this algorithm is designed to prolong the life of the network as a whole, as it effectively prevents stations with relatively little battery power from serving as coordinators.

We believe it is possible to improve on the Span algorithm using our RL-based power-management policies. Our motivating example is the case of a station which is receiving or transmitting streaming data. Since such a station is going to stay awake anyway, it makes sense that the station should volunteer to be selected as the coordinator in the network. We believe our mid-level approach stands a better chance than Span of being able to predict near and long-term traffic patterns that affect the selection of coordinators.

Chapter 5

Conclusions

In this thesis we have presented an adaptive mid-level framework for wireless device power management based on reinforcement learning. We have shown some initial results that demonstrate the potential of this technique. We have described several ways of extending our framework and we have also illustrated specific ways in which this framework can be applied to the task of power management in infrastructure and ad hoc wireless networks.

Key Conclusions

- We conclude that a mid-level approach to power management is superior to low-level techniques implemented at the MAC layer, and that a mid-level approach is more convenient for programmers than adding power management directly to applications.
- We conclude that a power manager needs to adapt to the patterns of the user and the network in order to guarentee good performance and demonstrated that techniques based on reinforcement learning are capable of adapting on a continuous basis.
- Finally, we conclude that adaptive, reinforcement learning based power management techniques have applications in ad hoc networks as well as in infrastructure networks.

5.1 Future Work

In chapter 5 we gave an extensive description of possible future directions for this work. We briefly summarize these points below:

- Extend the state space to better reflect the actual internal states of the WNIC.
- Experiment with different reinforcement learning algorithms such as Sarsa as Sarsa(λ) with goal of finding an algorithm that is insensitive to the WNIC's hidden states.
- Include a *history* state variable in the model so that the policy can predict the future based on the past.
- Use the *ns* network simulator to estimate the performance of our techniques in ad hoc and infrastructure networks.

Bibliography

[1] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications, vol.E80-B, no.8, p. 1125-31*, E80-B(8):1125–31, 1997.

[2] A. Tanenbaum. *Computer Networks, Third Edition*. Prentice Hall, 1996.

[3] IEEE Computer Society LAN MAN Standards Committee. *IEEE Std 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[4] R. Kravets and P. Krishnan. Application–driven power management for mobile communication. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 263–277, October 2000.

[5] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[6] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[7] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[8] C. Ellis. The case for higher level power management. In *Proceedings of the Seventh Workshop on Hot Topic in Operating Systems HotOS'1999*, Mar 1999.

[9] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[10] Laura Marie Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *IEEE INFOCOM*, 2001.

[11] C. Andren, T. Bozych, B. Rood, and D. Schultz. PRISM power management modes. Technical report, Intersil Americas Inc., 1997.

[12] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[13] C. Watkins and P. Dayan. Q–learning. *Machine Learning*, 8(3):279–292, 1992.

[14] G. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Proceedings of the 35th Design Automation Conference DAC'98*, 1998.

[15] Qinru Qiu and Massoud Pedram. Dynamic power management based on continuous-time markov decision processes. In *Design Automation Conference*, pages 555–561, 1999.

[16] L. Benini Eui-Young Chung and G. De Micheli. Dynamic power management for non-stationary service requests. In *DATE, Proceedings of the Design, Automation and Test in Europe Conference*, pages 77–81, March 1999.

[17] T. Šimunić, L. Benini, and G. D. Micheli. Event-driven power management of portable systems. In *International Symposium on System Synthesis*, pages 18–23, 1999.

[18] T. Šimunić, H. Vikalo, P. Glynn, and G. De Micheli. Energy efficient design of portable wireless systems, 2000.

[19] V. Jacobson, C. Leres, and S. McCanne. *The tcpdump Manual Page.* Lawrence Berkeley Laboratory.

[20] Ronny Krashinsky. Maintaining performance while saving energy on wireless lans.

[21] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks Journal*, 8(5), sept 2002.

[22] S. Singh and C. Raghavendra. Pamas: Power aware multi-access protocol with signalling for ad hoc networks, 1999.

[23] C. Jones, K. Sivalingam, P. Agrawal, and J. Chen. A survey of energy efficient network protocols for wireless networks, 2001.