DYNAMIC POWER MANAGEMENT BY REINFORCEMENT LEARNING

A THESIS WORK

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND THE COMMITTEE FOR UNDERGRADUATE STUDIES

OF
BRAC UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

BACHELOR OF SCIENCE (B.sc) in ELECTRICAL& ELECTRONICS ENGINEERING (EEE)

BY

SAFAYET HOSSAIN – 12121111

MUHAMMAD ADNAN IBN-ISMAIL - 11321059

SUPERVISED BY

Dr. Md. Muhidul Islam Khan
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
BRAC UNIVERSITY, DHAKA.

DECLARATION

We hereby declare that research work titled “Dynamic power management by reinforcement learning” is our own work. The work has not been presented elsewhere for assessment. Where material has been used from other sources it has been properly acknowledged /referred.

Signature of the Supervisor

……..

Dr. Md. Muhidul Islam Khan

Signatureof Authors

……

Safayet Hossain

12121111

….……

Muhammad Adnan Ibn-Ismail

11321059

ACKNOWLEDGEMENTS

Firstly, We would like to thank Dr. Md. Muhidul Islam Khan, Assistant Professor, Dept. of Computer Science & Engineering (CSE), BRAC University; for his supportive guidance and feedbacks for completion of the thesis. He was very cooperative and dedicated for this thesis.

ABBREVIATION

DPM	Dynamic Power Management
DVFS	Dynamic Voltage Frequency Scaling
MDP	Markov Decision Process Model
RL	Reinforcement Learning
SMDP	Semi Markov Decision Process
HAS	Hardware Assisted Sleep
HABS	Hardware Assisted Buffer Sleep
LAN	Local Area Network
TISMDP	Time Indexed Semi Markov Decision Process
USB	Universal Serial Bus
MLE	Maximum Likelihood Estimation
IPC	Instruction per Cycle
VL	Voltage Level
FL	Frequency Level

ABSTRACT

Optimization in design and utilization of both hardware and software is needed in order to achieve more energy efficient systems. In this paper we presented a Reinforcement learning based DPM approaches for our LAN card power management system. The presented approaches do not require priori model of the system as an Opposite to the existing DPM approaches. Thesis outcomes also show that sleeping is indeed feasible in the LAN and in some cases, with very little impact on other protocols. Moreover, reinforcement learning is a machine intelligence approach that has been applied in many different areas whereas Q-learning is one of the most popular algorithms that perform reinforcement learning. At last, with the desired outcomes of this thesis work, power management issues of LAN card system were solved effectively. In future we aim to compare DPM problem with mission learning problem. The RL based learning algorithm can then be implemented to find the right value of power constraint.

DYNAMIC POWER MANAGEMENT BY REINFORCEMET QLEARNING

Table of Contents

Page#

3. CHAPTER03

4. CHAPTER04

1. CHAPTER05

CHAPTER 01

INTRODUCTION:

Energy efficient systems are widely recognized and vast amount of funding are being approved by the electrical industries to develop efficient mechanisms to solve real world problems effectively [1], [2], [3],[4] because we do have huge amount of scopes to make our daily lives comfortable and smooth but our resources are limited. With little resource we have to build sustainable yet intelligent systems for our future generations. In addition, we cannot undermine the adverse impact of technologies on our environment. Therefore, we need work and research more and more about energy efficient systems which will give us what we need in a quicker and intelligent way and at the same time keep a close look on the use of resources. Energy efficient design requires the development of new computer-aided design techniques to help explore the trade- off between power and conventional design constraints, i.e., performance and area. Among these techniques, dynamic power management (DPM) [5] and its extensions, such as application-driven/assisted power management [6], [7], and dynamic voltage scaling (DVS) have been extensively applied with good results.DPM is a flexible and general design methodology aiming at controlling performance and power levels of electronic systems by exploiting the idleness of their components. A system is provided with a power manager (PM) that monitors the overall system and component states and controls the power state of each component. This control procedure is called power management policy. Therefore, DPM will have to integrate several methodologies in it. One effective methodology is Reinforcement Learning(RL).Reinforcementlearningisusedinmanydifferentareas.Itisvery

familiar learning method in natural life. The machine learns to achieve a goal by trial and error interaction within a dynamic environment. Target of RL is to minimize its average long term penalty. It is completed by learning a policy mapping between the state and the action. Q learning and SARSA are two of the most common algorithms in reinforcement learning. The whole thesis will work on DPM with RL method by implementing both Q algorithm and SARSA algorithm for the power management solution of LAN cards. After that this thesis will look into the comparison between these two algorithms and will choose the better one for the LAN card by observing the corresponding outputs achieved by using both the RL algorithms.

IMPORTANCE OF ENERGY EFFICIENCY:

While system design is concerned with selection and organization of system components, the system utilization addresses the question of how those components should be used. Electronic systems often consist of one or more microprocessors and a set of devices with multiple low-power states. Many microprocessors support dynamic clock frequency adjustment and some newer devicesalsosupportdynamicsupplyvoltagesetting[8].Thus,atthesystemlevelit is possible to reduce energy by transitioning components into low- power states (dynamic power management) and by changing the frequency and voltage level of the microprocessor(dynamic voltage scaling).

On the other hand, It is obvious that in order to support new generation network services and infrastructure, network operators and Internet service providers need a large number of more sophisticated network devices able to perform complex operations in a scalable way and assure expected quality of service. This is one of

the reasons for the rapid growth of the energy requirements of wired and wireless modern computer networks. Therefore, the energy consumption trends in the next generation networks have been widely discussed and the optimization of total power consumption in today's computer networks has been a considerable research issue [9, 10]. New solutions both in hardware and software have been developed to achieve the desired trade-off between power consumption and the network performance according to the network capacity, current traffic and requirements of the users. The aim is to reduce the gap between the capacity provided by a network for data transfer and the requirements, especially during low traffic periods. In particular,theenergydissipatedinanetworkcanbeminimizedbyswitchingoffidleenerg y consuming components such as routers, line cards, and communication interfaces, and by reducing the speed of processors and link speed. In general, data transfers should be aggregated along as few devices as possible instead of balancing traffic in a whole computer network. Selectively shutting down routers and links in periods of low demand seems to be a good solution for reducing the energy usage due to the fact that typical networks are usually over provisioned. The techniques developed for keeping the connectivity and saving the energy can be successfully used for energy-efficient dynamic management in LANs (local area networks),WANs(wide area networks)as well as in computing centers.

MOTIVATION & OVERVIEW OF THE THESIS:

Power management schemes exist to minimize the power consumption on devices such as desktops, note- books and a number of other portable devices. The schemes used for conserving power are generally implemented by the operating system in the device and use various power-saving techniques such as dynamic voltage scaling (DVS), slowing clocks and using lower power-consuming modes as

provided by the underlying hardware. However, no such dynamic power management schemes are available for internet devices such as routers and switches at the system level. In this paper, we look at the feasibility of introducing such schemes in LAN switches. We chose to begin with LAN devices and in particular LAN switches for several reasons: LAN switches comprise the bulk of network devices in the LAN and they also consume the largest percentage of energy. Given the intention to save energy on switches, we have to decide our methodologies. In order to save energy in a device, we can either turn it off or put it into deep sleep states where most of the components are powered off or clocked at a lower frequency at lower voltage levels. The one caveat is that these approaches can only be used when the device is idle for some minimal amount of time (very frequent power on/off actually uses more power due to spikes in current draw when a device is powered on and, furthermore, devices take a certain amount of time to transition between sleep and wake states that could result in packet losses). Turning our attention to the LAN switch, we note that saving energy here translates to powering off or putting to sleep LAN switch components, interfaces, or entire switches. However, the side effect of putting ports or switches to sleep is that layer 2 protocols running on the switches may be negatively affected. Thus, implementing power management schemes in a switch presents several challenges: switches do not function in isolation and hence slowing or powering down switches can result in performance penalties in terms of network throughput and end-to-end delay, or worse, packetloss.

In this paper, we examine the different questions arising from the approach of putting switch components to sleep. In our thesis work, we used traffic data from our LAN to show that there are significant periods of inactivity in our LAN that can be used for sleeping. We then developed an abstract sleep model for generic

switch architecture and use it to discuss algorithms for sleeping. Furthermore, our work shows our study of the performance of these algorithms on the traffic data collected at various interfaces at different locations in our LAN. We then finally used an efficient energy management system such as Dynamic Power Management (DPM) with Q – learning method to save the energy of LAN card with a suitable algorithm. We studied different sets of power management schemes. Then we studied reinforcement learning method with SARSA algorithm design.

We also developed a suitable algorithm and transferred it into mat lab programming. We finally achieved our desired result and we can say that the power of LAN card can be effectively managed with our suggested methodology.

CHAPTER 02

POWER MANAGEMENT SCHEMES:

A workload that uses a given system component can be represented by a two state finite state machine in terms of how it uses the component: busy and idle. Busy state corresponds to the times during which the workload uses the component to actively perform some processing. For instance, when an application thread in running on the CPU or the hard disk is spinning to serve a block request. Conversely, the idle state corresponds to the instances when the workload is not generating any requests for the component, as a consequence of which, it is inactive or not being utilized. Energy consumption for any workload on an operational system is the product of the power consumption of the system components and the runtime of the workload:

$$E = \int_{t0}^{t1} P(t)\, dt \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Based on this equation, an intuitive way of dynamically achieving energy savings is to reduce power consumption of the system with minimal impact on the execution time of the workload. This will result in reduction in energy consumption proportional to the decrease in power consumption. This approach

towards energy management is referred to as active power management, since it is based on actively managing the power consumption of the system to achieve energy savings. The power consumption can be managed during both the busy as well as the ideal states of the workload. Based on the sates of the workloads (and hence component), i.e. busy or idle, during which power management is performed, the active power management techniques can be divided into two categories: 1) DVFS 2) DPM

When the workload is in the idle state, the system component is inactive and is ability to actively execute work-loads or serve user requests is not required. Consequently, modern system components like CPUs, hard drives, network cards etc. support sleep states, which consume lower but compromise the ability of the component to actively serve workload requests. Dynamically utilizing such sleep states during the idle state to achieve energy saving is referred to as Dynamic Power Management or DPM.

The goal of both the active power management techniques – DPM and DVFS, is to maximize the reduction in power consumption with minimal impact on performance, so that the energy consumption (based on equation 1) can be minimized. The following discussion will provide details on the existing states of the art approaches for active power management. Now, first we have to analyze different sets of energy management systems and choose the best method for our research.

DYNAMIC POWER MANAGEMENT SYSTEM:

The fundamental premise for the applicability of power management schemes is that systems, or system components, experience non-uniform workloads during normal operation time. Non-uniform workloads are common in communication networks and in almost and interactive system.

System-level dynamic power management decreases the energy consumption by selectively placing idle components into lower power states. System resources can be modeled using state-based abstraction where each state trades off performance for power [20]. For example, a system may have an active state, an idle state, and a sleep state that has lower power consumption, but also takes some time to transition to the active state. The transitions between states are controlled by commands issued by a power manager (PM) that observes the workload of the system and decides when and how to force power state transitions. The power manager makes state transition decisions according to the power management policy. The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a constrained optimization problem.

Fig. 1 shows later (a) the system power consumption level over time without DPM, (b) the case when the ideal DPM is applied, and (c) the case when non ideal DPM isapplied.

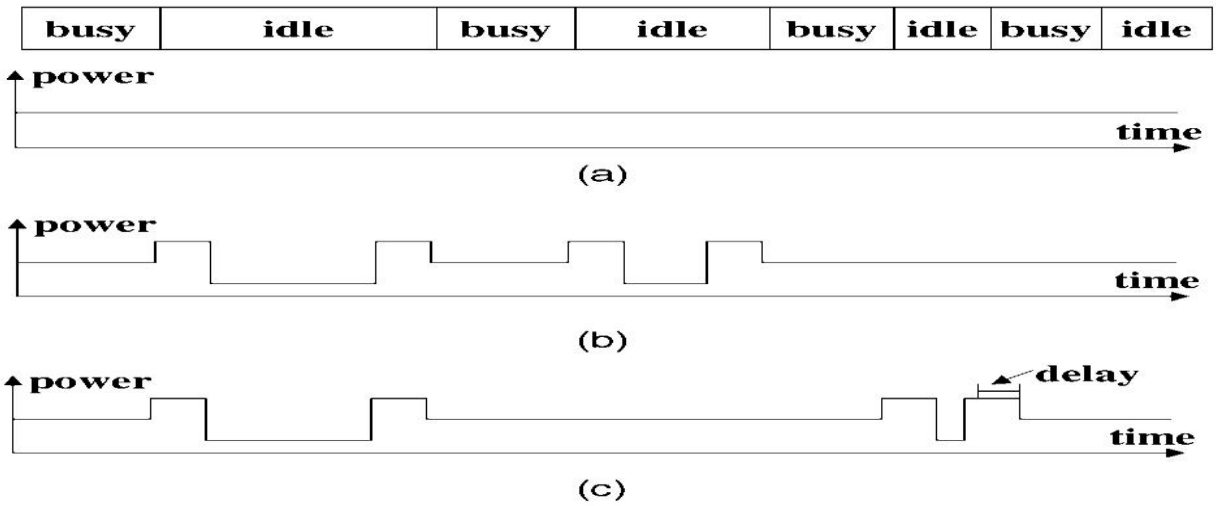


Figure: 01

Non ideal DPM wastes the idle interval at the second idle period and pays a performance penalty at the third idle period. These inefficiencies come from inaccurate prediction of the duration of the idle period or, equivalently, the arrival time of the next request for an idle component. Thus, the ultimate goal in DPM is to minimize the performance penalty and wasted idle intervals while, at the same time, minimizing power. The objective can be achieved by an ideal PM with complete knowledge of present, past and future workloads. In some cases such knowledge can be provided by applications that provide hints on future requirements of the system resources. Unfortunately, the application- driven approach is not viable when applications cannot be modified.

The existing DPM policies can be broadly classified into 3 categories.

1. Timeout Policies

2. Predictive Policies

3. Stochastic Policies

In general, we can model the unavoidable uncertainty of future requests by describing the workload as a stochastic process. Even if the realization of a stochastic process is not known in advance, its properties can be studied and characterized. This assumption is at the basis of many stochastic optimal control approaches that are routinely applied to real-life systems. DPM has been formulated and solved as a discrete-time stochastic optimal control problem by Benini et al. [24]. Also, continuous-time stochastic approaches were proposed in [23],[21],[22].Unfortunately, in many instances of real-life DPM problems, it is hard, if not impossible, to pre characterize the workload of a power management system. Consider, for instance a disk drive in a personal computer (PC). The work load for the drive strongly depends on the application mix that is run on the PC.

This, in turn, depends on the user, on the location, and similar “environmental conditions” which are not known at design or fabrication time. IN these cases, the stochastic process describing the workload is subject to large variation over time. For instance, hard disk work-loads for a workstation drastically change with the time of the day or the day of the week. This characteristic is called non stationary. It is generally hard to achieve robust and high-quality DPM without considering this effect. Optimal control of non stationary stochastic systems is a well- developed field .Several adaptive control approaches are based on an estimation procedure that “learns” online the unknown or time-varying parameters of the system, coupled with a flexible control scheme that selects the optimal control actions based on the

estimated parameters. This approach is also known as the principle of estimation and control. The most common power management policy at the system level is a timeout policy implemented in most operating systems. The drawback of this policy is that it wastes power while waiting for the timeout to e xpire[17].

Predictive policies for hard disks [18] and for interactive terminals [19] force the transition to a low power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. An incorrect estimate can cause both performance and energy penalties. The distribution of idle and busy periods for an interactive terminal is represented as a time series in [32], and approximated with a least-squares regression model. The regression model is used for predicting the duration of future idle periods. A simplified power management policy predicts the duration of an idle period based on the duration of the last activity period. The authors of [32] claim that the simple policy performs almost as well as the complex regression model, and it is much easier to implement. In [19], an improvement over the prediction algorithm of [32] is presented, where idleness prediction is based on a weighted sum of the duration of past idle periods, with geometrically decaying weights. The policy is augmented by a technique that reduces the likelihood of multiple wrong predictions. All these policies are formulated heuristically, and then tested with simulations or measurements to assess their effectiveness.

In contrast, approaches based on stochastic models can guarantee optimal results. Stochastic models use distributions to describe the times between arrivals of user requests (inter-arrival times), the length of time it takes for a device to service a user's request, and the time it takes for the device to transition between its power states. The system model for stochastic optimization can be described either with just memory less distributions (exponential or geometric) or with general distributions [25,26, 27]. Power management policies can also be classified into two

categories by the manner in which decisions are made: discrete time (or clock based) and event driven [25,26, 27]. In addition, policies can be stationary (the same policy applies at any point in time) or non- stationary (the policy changes over time). All stochastic approaches except for the discrete adaptive approach presented in are stationary. The optimality of stochastic approaches depends on the accuracy of the system model and the algorithm used to compute the solution. In both the discrete and the event-driven approaches optimality of the algorithm can be guaranteed since the underlying theoretical model is based on Markov chains. Approaches based on the discrete time setting require policy evaluation even when in low-power state, thus wasting energy. On the other hand, event-driven models based on the exponential distribution show little or no power savings when implemented in real systems since the exponential model does not describe well the request inter-arrival times [25,26, 27].

INTRODUCTION TO REINFORCEMENT LEARNING

All animals and automata exhibit some kind of behavior; they do something in response to the inputs that they receive from the environment they exist in. Some animals and automata change the way the behave over time; given the same input, they may respond differently later on than they did earlier on. Such agents learn. The field of machine learning studies learning algorithms, which specify how the changes in the learner's behavior depend on the inputs received and on feedback from the environment.

Learning algorithms fall into three groups with respect to the sort of feedback that the learner has access to. On the one extreme is supervised learning: for every input, the learner is provided with a target; that is, the environment tells the learner what its response should. The learner then compares its actual response to the target and adjusts its internal memory in such a way that it is more likely to produce the appropriate response the next time it receives the same input. We can think of learning a simple categorization task as supervised learning. For example, if you're learning to recognize the sounds of different musical instruments, and you're told each time what the instrument is, you can compare your own response to the correct one.

On the other extreme is unsupervised learning: the learner receives no feedback from the world at all. Instead the learner's task is to re-represent the inputs in a

more efficient way, as clusters or categories or using a reduced set of dimensions. Unsupervised learning is based on the similarities and differences among the input patterns. It does not result directly in differences in overt behavior because its "outputs" are really internal representations. But these representations can then be used by other parts of the system in a way that affects behavior. Unsupervised learning plays a role in perceptual systems. Visual input, for example, is initially too complex for the nervous system to deal with it directly, and one option is for the nervous system to first reduce the number of dimensions by extracting regularities such as tendencies for regions to be of constant color and for edges to be continuous.

A third alternative, much closer to supervise than unsupervised learning, is reinforcement learning: the learner receives feedback about the appropriateness of its response. For correct responses, reinforcement learning resembles supervised learning: in both cases, the learner receives information that what it did is appropriate. However, the two forms of learning differ significantly for errors, situations in which the learner's behavior is in some way inappropriate. In these situations, supervised learning lets the learner know exactly what it should have done, whereas reinforcement learning only says that the behavior was inappropriate and (usually) how inappropriate it was. In nature, reinforcement learning is much more common than supervised learning. It is rare that a teacher available who can say what should have been done when a mistake is made, and even when such a teacher is available, it is rare that the learner can interpret the teacher's feedback provides direct information about what needs to be changed in the learner, that is, features of the learner's nervous system. Consider an animal that has to learn some aspects of how to walk. It tries out various movements.

Some work -- it moves forward -- and it is rewarded. Others fail -- it stumbles or falls down -- and it is punished with pain.

Thus while much of the focus of machine learning has been on supervised learning, if we are to understand learning in nature, we need to study unsupervised and reinforcement learning. During the workshop, we will explore one reinforcement learning algorithm in some detail.

Reinforcement learning area used in many different areas. It is very familiar learning method in natural life. The machine learns to achieve a goal by trial and error interaction within a dynamic environment.

The general learning model method of Reinforcement Learning is :

1. An Agent
2. A finite state space
3. A set of available action
4. A plenty of function P = S*A →P

To simulate the learning of real biological systems, we need to make some simplifying assumptions about our agent and its behavior. (From now on, we'll refer to our learner as an "agent" to emphasize that it's actively doing things, not just learning what to do.) We will assume that the agent's life is a Markov decision process (MDP). That is,

- The agent perceives a finite set *S* of distinct states in its environment and has a finite set *A* of distinct actions that it can perform.
- At each discrete time step *t*, the agent senses the current state x_t, chooses a current action u_t, and executes it.

- The environment responds with a reward or punishment $r(x_t, u_t)$ and a new state $x_{t+1} = T(x_t, u_t)$. Note that we need to formalize reinforcement as a number, greater for more beneficial, less for more detrimental.
- The reinforcement and next-state functions are not necessarily known to the agent, and they depend only on the current state and action. That is, before learning, the agent may not know what will happen when it takes a particular action in a particular state, but the only relevant information for deciding what action to take is the current state, which the agent does have access to.

For example, consider an agent in a one-dimensional world of cells. In each cell except those on the end, the agent can go either east or west. At the extreme west end lives a swarm of mosquitoes. At the extreme east, there is a mango tree. Anywhere else, the agent receives no information at all from the environment. The agent can sense which cell it is in, but at the beginning of learning, it has no idea what cell it gets to by going east or west and what sort of reinforcement it will receive, if any. On the basis of the punishment it receives if and when it reaches the west end and the reward it receives if and when it reaches the east end, it must learn how to behave anywhere in the world.

The goal of reinforcement learning is to figure out how to choose actions in response to states so that reinforcement is maximized. That is, the agent is learning a policy, a mapping from states to actions. We will divide the agent's policy into two components, how good the agent thinks an action is for a given state and how the agent uses what it knows to choose an action for a given state.

There are several ways to implement the learning process but in this thesis, we will focus firstly on, Q learning, which is a form of reinforcement learning in which the agent learns to assign values to state-action pairs. We need first to make

a distinction between what is true of the world and what the agent *thinks* is true of the world. First let's consider what's true of the world. If an agent is in a particular state and takes a particular action, we are interested in any immediate reinforcement that's received but also in future reinforcements that result from ending up in a new state where further actions can be taken, actions that follow a particular policy. Given a particular action in a particular state followed by behavior that follows a particular policy, the agent will receive a particular set of reinforcements. This is a fact about the world. In the simplest case, the Q-value for a state-action pair is the sum of all of these reinforcements, and the Q-value function is the function that maps from state-action pairs to values. But the sum of all future reinforcements may be infinite when there is no terminal state, and besides, we may want to weight the future less than the here-and-now, so instead a discounted cumulative reinforcement is normally used: future reinforcements are weights by a value *gamma* between 0 and 1 (see below for mathematical details). A higher value of *gamma* means that the future matters more for the Q-value of a given action in a given state.

If the agent knew the Q-values of every state-action pair, it could use this information to select an action for each state. The problem is that the agent initially has no idea what the Q-values of any state-action pairs are. The agent's goal, then, is to settle on an optimal Q-value function, one which that assigns the appropriate values for all state/action pairs. But Q-values depend on future reinforcements, as well as current ones. How can the agent learn Q-values when it only seems to have access to immediate reinforcement? It learns using these two principles, which are the essence of reinforcement learning:

- If an action in a given state causes something bad to happen, learn not to do that action in that situation. If an action in a given state causes something good to happen, learn to do that action in that situation.
- If all actions in a given state cause something bad to happen, learn to avoid that state. That is, don't take actions in other states that would lead you to be in that bad state. If any action in a given state causes something good to happen, learn to like that state.

The second principle is the one that makes the reinforcement learning magic happen. It permits the agent to learn high or low values for particular actions from a particular state, even when there is no immediate reinforcement associated with those actions. For example, in our mosquito-mango world, the agent receives a reward when it reaches the east end from the cell just to the west of it. It now knows that *that* cell is a good one to go to because you can get rewarded in only one move from it.

That is, the optimal Q-value of for a particular action in a particular state is the sum of the reinforcement received when that action is taken and the discounted best Q-value for the state that is reached by taking that action. The agent would like to approach this value for each state-action pair. At any given time during learning, the agent stores a particular Q-value for each state-action pair. At the beginning of learning, this value is random or set at some default. Learning should move it closer to its optimal value.

An important aspect of reinforcement learning is the constraint that only Q-values for actions that are actually attempted in states that actually occurred are updated. Nothing is learned about an action that is not tried. The upshot of this is that the agent should try a range of actions, especially early on in learning, in order to have

an idea what works and what doesn't. More precisely, on any given time step, the agent has a choice: it can pick the action with the highest Q-value for the state it is in (exploitation), or it can pick an action randomly (exploration). Note that there is a tradeoff between these two strategies. Exploitation, because it is based on what the agent knows about the world, is more likely to result in benefits to the agent. On the other hand, exploration is necessary so that actions that would not be tried otherwise can be learned about. We formalize the action decision process in terms of probabilities. The simplest possibility is to flip a coin and with a certain probability exploit your knowledge (pick the action with the highest Q-value) or explore (pick a random action). A better one, which we will use, is to assign a probability to each action, basing it on the the Q-value for the action in the state. The higher an action's Q-value, the greater the probability of choosing it. But because these are still probabilities, there is still a chance of picking an action that currently has a low Q-value. It also makes sense to have the probability of exploration depend on the length of the time the agent has been learning. Early in learning, it is better to explore because the knowledge the agent has gained so far is not very reliable and because a number of options may still need to be tried. Later in learning, exploitation makes more sense because, with experience, the agent can be more confident about what it knows. The equation for the probability of choosing an action u_t, then, is

System level power management must consider the uncertainty and variability that comes from the environment, the application and the hardware. A robust power management technique must be able to learn the optimal decision from past history and improve itself as the environment changes. This paper presents a novel on- line power management technique based on model-free constrained reinforcement learning (RL). It learns the best power management policy that gives the minimum

power consumption for a given performance constraint without any prior information of workload. Compared with existing machine learning based power management techniques, the RL based learning is capable of exploring the trade- off in the power-performance design space and converging to a better power management policy. Experimental results show that the proposed RL based power management achieves 24% and 3% reduction in power and latency respectively comparing to the existing expert based power management.

The proposed dynamic power management (DPM) framework is based on model-free reinforcement learning (RL) technique that performs learning and power management in a continuous-time and event-driven manner. It has fast convergence rate and less reliance on the Mark ovian property. The presented DPM framework can dynamically perform power management according to both system's workload and battery state of charge. It uses the reinforcement learning

(RL) technique to determine the output power state for LAN card for a closed-loop policy. Experiments on measured data traces demonstrate the superior performance of the proposed dynamic power management method in comparison with prior works[35].

BRIEF THEORY FOR MARKOV DECISION PROCESS:

ResearchonMarkovDecisionProcessisanattractivewaytodealwithuncertainties inthe areaofartificialintelligence[30]andevenincontrolsystem[31,32].

Markov decision process (MDP) is a popular and attractive way for modeling the decisionprocesseswithuncertain-ties.Ithasbeenappliedtomanyproblems,

Including queuing systems, reinforcement learning, finite state machine, decision network, inventory control, etc. In this article, we extend our research from the MDP to the Continuous-time MDP (CTMDP). The main goal is to find a policy iteration algorithm based on the temporal difference learning method to solve the CTMDP problem.

The MDP, also referred to as controlled Markov chain, describes a problem in which a single agent must choose an action at every node of the chain in order to maximize some reward-based optimization criterion. The basic idea of the Mark ovian property is that the occurrence of the current state only depends on the previous state.

Most literature about the MDP paid attention to the decision processes without considering the effects caused by transition time which commences at a state and continues until the next state arrives. However, for some contexts with a dynamic environment, the time length of transition appears to be an important factor for the quality of solution. For ex- ample, for modeling a robotic path planning problem as the MDP problem, the state is defined as its position and we consider the transition from state i to state j after an action a . Under the traditional MDP model, no matter how long the transition time is, the solution will be the same because the transition probability doesn't change. In fact, it is more reasonable to assume that these transitions whose transition time is closer to the expected transition time will happen more probably. The expected transition time might be associated with the maneuverability of players and the dynamic properties of the environment.

It is expected that the Continuous-time MDP (CTMDP) will provide a more

suitable model for such dynamic problems. One of the major differences between MDP and CT- MDP is that the MDP corresponds to a Markov chain X^{+} but the CTMDP corresponds to a Markov stochastic process *Xt* (For simplicity, we denote $X(t)$ as *Xt* in this article). The Mark ovian property can be described as $P(Xs+t=j|Xu;u \leq s)=P(Xs+t=j|Xs)$,

Sustained $0 < t, s < \infty$.Among Markov theories, a Markov process can be described absolutely by two parts: the embed Markov chain and the expected stay time at each state. As a result, the CTMDP has not only considered the Markovian property but also the time property at each transition. Under the probability model based on the MDP, reinforcement learning works well to train the agent to perform at high levels in many complex domains (for example, game playing, helicopter auto flying, etc). Temporal difference is the most important approach to train the evaluation function in reinforcement learning, and then the optimal policy can be obtained.

CHAPTER 03

THESIS METHODOLOGIES:

We worked towards a sustainable power management method for LAN card system. We chose to use dynamic power management with Q reinforcement learning method. We worked on Dynamic power management method as relevant literature about DPM demonstrates various approaches. In the simplest approach, the greedy policy, a *device* transitions to the sleep state as soon as it is idle. Here, the term *device* refers to any electronic equipment that serves a particular purpose and has more than one modes of power consumption (or performance). The greedy policy can give the best power optimization aslong as the *requests* arrive at long time intervals. A *request* represents a task generated by an application that needs processing. Another simple heuristic policy is the time-out policy where a device is shut down after it has been idle for a certain threshold of time period. The time-out policy can be static or adaptive [5][20] which adjusts the time-out threshold based on the previous idle period history. The main shortcoming of time-out policies is the power wasted during the time-out period, especially when the workload (arrival rate of requests) is lower. This problem is better dealt by the predictive policies which work on a system model that is learned from the history information in order to best adjust themselves to the dynamic system of advice.

The basic idea in predictive policies is to predict the length of idle periods and shut down the device when the predicted idle period is longer than a certain threshold time period. Nevertheless, predictive policies do share a few limitations. First, they do not consider the response time of the device. Second, they do not deal with general system models where multiple incoming requests can be queued before processing. Third, they cannot perform well with non-stationary requests where the workload model is unknown.

Some of these limitations (queuing, power-performance trade-off) are addressed by stochastic policies [9] .These approaches make probabilistic assumptions about the usage patterns of a device and exploit the nature of the probability distribution to formulate an optimization problem, the solution of which derives an optimal DPM strategy. The device states and queues in stochastic policies are generally modeled as Markov chains. These policies do provide a flexible way to control the trade-off between power consumption and device response depending on the optimization constraints. However, Markov model is generally assumed to be stationary and known in advance. Therefore, these policies no longer remain optimal as workload becomes non- stationary.

From the above discussion, it is evident that the performance of any selected policy heavily depends on the workload. Real workloads are usually non-stationary and compose a strict limitation on the success of any single policy. A model-free, machine learning approach can cope with this issue by interacting with the environment, implementing certain actions, evaluating the

effects of the implemented actions and adjusting itself according to the environment. Compared with the existing machine learning DPM approaches, the RL based DPM approaches can deal with the non-stationary workloads in a much better way and can explore the trade-off between a system's power consumption and response time. The model-free, RL based approaches presented in use online learning algorithms that dynamically select the best DPM policies from a set of pre-selected candidate policies. These algorithms do lead to optimal DPM policies, but they heavily rely on and are limited to the pre-selected candidate policies. In [21], authors propose an enhanced RL algorithm for system-level DPM. It is also a model-free approach that does not require prior knowledge of the state-transition probabilities. However, the number of state-action pairs in this system is quite large, which may result in increased computational complexity and slow convergence..

In a recent work [18], a model-free, RL based DPM approach was used for non-stationary workload. The learning agent in this approach receives partial information about the workload from a workload estimator using a ML-ANN with back propagation algorithm. Based on the estimated workload, this approach evaluates certain time-out values in idle state and waits for certain number of requests to be accumulated in the service queue when the system is in sleep state. Workload estimation using a ML-ANN achieves higher accuracy with the traffic data and the results show that the algorithm is capable of exploring the trade-off in the power-performance design space and converging to an optimal policy. However, since the algorithm waits for certain number of requests in the queue, a drawback of this approach is the high latency in requests processing when the workload drops abruptly.

We propose a novel RL based DPM algorithm in this paper for the power

management of our LAN card. The proposed algorithm uses time-out values both in sleep and idle states with workload estimation from a ML-ANN. Apart from this, we use multiple-states update in both sleep and idle modes and use a better exploration-exploitation policy to help algorithm converge fast and explore the design space deeper. As compared to the algorithm in

[18], our results show that the algorithm proposed in this paper can find a better trade-off curve of power-performance and results in a much lower latency while keeping the power consumption at an acceptable level.

Now firstly, we approached towards RL (Reinforcement learning) methodology. RL is a machine learning approach that is concerned with mapping situations to actions, in order to minimize a numerical penalty (or cost). As opposed to other machine learning approaches, for example supervised learning which is based on learning from examples provided by an external supervisor, RL is not dictated which actions to take. Instead, it must interact with the environment and discover the actions which yield the most reward (minimum penalty) by trying them. During the learning process, the *agent* observes the environment and issues appropriate actions based on the system state. As a result, the system changes state and the new state assign the agent a penalty (or reward) which indicates the value (appropriateness) of the state transition. The overall goal of the learning process is to maximize the scalar reward (or minimize the penalty) in each state.

RL assumes that the system dynamics follow Markov property, i.e., then

next state $s^{r} \in S$ and immediate reward r depend only on the current state $s \in S$

and action $a \in A$, as given by equation 1

$$Pr\{st+1=s, rt^{r}+1=r|st,at\}\ldots\ldots\ldots\ldots...(2)$$

Where *Pr* is the probability of reaching state s^{r} and getting reward *r* at time *t* +1.Apolicy,π,isamappingfromeachstate,$s \in S$,andaction,$a \in A(s)$to the probability of taking action *a* when in state *s*. Informally, the *value* of a state *s*under a policy π, denoted by $V^{\pi}(s)$, is the expected reward when starting in states and following the policyπ

thereafter. We can define $V^{\pi}(s)$ as follows [12]:

$$V^{\pi}(s)=E\pi\{Rt|st=s\}$$

Where $E\pi\{.\}$ denotes the expected value given that the agent follows policy π,

and *t* is any time step, $\gamma \in (0, 1)$ is a discount factor. Similarly, we define the value of taking action *a*in state *s*under a policy π, denoted by $Q^{\pi}(s, a)$, as the expected reward starting from *s*, taking action *a*, and thereafter following policy π.

$$Q^{\pi}(s,a)=E\pi\{Rt|st=s,at=a\}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

In a typical Markovian environment, we use a value-iteration algorithm with state transition probabilities to take an action in some state *s*. However, in a model-free learning, the agent has no prior information about the state transition probabilities. Therefore, we need an estimate of the value function described in equation 3.

A variant of RL, the Q-learning is the simplest form of RL that can directly approximate the value function $V^{\pi}(s)$ independent of the policy being followed. The Q-learning principle is given in equation 4.

$$\forall(s, a) \in S \times A : Q(st, at) = Q(st, at) + \alpha t(st, at)\{rt+1 + \gamma \max a\, Q(st+1, a) - Q(st, at)\} \ldots. (4)$$

Where $\alpha t(st, at) \in (0, 1)$ is the learning rate. $Q^{\pi}(s, a)$ for each state-action pair represents the expected long-term reward if the system starts from state *s*, takes action *a*, and thereafter follows policy π. Based on this value function, the agent decides which action should be taken in current state to achieve the maximum long-termre ward, without knowing the state-transition probabilities.

The learning algorithm is described below. *M* represents the transition matrix which keeps track of the visited states, actions, corresponding cost and other parameters. In each decision epoch, the system finds itself either in sleep state or in idle state. In both the states, the PM selects a time-out value (Equation 11) and relinquishes the control until the time-out period expires (or if some requests arrive during the time-out period in idle state). At the end of the time-out period (or when the time-out period is forced to terminate by

Algorithm 1 RL based time-out policy :

Here is the algorithm used in our thesis,

Require: Power-performance parameter $\lambda \in (0, 1)$

1. Initialize *Q*, *M* and probability matrix *pr* arbitrarily.
   While Policy not good enough do

2. Obtain the current workload estimation (Sec.V.B)
3. Get the current observation: (*s*,*a*)
4. Select an action, *a*, with probability *pr* (Sec. V.E)
5. Execute the selected action
6. Calculate cost of the last action: $ct+1(s, a)$ (Sec.V.A)
7. Update the learning rate: $\alpha t(s, a)$ (Sec.V.D)
8. Update *M* with new state-action pair
9. Update Q-value: $Q^{t+1}(s, a)$ (Sec.V.C)

end while

Q LEARNING ALGORITHM:

Our target is to minimize its average long term penalty. It is completed by learning a policy mapping between the state and the action. Q learning is one of the most common algorithms in reinforcement learning. by performing action the system moves from one states to another states. the new agent gives us the new value of the state transition. The agent keeps value (s,a) for each state action pair, which represents the starts from state s, action a, thereafter policy , the core of the Q learning function algorithm update value of the function. For a value for each state and action pair is initially chosen by the designer and it will be updated each time an action by the following equation:

$$Q(s,t) \leftarrow Q(s,t) + Є(s,t) * [P_{t+1} + \gamma \min Q(S_{t+1}, a) - Q(S_t, a_t)]$$

Here,

Q(s,t) = Old value

Є(s,t) = Learning rate

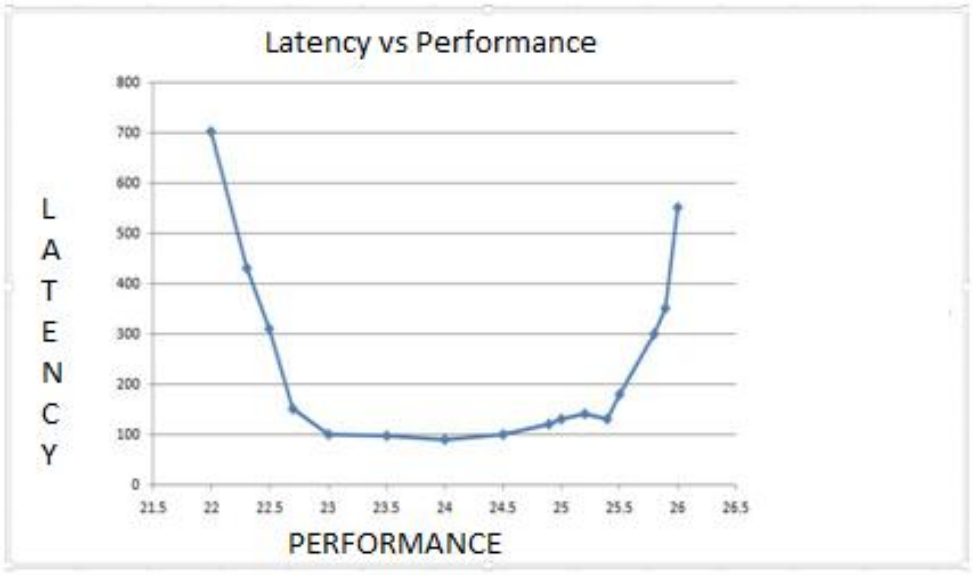
P_{t+1} = Penalty

γ = Discount factor

minQ(S_{t+1},a) = Min Future value

Q(S_t, a_t) = Old value

Q Learning Result Final Graph :



SARSA ALGORITHM:

SARSA (so called because it uses state-action-reward-state-action experiences to update the *Q*-values) is an *on-policy* reinforcement learning algorithm that estimates the value of the policy being followed. An experience in SARSA is of the form $\langle s,a,r,s',a' \rangle$, which means that the agent was in state s, did action a, received reward r, and ended up in state s', from which it decided to do action a'. This provides a new experience to update $Q(s,a)$. The new value that this experience provides is $r+\gamma Q(s',a')$.

SARSA takes into account the current exploration policy which, for example, may be greedy with random steps. It can find a different policy than Q-learning in situations when exploring may incur large penalties. For example, when a robot goes near the top of stairs, even if this is an optimal policy, it may be dangerous for exploration steps. SARSA will discover this and adopt a policy that keeps the robot away from the stairs. It will find a policy that is optimal, taking into account the exploration inherent in the policy.

Controller SARSA(S,A,γ,α)
inputs:
S is a set of states A
is a set of actions γ
the discount
α is the step size

internal state:
real array *Q[S,A]*
previous state *s*
previous action *a*
Begin
Initialize *Q[S,A]* arbitrarily
observe current state *s*
Select action *a* using a policy based on *Q* **repeat forever:**
carry out an action *a*
observe reward *r* and state *s′*
select action *a′* using a policy based on *Q*
Q[s,a] ←Q[s,a] + α(r+ γQ[s′,a′] - Q[s,a]) s ←s′
a ←a′
end-repeat
End

COMPARISON BETWEEN THE SARSA LEARNING AND Q LEARNING ALGORITHM:

SARSA stands for State-Action-Reward-State-Action. In SARSA, the agent starts in state 1, performs action 1, and gets a reward (reward 1). Now, it's in state 2 and performs another action (action 2) and gets the reward from this state (reward 2) before it goes back and updates the value of action 1 performed in state 1.

In contrast, in Q-learning the agent starts in state 1, performs action 1 and gets a reward (reward 1), and then looks and sees what the maximum possible reward for an action is in state 2, and uses that to update the action value of performing action 1 in state 1. So the difference is in the way the future reward is found. In Q-learning it's simply the highest possible action that can be taken from state 2, and in SARSA it's the value of the *actual* action that wastaken.

This means that SARSA takes into account the control policy by which the agent is moving, and incorporates that into its update of action values, where Q-learning simply assumes that an optimal policy is being followed. This difference can be a little difficult conceptually to tease out at first but with an example will hopefully become clear.

SARSA is better than Q-learning in every aspect, the benefit is that it takes into account what your actual system policy, rather than just assuming that you're doing the right thing. But if you have a system where your policy is changing a lot it could be much more desirable to use the Q-learning approach and learn assuming that you're moving optimally, and then incorporate that however you will, rather than having a SARSA system that tries to account for your constant changing movement policy.

In a scenario where we only have a few episodes for learning, there's no real difference between the two, as neither was built to address that situation more efficiently than the other.

THE LEARNING ELEMENTS:

This section describes various elements of the learning, including the cost function, workload estimation, updating learning rate and experimental result.

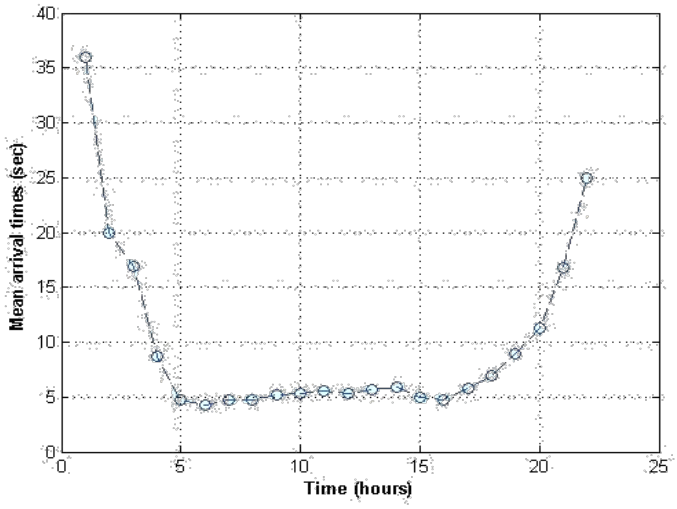
❖ COSTFUNCTION

In the learning algorithm, we use *cost* instead of *reward* which can be treated in the similar way. The cost assigned to an action is a weighted combination of the average power consumption incurred due to the action and the performance penalty. We consider the average latency per request as the performance measure which is equal to the average queuing time plus the average execution time. The cost function is given in equation 5.

$$ct(s, a, \lambda) = \lambda \times \frac{1}{tk+1 - tk} \times \sum_{j=tk}^{tk+1} Pj + (1 - \lambda) lt(s, a) \quad \ldots\ldots\ldots\ldots \quad (5)$$

In the above expression, $tk+1 - tk$ is the time that the SP remains in state s, and

$\lambda \in (0, 1)$ is power-performance trade-off parameter. For $\lambda \rightarrow 0$, the learning algorithm gives more importance to latency, thus resulting in a higher power consumption. On the other hand, when $\lambda \rightarrow 1$, the learning algorithm turns to aggressive power savings, resulting in higher latency. The value of λ can be varied slowly from 0 to 1 to obtain the pare to-optimal trade-off curve.



❖ UPDATING LEARNINGRATE

The learning rate $\alpha t(s, a)$ is decreased slowly in such a way that it reflects the degree to which a state-action pair has been chosen in the recent past. It is calculated as:

$$a_{t\,(s\,,a)} = \frac{\xi}{Visited\ (s,a)}$$

Where ξ is a positive constant. Every time a state-action pair (s, a) is visited with this learning rate, the difference between its estimated Q-value $Q^{(t+1)}(s, a)$ and the current Q-value $Q^{t}(s, a)$ approaches to zero. Hence, for all state-action pairs, the algorithm converges to an optimal policy.

❖ WORKLOAD ESTIMATOR:

In this section we want an extension to the DTMDP model in continuous time process. In CTMDP power management (PM) will set in the discrete time settings. Assume that system transition is follow exponential distribution. We want to show that parameters are uniquely based on exponential and also idle state also can high energy coast and give best performance because of the power manager makes decision as early as possible system goes idle. The decision stays this position until the state before revising the decision. SMDP model usually treat a simple distribution at the same time with an exponential distribution. For the sleep state of the LAN card transition modeled using uniform distribution, exponential distribution model present the user request. It works according to use command.

Exponential distribution model used to model arrived in the active states. Now we want to show that the arrival in the idle states are more than better when filtering out small inter arrival time. In this work we present the time indexed Markov chain SMDP model (TISMDP) which is non exponential arrival distribution coupled with uniform transition distribution. The strategy of the time indexed SMDP model can be solved in polynomial time by linear optimization problem. Build the policy in this way.

We do not only find out optimal result for the policy optimization the TISMDP model, we also representation the simulation and more importantly real measurements of WLAN card. Result can be 3 times larger for WLAN card according this exponential equation

$$E(t) = 1\text{-}e^{-(\lambda t)}.$$

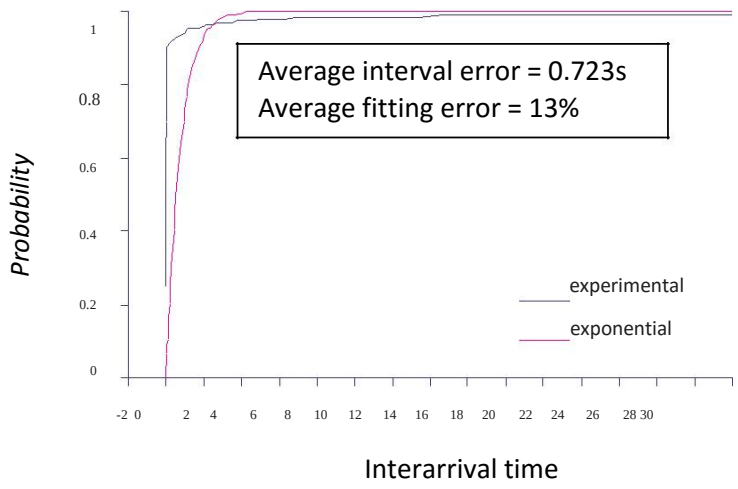
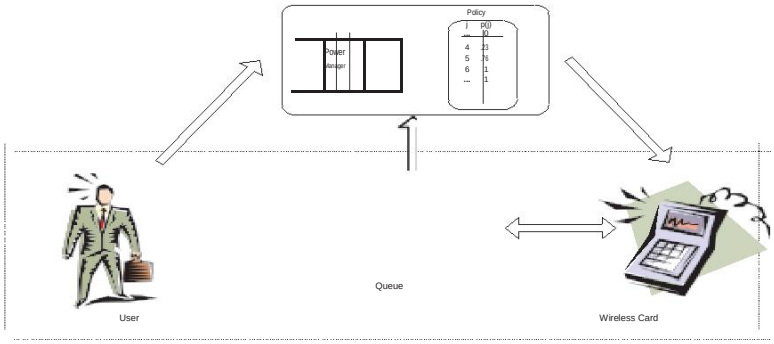


Figure: Curve for the exponential equation

SYSTEM MODEL:

The optimization of energy consumption under performance constraint (or vice versa) is performed for three different devices: a hard disk in a laptop computer, a personal communication interactive device, Smart Badge [33], and a WLAN card [34]



The system modeled consists of the user, device (hard disk, Smart Badge or WLAN card) with queue (the buyer associated with the device) as shown in figure1. The power manager observes the all event occurrences of interest and makes decisions on what state the system should transition to next, in order to minimize energy consumption for a given performance constraint.

BLOCK DIAGRAM OF THE SYSTEM MODEL:

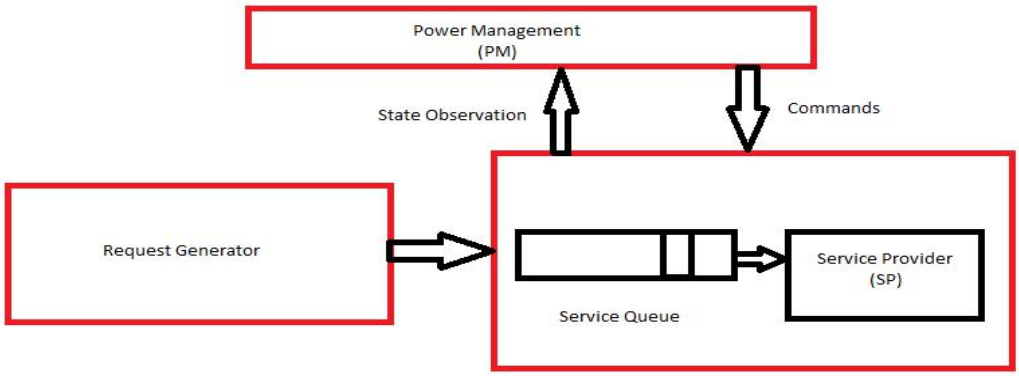


Figure: System Model (block)

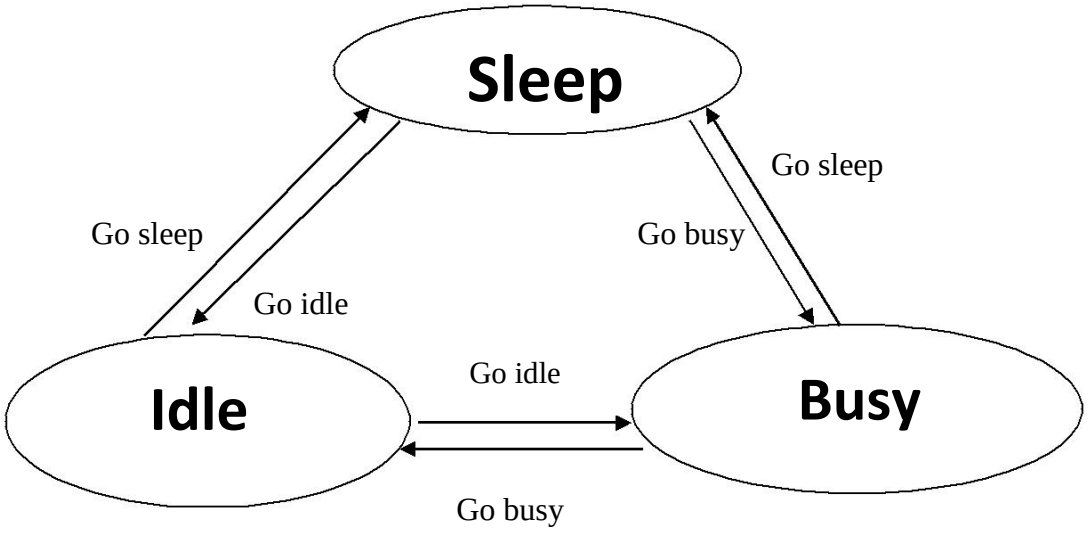
The consumption of energy under performance constant is performed for three different vice versa such as a hard disk in a laptop, computer or personal communication and a LAN card. A system design consists of the demand of the user device hard disk or LAN card with queue the buffer associated with the device as shown in figure-1. The power manager observes the all event of interest and makes decision on what state the system should transition to next, performance remains same when the power will consume.

STATE ACTION AND THEIR TRANSFER RATE:

To solve Q learning method, we assumed a model based on power management system. It consists of two parts- hardware and software. The hardware could be devices such as hard disk, processor etc. OS, application software, user input etc. are the example of software part. The user always observes the control knobs. Some of I/O requests and software system activity depends on operating system, based on the information. The current system state will be classified and the penalty of current state action pair will be calculated.

Now the action of the system consists:

1) Go-Sleep.
2) Go-Idle.
3) Go-Busy.



Go idle	Go sleep	Go busy
0.7w	0w	0.9w

USER MODEL:

We collected some user request and observed for the computer hard disk running a windows operating system with standard software. For WLAN card we used tcp dump to get user request arrival time for two different applications. Not only we obtain optimal result for strategy optimization using the TISMDP mode but also presenting simulation and the real measurement of a LAN card.

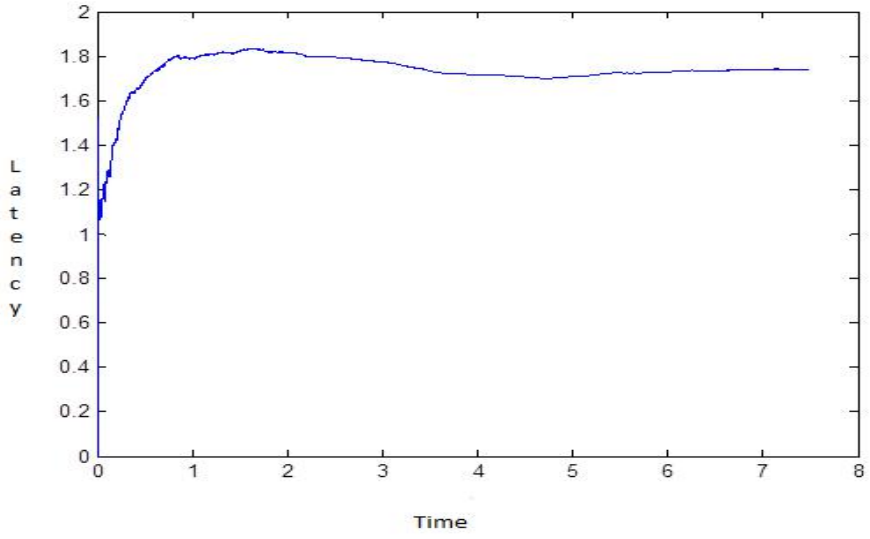


Figure : Latency vs. time curve

Here is the latency vs. time curve which is unstable at the beginning but with the increment of time it is becoming stable and it becomes 1.8. It shows the stability of the digital system which is under consideration for the research. Using Q learning, we achieved this stability for the latency. So, Q learning algorithm helps to reach convergence level for latency.

CHAPTER 04

CODE:

We used matlab for our thesis's simulation. The code which we used to determine the LAN card's power management algorithm with reinforcement Q learning method:

```
function action=  action_ selector (Q, rule_ number, epsilon)

%% This function returns the next selected actions using epsilon-greedy
policy global NA

Global fql % global parameters
initialized ran=rand(1);

% action index is from 1 to J=maximum number of actions which is the same

% as number of columns in Q

% setting the exploration probability

exploitation probability=1-epsilon;

% Selecting an action via epsilon-greedy policy

if ran<exploitation probability

 % exploit

 [max Q factor,index]=max(Q(rule_number,:)); % note that max/min should be
in accordance with reward/cost as reinforcement signal

action=index;
```

```
else % explore
action=ceil(rand(1)*NA);
end
end
```

2.

```
function reward=reward calculator (current_state,next_state)
%% This function calculates the reinforcement signal
% Here we can amend the reward function based on the problem formulation,
% e.g., VS caler paper
% state=[workload, rt, throughput, #VM], DESIRED_RT=Response time
SLO global DESIRED_RT W MAX_THROUGHPUT MAX_VM
ifnext_state(2)<DESIRED_RT
slaFactor=0;
elseifnext_state(2)>2*DESIRED
_RT slaFactor=1;
else slaFactor=(next_state(2)-
DESIRED_RT)/DESIRED_RT; end

end
Ut_1=W(1)*next_state(3)/MAX_THROUGHPUT+W(2)*(1-
next_state(4)/MAX_VM)+W(3)*(1-slaFactor);
 ifcurrent_state(2)<DESIRED_RT
```

```
slaFactor=0;
else
ifcurrent_state(2)>2*DESIRED_RT
slaFactor=1;
else
slaFactor=(current_state(2)-DESIRED_RT)/DESIRED_RT;
end
end
Ut=W(1)*current_state(3)/MAX_THROUGHPUT+W(2)*(1-
current_state(4)/MAX_VM)+W(3)*(1-slaFactor);
reward=Ut_1-Ut;
end
```

SIMULATION GRAPHS:

1)

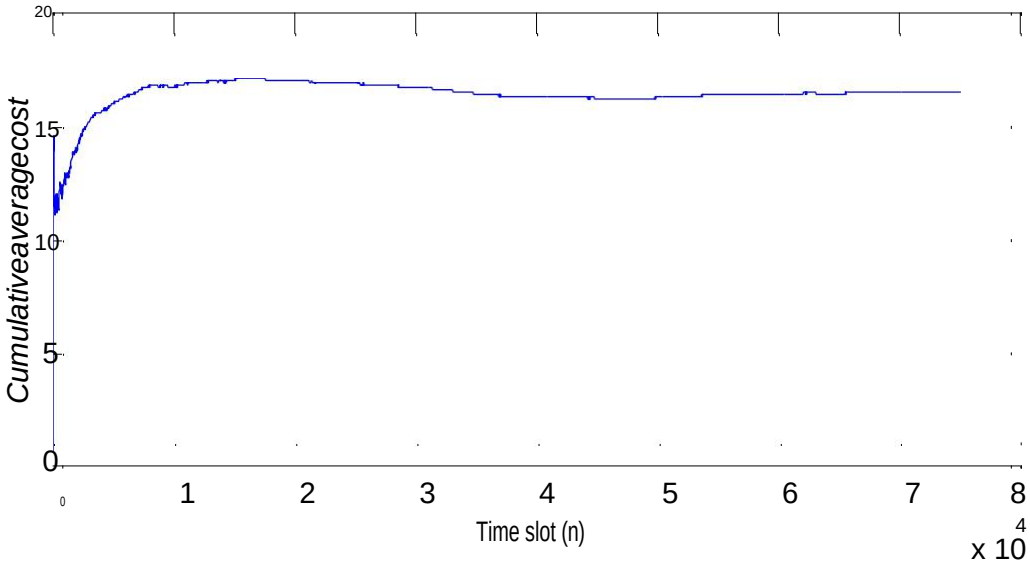


Figure: Cumulating average cost vs. time slotgraph

Here, this graph is a cumulative average vs. time slot graph which at the very beginning is fluctuating but then using Q learning, it reaches an optimal stage and reaches convergence.

2)

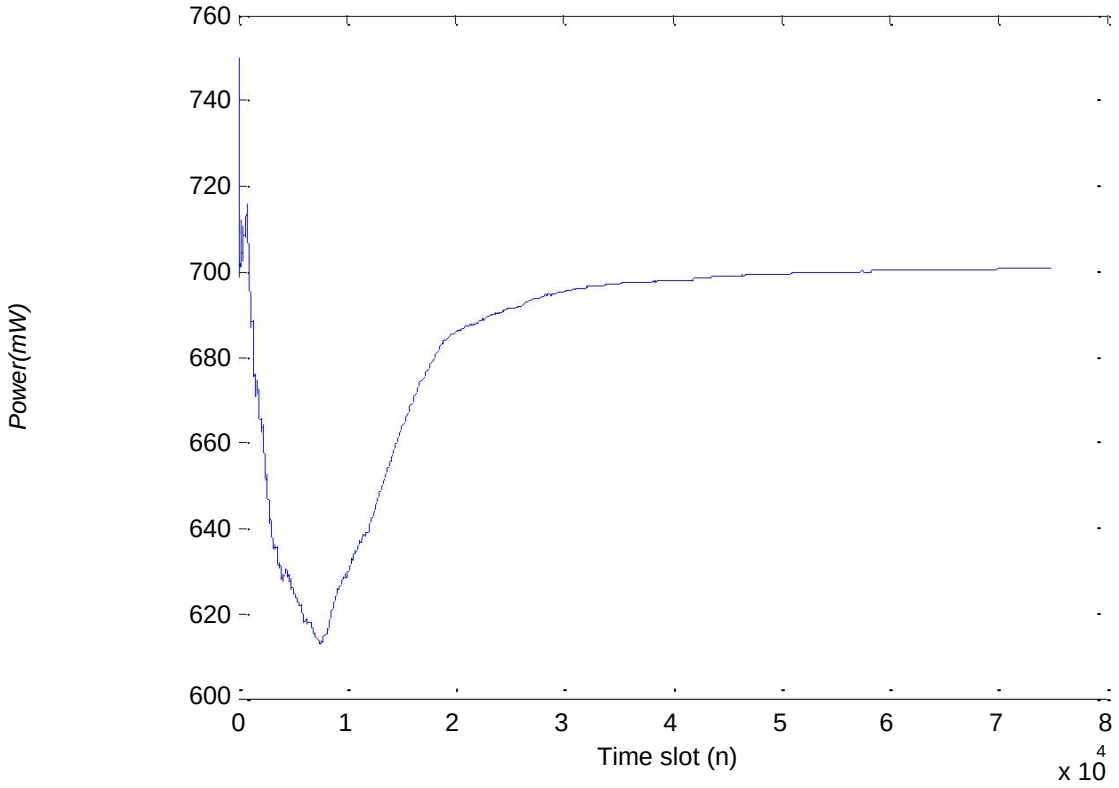


Figure: Power vs. time slot graph

Here , the graph is fluctuating and the dropping rapidly to 610 mW but after the increment of time it goes up again and becomes stable at 700 mW. This graph also shows the power consumption stability of the system considered.

FINAL GRAPH:

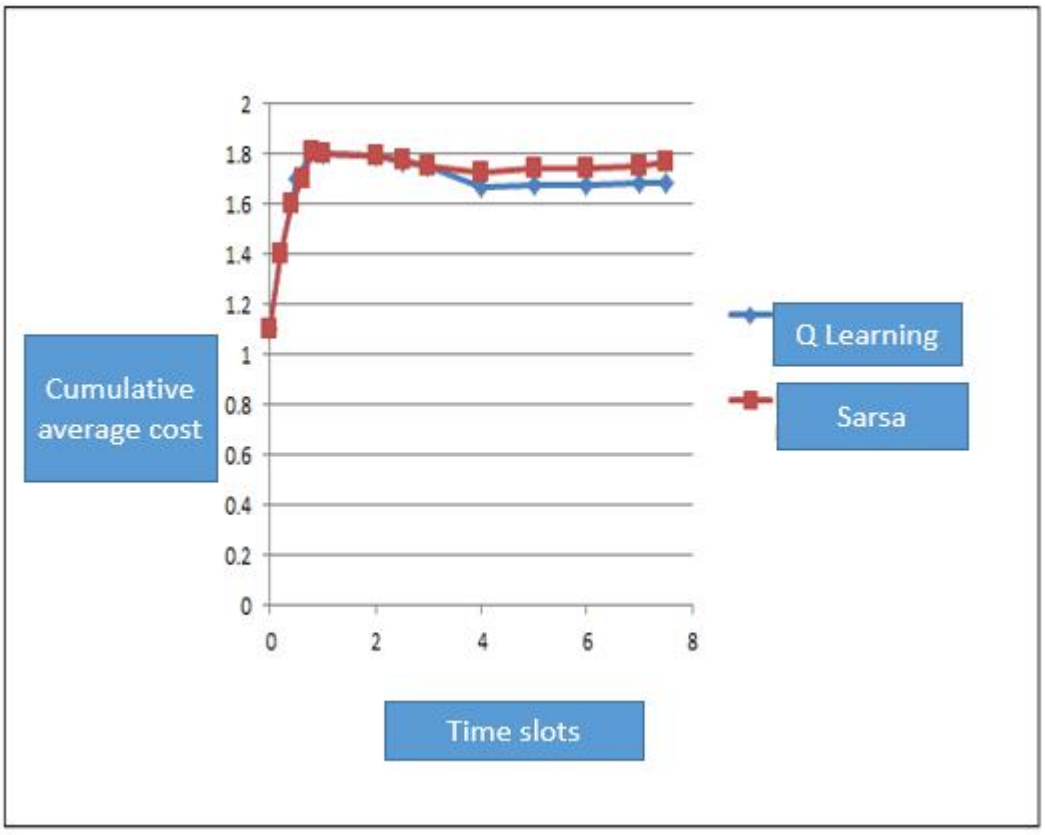


Figure: Comparison graph between Q and SARSA

Here is the performance vs. the latency curve. We do not want high rate of performance with high rate of latency for any device as it is not an ideal scenario. Our prime target was to find a point where performance is satisfactory and latency is low which is called pare to optimal point. The pare to optimal point here is 24.5 which is a better point than any other considered. We also applied both Q learning and SARSA algorithm. We can clearly see that SARSA gives better results than Q learning. Q learning gives slightly distorted result where SARSA gives a smooth result which is very important. That is why we compared Q learning algorithm with SARSA algorithm and took the result from SARSA algorithm for our final achievement.

CHAPTER 05

FINAL ACHIEVEMENTS:

We have presented elaborately my work and research so far. We have presented dynamic power management methodologies to solve the power management issues of digital devices. We also have introduced reinforcement Q learning along with the DPM to have a better result for digital device's power management system. We did not take rest unless we got a better result and therefore, introduced another learning algorithm which is SARSA. We have worked with WLAN card as a digital device and through our research; we have achieved a solution for the problem associated with its power wastage issue. We also have considered no noise while dealing with the WLAN card power management schemes. The latency vs. performance curve shows the pare to optimal point is 24.5 and it is a very convincing result. We also found out that SARSA gives a better result than Q learning algorithm and we implemented that finding in our research work. This result can pave the way for more energy efficient software and hardware design in future.

SCOPES OF FUTURE WORK:

Although much research has been devoted to energy efficient system design and utilization, this area has not yet reached complete maturity. There are still quite a few limitations to overcome.

The dynamic power management algorithms we presented assume stationary workloads. Although adaptation can be done using the methodology discussed in [14], another approach would be to develop a dynamic scheduler that adaptively changes the mode of operation of system components based on non-stationary workload, thermal control and battery conditions. Such scheduler would need close communication of the energy consumption and performance needs between the operating system, the applications and the hardware. The scheduler would also address the limitation of my research. In effect, this approach requires energy aware operating system that allows the dynamic power manager to have close interaction with the task scheduler and the process manager.

As system designers become more conscious of power dissipation issues and an increasing number of power-optimized commodity components is made available, the new generation of power optimization tools is needed to choose and manage such components, and guide the system designer towards power-efficient implementation. The cycle-accurate energy consumption simulator and profiler are just samples of what might be possible. Similar tools, with many more component models (e.g. model of the wireless link) and multiple abstraction levels are needed. More importantly, the methodology for energy efficient software design is still in its infancy. The compilers are just beginning to consider energy consumption as a criterion in code optimization. Some optimizations can be automated at the compiler level, but for others it may be more appropriate to develop a system that

can guide the designer in selection and implementation of appropriate optimizations. Energy efficient design and utilization at the system level will continue to be a critical research topic in the next few years as there are still many unsolved problems and open issues.

REFERENCES:

1. Chandrakasan and R. Brodersen, *Low-Power Digital CMOS Design.* Kluwer, 1995

2. J. Monteiro and S. Devadas, *Computer-Aided Techniques for Low-Power Sequential Logic Circuits.* Kluwer,1997

3. *Low-Power Design in Deep Submicron Electronics*, W. NebelandJ. Mermet, Eds. Kluwer 1997

4. *LowPower Design Methodologies,* J.M.RabaeyandM.Pedram,eds.Kluwer,1996

5. L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Kluwer,1997.

6. R.KravetsandP.Krishinan,"Application-DrivenPower Management for Mobile Communication," Wireless *Networks, vol*.6,

7. J. Flinn and M. Satyanarayanan, "Energy-Aware Adaptation for MobileApplications,"*Proc.Symp.OperatingSystemsPrinciples*, pp. 48-63,1999.

8. . Kołodziej, S. Khan, F. Xhafa, Genetic algorithms for energy-aware scheduling in compu- tational grids, in: Proc. 6th IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC2011), 26-28.10.2011, Barcelona, Spain, 2011, pp.17–24.

9. R. Bolla, R. Bruschi, F. Davoli, F. Cucchietti, Energy E fficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures, IEEE Communications Surveys & Tutorials 13 (2011)223–244.

10. S. N. Roy, Energy logic: a road map to reducing energy consumption in telecom munications networks, in: Proc. 30th International Telecommunication Energy Conference (INTELEC 2008),2008

11. P. Yang, C.Wong, P. Marchal,F.Catthoor,D.Desmet,D. Verkest, and R. Lauwereins. Energy-aware runtime scheduling forembedded-multiprocessor SOCs. *IEEE Design &Test of Computers*, 18(5):46– 58,2001

12. M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpuenergy. In *OSDI '94: Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 2, Berkeley, CA, USA, 1994. USENIXAssociation.

13. K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power cpu. In *MobiCom '95:*

Proceedings of the 1st annual international conference on Mobile computing and networking, pages 13–25, New York, NY, USA, 1995. ACM Press.

14. Y. Zhu and F. Mueller. Feedback edf scheduling of real-time tasks exploitingdynamicvoltagescaling.*Real-TimeSyst.*,31(1-3):33–63, 2005.

15. A. Weisseland F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesisforem-beddedsystems*,pages238–246,NewYork,NY,USA, 2002. ACMPress.

16. P.J.deLangenandB.H.H.Juurlink.Leakage-awaremultiprocessor schedul-ingforlowpower.In*IPDPS*.IEEE,2006.

17. D. Ramanathan, R. Gupta, "System Level Online Power Management Algorithms", *Design, Automation and Test in Europe*, pp. 606–611,2000

18. Y. Lu and G. De Micheli, "Adaptive Hard Disk Power Management on Personal Com- puters", *IEEE Great Lakes Symposium on VLSI*, pp. 50–53, 1999.

19.C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Sav- ing of Event-Driven Computation", in *International Conference on Computer Aided Design*, pp. 28–32, 1997.

20. Intel, Microsoft and Toshiba, "Advanced Configuration and Power Interface specification", available at *http://www.intel.com/ial/powermgm/specs.html*,1996.

21. Q. Qiu, Q. Wu, and M. Pedram,"Stochastic Modeling of a Power-Managed System: Construction and Optimization," Proc. Int'l Symp. Low Power Electronic Devices, pp.194-199,1999.

22. Q. Qiu , Q. Wu, and M. Pedram, "OS-Directed Power Management for Mobile Electronic Systems," Proc. 39th Power Source conf., pp. 506-609,2000.

23. Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Process," *Proc. Design Auto-mation Conf.*, pp. 555-561,1999.

24. L. Benini, A. Bogliolo, G. Paleologo, and G. De Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813-833, June1999.

25. T. Simunic, L. Benini and G. De Micheli, “Energy Efficient Design of Portable Wireless Devices”, *International Symposium on Low Power ElectronicsandDesign*,pp.49–54,2000.

26. T.Simunic,L.BeniniandG.DeMicheli,“DynamicPowerManagementfor Portable Systems”, *The 6th International Conference on Mobile Computing and Networking*, pp. 22–32,2000.

27. T. Simunic, L. Benini and G. De Micheli, “Power Management of Laptop Hard Disk”,*Design, Automation and Test in Europe*, p. 736,2000.

28. P. Rong, and M. Pedram, “Battery-Aware Power Management Based on Markovian Decision Processes,” IEEE Trans. on Computer Aided Design, Vol. 25, No. 7, Jul. 2006, pp.1337-1349.

29. Kan-Jian Zhang, Yan-Kai Xu, Xi Chen and Xi-Ren Cao. Policy iteration based feedback control, in *Automatica*, 44(4): 1055– 1061,2008.

30. Francisco S. Melo, Manuela Veloso. Decentralized MDPs with sparse interactions, in *Artificial Intelligence*, 175(11): 1757– 1789,2011.

31. Roger Brockett. Optimal Control of Observable Continuous Time Markov Chains, in *Proceedings of the 47th IEEE Con- ference on Decision and Control*, Cancun, Mexico, Dec. 9-11,2008.

32. M. Srivastava,A. Chandrakasan. R. Brodersen, “Predictive system shutdown and other architectural techniques for energy efficient programmable computation,” *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 42–55, March1996.

33. *G. Q. Maguire, M. Smith and H. W. Peter Beadle\Smart Badges: a wearable computer and* communication system", 6th International Workshop on Hardware/Software Codesign,1998.

34. Lucent, IEEE 802.11 Wave LAN PC Card – Users Guide,p.A-1.

35. Q. Qiu and M. Pedram, \Dynamic power management based on continuous-time markov decision processes”, Design Automation Conference , pp.55{561,1999.