A Reinforcement Learning Framework For Dynamic Power Management of a Portable, Multi-Camera Traffic Monitoring System

Umair Ali Khan, Bernhard Rinner
Institute of Networked and Embedded Systems
Alpen-Adria Universitat, 9020-Klagenfurt, Austria
umair.khan@aau.at, bernhard.rinner@aau.at

***Abstract*—Dynamic Power Management (DPM) refers to a set of strategies that achieves efficient power consumption by selectively turning off (or reducing the performance of) a system components when they are idle or are serving light workloads. This paper presents a Reinforcement Learning (RL) based DPM technique for a portable, multi-camera traffic monitoring system. We target the computing hardware of the sensing platform which is the major contributor to the entire power consumption. The RL technique used for the DPM of the sensing platform uses a model-free learning algorithm that does not require a priori model of the system. In addition, a robust workload estimator based on an online, Multi-Layer Artificial Neural Network (ML-ANN) is incorporated to the learning algorithm to provide partial information about the workload and to take better decisions according to the changing workload. Based on the estimated workload and a selected power-latency tradeoff parameter, the algorithm learns to use optimal time-out values in sleep and idle modes of the computing hardware. Our results show that the learning algorithm learns an optimal DPM policy for the non-stationary workload, while significantly reducing the power consumption and keeping the system response to a desired level.**

***Keywords*-Dynamic Power Management; Reinforcement Learning; Traffic Monitoring;**

I. INTRODUCTION

Most of the existing traffic monitoring systems use no or limited image processing capabilities and exploit sensors such as induction loops, laser sensors and radars, etc. Additionally, these systems are based on stationary installations where the sensor nodes are permanently mounted at gantries. These fixed-mounted installations are usually expensive to set-up and decrease the flexibility of the monitoring system. During the installation, the road needs to be closed and a substantial calibration effort is also required.

MobiTrick is a portable and compact traffic monitoring system that is aimed at temporary installations. The desired operation of the sensing platform is intended for short-terms (e.g., a few hours, days) and it can be deployed more flexibly for various monitoring tasks, e.g., law enforcement and construction site monitoring. Instead of using large sensors, MobiTrick uses only image processing operations for traffic monitoring. Since the system is portable and compact, its embedded design restricts to use intricate cooling systems and large batteries. Therefore, it has a strict limitation on power consumption and apart from its low-power design, it must use an online power management strategy to minimize the power consumption during operation. This motivates the search for an effective DPM technique for MobiTrick sensing platform.

Relevant literature about DPM demonstrates various approaches. In the simplest approach, the greedy policy [9][8], 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 as long 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 [10][1] 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, specially when the workload (arrival rate of requests) is lower. This problem is better dealt by the predictive policies [2][14][4] which work on a system model that is learned from the history information in order to best adjust themselves to the dynamic system of a device. 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][11][16][13]. 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 [7][19] 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. Another similar work [22] uses the merits of [21] and proposes a RL based DPM algorithm with workload prediction for the power management of a WLAN card. The workload prediction in this work is performed with a Bayes classifier which performs well for this setting due to the regular traffic. Nevertheless, the same workload prediction can not render acceptable accuracy with a vehicle traffic data where the vehicles arrival rates follow a non-stationary pattern.

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 backpropagation 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.

Following the merits of [18], we propose a novel RL based DPM algorithm in this paper for the power management of our sensing platform. 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 pareto-optimal trade-off curve of power-performance and results in a much lower latency while keeping the power consumption at an acceptable level.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of the MobiTrick sensing platform. Section 3 gives some background on reinforcement learning. Section 4 explains the implementation of RL based DPM for MobiTrick. Section 5 presents the overall learning algorithm and the results. We conclude our work with some discussion on future work in Section 6.

II. MobiTrick Sensing Platform

MobiTrick has a heterogeneous setup with different types of sensors, each having different capabilities. The three main components of the sensing platform comprise visual sensors (RGB, grayscale, infrared, D/N), non-visual sensors (inertial measurement units, GPS receiver) and a computing board for image processing. The sensing platform is intended to perform typical traffic monitoring tasks such as vehicle detection and classification, license plate detection, over-height estimation, incident detection. The heterogeneous setup serves many purposes. First, the tasks can be distributed among different sensors (e.g., license plate detection with an infrared camera and context image from a color camera). Second, low-level operations can be performed with less capable and more power-efficient sensors. Third, the redundant information from multiple sensors helps increase reliability.

Figure 1 provides a high-level overview of the MobiTrick sensing platform. The sensing platform has a multi-tier architecture where the sensors reside at different levels based on their energy consumption and capabilities. A smart, low-power, color camera that can run on-board algorithms operates at the lowest level and triggers other cameras at the higher levels at the detection of an event. When triggered, the higher-level cameras send images to the smart camera. The queued images from all the cameras are then sent periodically to the computing board for processing.

The contribution of an efficient DPM policy in this scenario

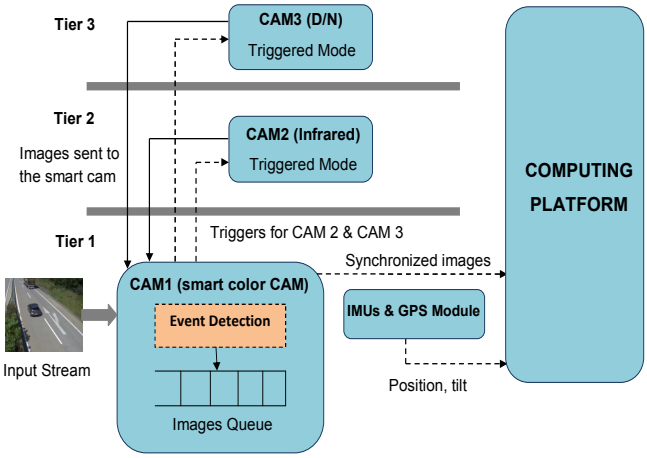

Figure 1. MobiTrick sensing Platform.

is to determine when it is appropriate to wake-up the board and process the images stored in the smart camera. In addition, it must also decide when it is efficient to shutdown the board, while minimizing the power consumption and keeping the system performance at an acceptable level.

III. Reinforcement Learning

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 assigns 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., the next state $s' \in S$ and immediate reward r depend only on the current state $s \in S$ and action $a \in A$, as given by equation 1.

$$P_r\left\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\right\} \tag{1}$$

Where P_r is the probability of reaching state s' and getting reward r at time $t+1$. A policy, π, is a mapping from each state, $s \in S$, and action, $a \in A(s)$ 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 state s and following the policy π thereafter. We can define $V^\pi(s)$ as follows [12]:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t | s_t = s\right\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned} \tag{2}$$

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.

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\left\{R_t | s_t = s, a_t = a\right\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \end{aligned} \tag{3}$$

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 [3] 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.

$$\begin{aligned} &\forall (s,a) \in S \times A : Q(s_t, a_t) = Q(s_t, a_t) \\ &\quad + \alpha_t(s_t, a_t)\left\{r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right\} \end{aligned} \tag{4}$$

Where $\alpha_t(s_t, a_t) \in (0,1)$ 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-term reward, without knowing the state-transition probabilities.

IV. RL Based DPM For MobiTrick

In the MobiTrick sensing platform, we target the DPM of the computing board which is the main source of power consumption. It is an Intel Atom based computing platform [17] whose power and state-transition delay characteristics are given in Table I.

Table I
Power and delays characteristics of the computing board

P_{sleep}	P_{idle}	P_{busy}	P_{trans}	t_{s2i}	t_{i2s}
3W	25W	32W	15W	6s	4s

Where P_{sleep}, P_{idle}, P_{busy} and P_{trans} represent the power consumption in sleep, idle, busy and transition state. t_{s2i} and t_{i2s} represent the time taken to switch from sleep to

idle state and vice versa, respectively. Since the model of the system is pre-characterized, we know how many power modes the system has and how it switches its power mode given a power command. Additionally, we also have partial information about the workload (ML-ANN based estimation). This information is adequate to design a Q-learning algorithm to find an optimal DPM policy, based on a selected power-performance parameter.

In our setup, the RL environment consists of a WorkLoad estimator (WL), a Service Queue (SQ) to buffer the requests before processing, the computing board which works as a Service Provider (SP) and a Power Manager (PM) - the learning agent that issues appropriate power commands to SP. The workload estimator, SQ and PM reside on the smart camera. In this way, the smart camera works as a controller in the sensing platform that issues control signals (triggers, power commands, request receive/send) to other components of the system. We consider three power states of SP, i.e., $sp = \{sleep, idle, busy\}$. Therefore, the available power commands (actions) selected by PM include $a = \{go\_sleep, go\_idle, go\_busy\}$. At each decision epoch, PM receives an observation of the system that include the current state of WL, SQ and SP. Based on this composite state, $S = \{WL, SQ, SP\}$, PM issues a command to SP from the action set. Figure 2 depicts the high-level view of the power management setup.

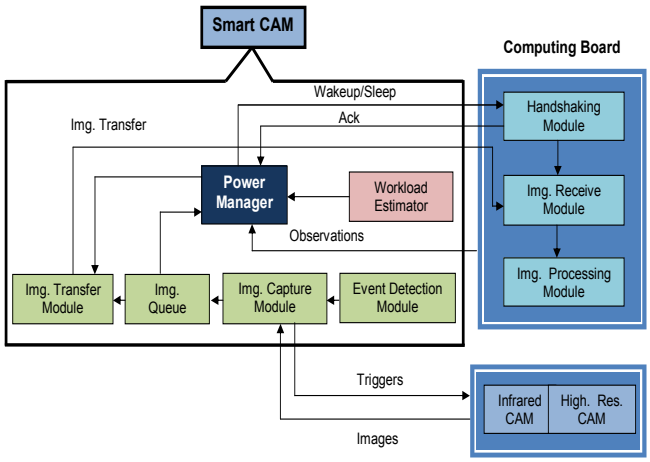

Figure 2. Depiction of the system under power management

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 some requests), the PM regains the control and evaluates the last action by assigning it a cost (Equation 5) and updating the Q-value of the last state-action pair (Equation 9). The PM, then, selects another action in the new state and issues appropriate signal to the SP.

Algorithm 1 RL based time-out policy

Require: Power-performance parameter $\lambda \in (0, 1)$
1. Initialize Q, M and probability matrix p_r arbitrarily.
while Policy not good enough **do**
2. Obtain the current workload estimation (Sec. V.B)
3. Get the current observation: (s, a)
4. Calculate action probabilities: $p_r^k(s, a_k)$ (Sec. V.E)
5. Select an action, a, with probability p_r (Sec. V.E)
6. Execute the selected action
7. Calculate cost of the last action: $c_{t+1}(s, a)$ (Sec. V.A)
8. Update the learning rate: $\alpha_t(s, a)$ (Sec. V.D)
9. Update M with new state-action pair
10. Update Q-value: $Q^{t+1}(s, a)$ (Sec. V.C)
end while
11. Generate policy: $\pi = min_a Q(s, a), \forall s \in S, a \in A$

V. The Learning Elements

This section describes various elements of the learning, including the cost function, workload estimation, state-action pair updating principle, and the action-selection policy.

A. Cost Function

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.

$$c_t(s, a, \lambda) = \lambda \frac{1}{(t_{k+1} - t_k)} \sum_{j=t_k}^{t_{k+1}} P_j + (1 - \lambda) l_t(s, a) \quad (5)$$

In the above expression, $t_{k+1} - t_k$ is the time that the SP remains in state s, and $\lambda \in (0, 1)$ is power-performance trade-off parameter. For $\lambda \to 0$, the learning algorithm gives more importance to latency, thus resulting in a higher power consumption. On the other hand, when $\lambda \to 1$, 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 pareto-optimal trade-off curve.

B. Workload Estimation

In this work, workload (or the request rate) comprises the rate of images captured by all the cameras at the detection of

events (vehicle detection). Therefore, the workload reflects the arrival rate of the vehicles. We used a real workload for the learning algorithm by taking a test recordings on a highway, where we recorded the inter-arrival times of vehicles by a vehicle detection algorithm [6] and acquired 11700 requests (events) that served as the data size for the learning algorithm. The mean inter-arrival times per hour from the 22-hours (2:00 A.M to 12.00 A.M) recordings are depicted in Figure 3.

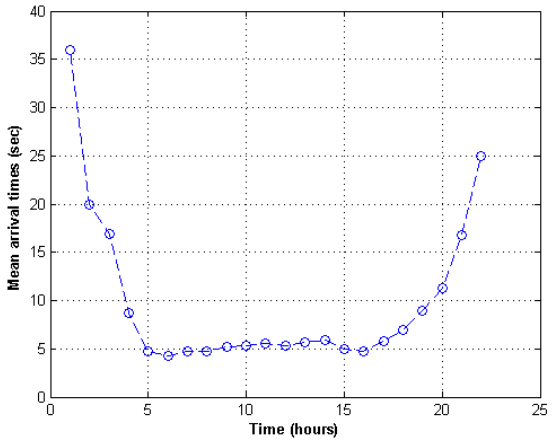

Figure 3. Mean request inter-arrival times per hour

It is evident that the recorded workload exhibits a non-stationary pattern. For workload estimation, we use a fix-sized moving window on the history of previous inter-arrival periods and input these inter-arrival periods to the ML-ANN. The ML-ANN estimates the length of the next inter-arrival period. If the length of the next inter-arrival period is estimated to be longer than a certain threshold T_{thr}, the workload is classified as *low* (or *high* otherwise). Figure 4 shows the 3-layer ANN based workload estimator where v_{ij} denotes the weight of the connection between j^{th} neuron from the input layer and the i^{th} neuron of the hidden layer. Whereas, w_i represents the weight of the connection between i_{th} neuron from the hidden layer and the neuron of the output layer.

The outputs of the input layer, hidden layer and output layer, represented by I_i, h and z are calculated as follows:

$$I_i = \sum_{j=1}^{n} v_{ij}x_j, \quad i = 1, ..., m \tag{6a}$$

$$y_i = f(I_i) \tag{6b}$$

$$h = \sum_{i=1}^{m} w_i y_i \tag{6c}$$

$$z = f(s) \tag{6d}$$

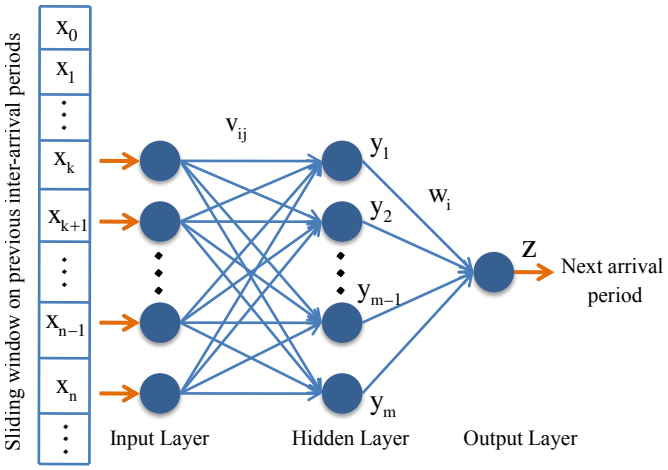

Figure 4. ML-ANN based workload estimator

Where f is the sigmoid function. The weights are adjusted according to the following rule [15].

$$\Delta v_{ij}(t) = \eta(u - z)w_i x_j f^{'}(h) f^{'}(I_i) + \mu \Delta v_{ij}(t-1) \tag{7a}$$

$$\Delta w_i(t) = \eta(u - z)y_i f^{'}(h) + \mu \Delta w_i(t-1) \tag{7b}$$

$$f^{'} = f(1 - f) \tag{7c}$$

Where u is the observed inter-arrival period, η is the learning rate, and μ is a positive constant which is determined experimentally. The experimental size of the moving window was selected to be $n = 8$. The prediction accuracy of the online ML-ANN workload estimator, which is defined as the ratio of correctly predicted intervals to the total number of predicted intervals, is 81.24%.

C. The Learning Principle

In our case, each system state has a composite form, i.e., $S = \{WL, SQ, SP\}$, where $WL = \{0, 1\}$ and $SP = \{sleep, busy, idle\}$. We use state aggregation for limiting the values of SQ to a state having no requests in the queue and the one having some requests, i.e., $\forall sq \in SQ, sq = \{0, N | N \in \mathbb{N}\}$. The action set A for each state comprises time-out values t_{out}^{k} which depend on T_{thr} and are defined as:

$$t_{out}^{i} = \epsilon_i T_{thr}, \quad \epsilon_i \in \mathbb{R}^{+} \tag{8}$$

Where ϵ_i is a positive weight. The learning algorithm in our case looks for minimizing long-term cost, rather than maximizing long-term reward. Hence, for the composite state S, action set $A = \{t_{out}^{1}, t_{out}^{2}, ..., t_{out}^{n}\}$, and $\forall (s, a) \in S \times A$, equation 4 can be modified as follows:

$$\begin{aligned} Q^{(t+1)}(s_t, a_t) =& Q^{t}(s_t, a_t) \\ &+ \alpha_t(s_t, a_t)\{(1 - e^{-\beta t_{out}})c_{(t+1)}(s, a, \lambda) \\ &+ e^{-\beta t_{out}} \min_{a} Q^{(t+1)}(s_{(t+1)}, a) - Q^{t}(s_t, a_t)\} \end{aligned} \tag{9}$$

Where $Q^{(t+1)}(s_t, a_t)$ is the estimated Q-value of state s after taking the action a; $Q^t(s_t, a_t)$ is the Q-value of state s before taking the action a; and $(1-e^{-\beta t_{out}})c_{(t+1)}(s, a, \lambda)$ is the sample discounted cost received by executing a time-out period t_{out}, where $\beta \in (0, 1)$. The term $\min_a Q^{(t+1)}(s_{(t+1)}, a)$ refers to the best (minimum) value in next state $s_{(t+1)}$ corresponding to an action a. The learning algorithm looks for minimizing the difference between the current estimate of a state-action pair and the best discounted value of the next state.

D. Updating Learning Rate

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:

$$\alpha_t(s, a) = \frac{\xi}{visited(s, a)} \tag{10}$$

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.

E. Exploration-Exploitation Policy

It is necessary for a learning algorithm not only exploring the state-space deeper, but also exploiting the experience gained so far. In order to acquire sufficient knowledge about the dynamics of the system, every state-action pair must be visited adequate number of times. We use a semi-greedy policy in this work. This policy starts out with selecting random actions (exploration), each with equal probabilities. As the algorithm proceeds and acquires more knowledge about the system, the probability of actions with minimum cost begins to increase. Eventually, the policy tends to become greedy by selecting actions with heighest probability and minimum cost (exploitation). The action probabilities in a state is given by Equation.

$$p_r^k(s, a) = \frac{e^{Q^t(s,a_k)/T}}{\sum_{k=1}^{n} e^{Q^t(s,a_k)/T}}, \quad \forall a \in A, \; n = |A| \tag{11}$$

Where T is called a temperature coefficient. It is initialized with a high value which gives equal weights (probabilities) to all the actions and hence the policy tends to explore the state-space, giving less importance to exploitation. T is then decayed over time and as the learning progresses the distribution becomes more tight on low-cost actions (equivalent to greedy decisions), and the exploration tendency of the agent reduces.

F. Multiple States Update

Selecting time-out values as actions enables multiple updates in both sleep and idle states in order to accelerate the convergence speed of the algorithm. Consider a case where the SP is in sleep state and the PM selects an action corresponding to a specific time-out value. If no request comes before the time-out expires, all the actions (time-out values) corresponding to smaller time-out values can be evalualted using Equation 9. This is because all the actions with smaller time-out values, if taken, would result in the same cost and the same next state Q-value estimate. Likewise, if the PM selects a time-out value in idle state and a request comes before the time-out expires, all the actions with higher time-out values can be evaluated.

VI. Experimental Results

We varied the power-performance trade-off parameter, λ, between 0 and 1 to produce a pareto-optimal trade-off curve shown in Figure 5. We compared our modified RL based DPM algorithm (Timeout/Timeout model) with the algorithm presented in [18] which uses time-out values in idle state and waits for N number of requests in sleep state (Timeout/N model).

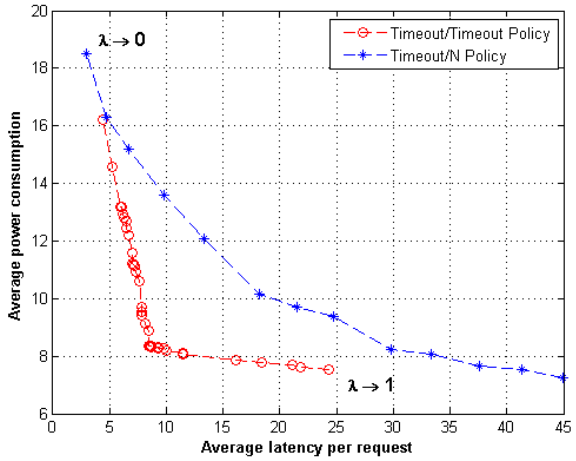

Figure 5. Power-performance pareto optimal trade-off curve

Figure 5 shows that RL based Timeout/Timeout model achieves aggresive power saving and produces a much deeper trade-off curve, while significantly reducing the average latency per request as compared to Timeout/N model. Selecting time-out values in sleep state has a direct impact on the time-out values in idle state. The optimal selection of time-out values in sleep state helps selecting optimal time-out values in idle state as well (and vice versa), thus avoiding the algorithm to waste energy by executing higher time-out values in idle state when the workload is lower. In Timeout/Timeout model, we can ensure the range of latency for the requests, which is given as follows:

$$[l_{min}, \; l_{max}] = [tp, \; (t_{i2s}, t_{sleep}, t_{s2i}, t_p)] \tag{12}$$

Where the minimum latency, l_{min}, can be as small as the processing time, t_p, of the request. On the other hand,

suppose that PM issues a sleep command to the SP and a request arrives at the same time. The maximum latency, l_{max}, in this case can be as high as the sum of transition time from idle to sleep, t_{i2s}, time-out period in sleep state, t_{sleep}, transition time from sleep to idle, t_{s2i}, and the processing time of the request.

The probability distribution of the actions (time-out values) in each state is shown in Figure 6. The two modes $SP = \{0, 1\}$ represent the sleep and idle state respectively. The actions $sleep \rightarrow busy$ and $busy \rightarrow idle$ are assumed to be autonomous and hence can be excluded from the action set. Figure 6 shows the probability distribution of action set for small value of the power-performance parameter ($\lambda = 0.01$), where the learning algorithm is very sensitive to the average latency per request and hence the probability of smaller time-out values (based on the estimated workload) in sleep state is high. In contrast, the probabilities of larger time-out values (based on the estimated workload) in idle state are high. The grayed cells shows the best actions with the highest probablities in each state that will be consistently selected when the policy becomes greedy.

Time-out values (Increasing order)	**States *S=(WL, SQ, SP)***					
	(0,0,0)	**(0,N,0)**	**(1,0,0)**	**(1,N,0)**	**(0,0,1)**	**(1,0,1)**
t_{out}^{1}	0.30456	0.39329	0	0.50833	0.02705	0.02684
t_{out}^{2}	0.32312	0.28466	0	0.19102	0.05229	0.04396
t_{out}^{3}	0.18683	0.10155	0	0.14059	0.07385	0.06309
t_{out}^{4}	0.12798	0.10831	0	0.14471	0.09599	0.08582
t_{out}^{5}	0.03588	0.10548	0	0.01383	0.10749	0.12285
t_{out}^{6}	0.01972	0.00269	0	0.00115	0.11379	0.11291
t_{out}^{7}	0.00175	0.00358	0	0.00027	0.13275	0.13867
t_{out}^{8}	0.00016	0.00041	0	0	0.13392	0.13894
t_{out}^{9}	0	0	0	0	0.12609	0.12986
t_{out}^{10}	0	0	0	0	0.13677	0.13706

Figure 6. Probability distribution of actions in each state

From the above figure, it is also evident that the state $(1, 0, 0)$ is never visited. This can be explained as follows. Suppose that the workload is estimated to be high and the SP has just entered idle state after processing all the requests in the queue. This represents a composite state $s = (1, 0, 1)$. The PM now selects and execute a time-out period based on the workload estimation. Since the workload is estimated to be high and the time-out period is selected accordingly, it is highly likely that some requests will arrive before the time-out period expires. Even if no requests arrive during the time-out period and the PM issues sleep command to SP, it is highly likely that some requests will arrive during the transition period, leaving the SP either in state $s = (1, N, 0)$ or in state $s = (0, N, 0)$. Likewise, if the SP is in state $s = (0, 0, 0)$ and the workload becomes high, the queue can no longer be empty, again leading to state $s = (1, N, 0)$. This also ensures the reliability of the workload estimator. However, it is still safe to keep the state $(1, 0, 0)$ in state-space, lest it may be visited due to some random action or in a different workload dataset.

We also analyzed the impact of using different threshold periods T_{thr} on the learning algorithm. This also changes the action set of time-out values which are function of T_{thr}. Figure 7 shows a comparison of power-performance curves for different values of T_{thr}. This comparison shows that the power-performance curve does not change significantly by changing the threshold period T_{thr} and hence the learning algorithm can adopt to different threshold periods. This is because selecting a smaller value of T_{thr} does allow the algorithm choosing (relatively) smaller time-out values in the sleep state resulting in decreasing the average latency. However, it also compels the algorithm to select (relatively) smaller time-out values in idle state as well in order to compensate the (increased) power consumption. As a result, the average latency and the power consumption do not vary drastically for different values of T_{thr}. Instead of using a static value of T_{thr}, it may be adjusted over time by the mean of last n inter-arrival periods.

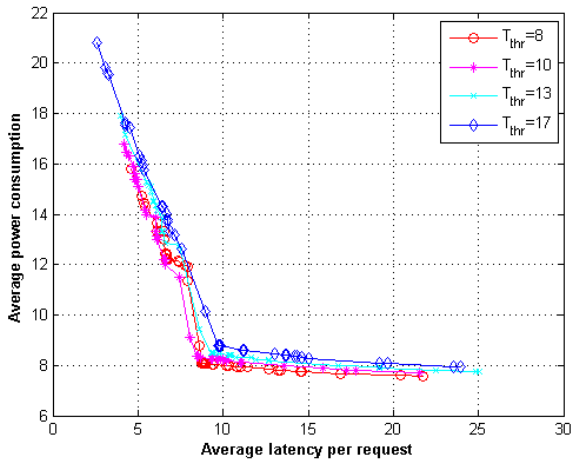

Figure 7. Comparison of power-performance curves for different threshold periods

It is worth mentioning that the value of T_{thr} must be bounded for a given set of weights ϵ_i to avoid divergence of the algorithm or to prevent a local minima. Moreover, selecting very small values of weights-threshold combination results in smaller time-out values in action set and hence increases the average power consumption due to frequent transitions among the states. Likewise, selecting very high values of weights-threshold combination can also mitigate the power saving achieved in sleep state (with higher time-out values) by spending more time in idle state, executing higher time-out values and hence wasting energy.

For a given set of weights ϵ_i, the boundary values for T_{thr} can be determined experimentally based on a specific workload. For our experimental workload, the acceptable range of T_{thr} is between 5 and 20 for the set of weights $\epsilon_i = \{0.1, 0.2, ..., 1, 1.1, ...\}$.

VII. Conclusion and Future Work

In this paper, we presented a RL based DPM approach for our stereo-vision based, multi-camera traffic monitoring system. The presented approach does not require a priori model of the system (the transition probabilities of individual components), as against to the existing stochastic DPM approaches. The algorithm learns the dynamics of the system by interacting with it, implementing certain actions during learning and evaluating their effects. Additionally, our proposed approach augments the good features of the traditional DPM approaches, i.e., time-out, greedy, predictive and stochastic, into a single learning scheme. The algorithm finds the optimal time-out values both in sleep and idle mode and tries to shutdown the computing board as soon as possible (greedy). Likewise, the algorithm also uses a predictive approach for workload prediction. Besides, the learning algorithm uses the same elements as used in stochastic approaches (queuing model, etc). Our results show that the proposed Timeout/Timeout algorithm provides a decent and much deeper power-latency trade-off curve as compared to Timeout/N algorithm.

In future, we aim to formulate the DPM optimization problem as a dual problem, i.e., minimizing power consumption (or latency) for a given performance (or power) constraint. The RL based learning algorithm can then be implemented to find the right value of the power-performance constraint (λ) that exactly meets the power (or performance) constraint.

Acknowledgement

This work has been sponsored in part by the Austrian Research Promotion Agency under grant 825840.

References

[1] A. Karlin, M. Manasse, L. McGeoch and S. Owickim. Competitive randomized algorithms for non-uniform problems. *Algorithmica*, 11(6):542–571, 1994.

[2] C. H. Hwang, A. C. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *International Conerence on Computer Aided Design*, 1997.

[3] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, 1989.

[4] E. -Y. Chung, L. Benini, G. De Micheli. Dynamic power management using adaptive learning tree. In *International Conference on Computer Aided Design*, pages 274–279, 1999.

[5] F. Douglis, P. Krishnan, B. Bershad. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8:381–413, 1995.

[6] F. Pletzer, R. Tusch, B. Rinner, L. Böszörmenyi. Robust traffic state estimation for smart cameras. In *Proceedings of 9th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS), Beijing, China*, 2012.

[7] G. Dhiman, T. Rosing. Dynamic power management using machine learning. In *IEEE/ACM International Conference on Computer-Aided Design*, 2006.

[8] J. M. Pedram. *Power Aware Design Methodologies*. Kluwer Academic, 2002.

[9] L. Benini, A. Bogliolo, G. A. Paleolog, G. de Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18:813–833, 1999.

[10] P. Greenawalt. Modeling power management for hard disks. In *International Workshop on Modeling, Analysis, and Simulation for Computer and Telecomm. Systems*, pages 62–65, 1994.

[11] Q. Qiu, M. Pedram. Dynamic power management based on continuous-time markov decision process. In *Design Automation Conference*, pages 555–561, 1999.

[12] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[13] S. Shukla, R. Gupta. A model checking approach to evaluating system level dynamic power management policies for embedded systems. In *High-Level Design Validation and Test Workshop*, pages 53–57, 2001.

[14] Srivasta et al. Predictive system shutdown and other architecture techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems*, 4:42–55, 1996.

[15] T. Phit, K. Abe. Packet inter-arrival time estimation using neural network models. In *Internet Conference*, 2006.

[16] T. Simunic, L. Benini, G. De Micheli. Event-driven power management of portable systems. In *International Symp. on System Synthesis*, pages 18–23, 1999.

[17] U. A. Khan, M. Quaritsch, B. Rinner. Design of a heterogeneous, energy-aware, stereo-vision based sensing platform for traffic surveillance. In *9th Workshop on Intelligent Solutions in Embedded Systems*, 2011.

[18] U. Khan, B. Rinner. Dynamic power management for portable, multi-camera traffic monitoring. In *18th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2012.

[19] V. Lakshmi, E. Monie. Hardware architecture of reinforcement learning scheme for dynamic power management in embedded systems. *EURASIP Journal on Embedded Systems*, 2007:645–650, 2007.

[20] Y. -H. Lu, G. De Micheli. Adaptive hard disk power management on personal computers. In *Great Lakes Symp. VLSI*, pages 50–53, 1999.

[21] Y. Tan, W. Liu, Q. Qiu. Adaptive power management using reinforcement learning. In *EEE/ACM International Conference on Computer-Aided Design*, 2009.

[22] Y. Wang, Q. Xie, A. Ammari, M. Pedram. Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. In *48th Design Automation Conference*, pages 41–46, 2011.