Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2012, Article ID 870281, 20 pages
doi:10.1155/2012/870281

Research Article

Effective Stochastic Modeling of Energy-Constrained Wireless Sensor Networks

Ali Shareef and Yifeng Zhu

Department of Electrical and Computer Engineering, University of Maine, Orono, ME 04469, USA

Correspondence should be addressed to Yifeng Zhu, yifeng.zhu@maine.edu

Received 16 June 2012; Accepted 11 September 2012

Academic Editor: Runhua Chen

Energy consumption of energy-constrained nodes in wireless sensor networks (WSNs) is a fatal weakness of these networks. Since these nodes usually operate on batteries, the maximum utility of the network is dependent upon the optimal energy usage of these nodes. However, new emerging optimal energy consumption algorithms, protocols, and system designs require an evaluation platform. This necessitates modeling techniques that can quickly and accurately evaluate their behavior and identify strengths and weakness. We propose Petri nets as this ideal platform. We demonstrate Petri net models of wireless sensor nodes that incorporate the complex interactions between the processing and communication components of an WSN. These models include the use of both an open and closed workload generators. Experimental results and analysis show that the use of Petri nets is more accurate than the use of Markov models and programmed simulations. Furthermore, Petri net models are extremely easier to construct and test than either. This paper demonstrates that Petri net models provide an effective platform for studying emerging energy-saving strategies in WSNs.

1. Introduction and Motivations

Application for wireless sensor networks (WSNs) has abounded since their introduction in early 2000. WSNs are being used from surveillance, environmental monitoring, inventory tracking, and localization. A sensor network typically comprises of individual nodes operating with some limited computation and communication capabilities, and powered by batteries with limited energy supply. Furthermore, these networks are situated at a location where they may not be easily accessible. Their distributed nature, small footprint, cheap, and wireless characteristics make them very attractive for these outdoor, unattended, and hostile environment applications.

One of the motivating visions of WSNs was large-scale remote sensing such as large areas of a rainforest for environmental parameters such as humidity and temperature. However, given the remoteness of such a site, this can be a challenging problem. Modern WSNs were proposed for solving such problems, and it was envisioned that these WSN nodes could be sprinkled over an area from the back of an airplane as it flew over such an area. The nodes wherever they fell would automatically set up an ad hoc network, collect the necessary sensory information, and route the information to a base-station. Although great strides have been made in WSN designs and implementation, we are nowhere near meeting this original motivating problem.

One reason why this original problem has been difficult to solve is that WSNs are still relatively expensive in large quantities. However, the much larger problem is the limited energy available on these devices. The utility of WSNs is limited to the life of the battery under the energy consumption rates. While energy harvesting in WSNs is an active research area [1], generally this is not feasible yet for entirely sourcing the energy needs of an WSN. WSNs are still very much bound to batteries.

An avenue for mitigating this energy dilemma is through the design of energy-efficient communication and active/sleep scheduling algorithms. In this way, a vital resource can be rationed to last a much longer time. This minimizes the overall maintenance and replacement costs of a WSN network. However, proposing energy-efficient designs requires

a detailed understanding of the energy consumption behavior of the nodes which comprise WSNs. This detailed understanding arises from accurate implementation of models for these nodes and analyzing these models under a variety of different states.

For example, many processors are available today that are capable of moving to a sleep mode where they consume minimal energy. However, should a processor be put to sleep immediately after computation, or after some time has elapsed? Or even, perhaps it should never be put to sleep? If it is best to move the processor to sleep after a time delay, what should this delay or *Power Down Threshold* be for a given system? Keep in mind, there is a high energy cost associated with waking up a processor from a sleep mode due to the internal capacitances. If the threshold is too short, then the CPU goes to sleep more often, and there is a stiff price to be paid each time the CPU needs to be woken up. If the threshold is too long, the CPU idles consuming energy wastefully. Nevertheless, there is an optimum threshold that results in the least amount of energy consumption that strikes an optimum balance between putting the CPU to sleep and maintaining it in an active mode. This threshold can also be referred to as break-even time [2].

Another example of emerging technology that can be exploited in wireless sensor networks is the use of processors that have dynamic voltage scaling capabilities. In these processors, the voltage and clock frequency can be dynamically adjusted to obtain a minimum clock frequency to complete the task while using minimal energy [3, 4]. Currently the two types of DVS systems available are those that stop execution while changing voltage and frequency and those that are capable of changing its operating parameters at run time [5, 6]. However, in order to begin investigating energy optimization techniques, such as answering the questions given earlier, we need to devise models that can be used to accurately compute the energy consumption of a wireless sensor node. This need motivates the research presented in this paper.

Currently, there are two classes of modeling and simulating techniques: stochastic and simulation-based methods. Each has its strengths and weaknesses. We propose another method of modeling that has not been used in the past to model WSNs: Petri nets. This paper studies two methods of energy modeling: Markov chains, and Petri nets [7, 8]. These modeling techniques will be compared against a programmed simulation model. This paper makes the following contributions.

(i) We successfully model a processor using a Markov model based on supplementary variables; we also model a processor using colored Petri nets. These models are capable of estimating the average energy consumption of a processor that can power down to a sleep mode. While Markov models have long been used for modeling systems, we show that for estimating CPU energy consumption, the Petri net is more flexible and accurate than the Markov model.

(ii) Using Petri nets, we develop a model of a wireless sensor node that can accurately estimate the energy consumption. We successfully apply this model to identify the optimum powering down threshold for a given wireless sensor network application.

(iii) Our model of a sensor node based on Petri net can be utilized to construct a wireless sensor network. This provides a platform that can be used to study energy consumption at the network layer for application such as cross-layer energy-aware routing.

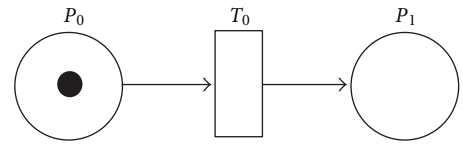


Figure 1: Example of Petri net.

The paper is organized as follows. Section 2 gives a short introduction to Markov models and Petri nets and discusses related work. Section 3 presents the CPU energy models and Section 4 validates the CPU models. Section 6 presents the model for a wireless sensor node and Section 4 uses this model to study the energy optimal *Power Down Threshold*. Section 8 introduces the use of Petri nets in modeling wireless sensor networks. Section 9 concludes this paper.

2. Background and Related Work

2.1. Introduction to Markov Models and Petri Nets. Traditionally, Markov models have been used; however, they are very restrictive in the type of behaviors that can be modeled. A Markov model is a discrete-time stochastic process composed of event chains with Markov property, meaning that the next state in a Markov model is only dependent upon the current state instead of previous ones.

The advantage of using Markov chains for modeling systems is that once the appropriate equations are derived, the average behavior can be easily obtained by evaluating the appropriate equations. However, the task of obtaining the equations relevant to the system can be time-consuming, if not impossible.

Petri nets, on the other hand, are very powerful tools that can be utilized to build statistical models of complicated systems that would otherwise be very difficult. Petri nets is a simulation-based approach for modeling. A Petri net is a directed graph of nodes with arcs. Nodes are referred to as places and are connected to transitions with arcs. In this paper, arcs will be drawn as directed arrows.

An example of a simple Petri net is shown in Figure 1 that contains two places P_1 and P_0, and a transition T_0. The input place of T_0 is P_0, and the output place of T_0 is P_1. T_0 is enabled only if P_0 contains as many tokens as specified by the arc. In this example, T_0 is enabled because it requires only one token in P_0. Once a transition is enabled, it will fire according to a specified timing parameter. During the process of firing, a transition will remove a number of tokens, as specified by the arc, from the input place and deposit these tokens in the output place. If an immediate transition is used, the transition will fire as soon as it is enabled. Deterministic transitions fire if some predetermined time has passed after

it has been enabled, and exponential transitions fire if some random time in some time interval has passed. A Petri net can be used to model the behavior of a system utilizing this flow of tokens to represent movement of control through the different processes of the system. Statistical analysis of the number of tokens in a specific place or the number of particular transitions can provide insights into the behavior of the modeled system. Many software packages are available that can be used to design and perform analysis upon Petri nets. The software that will be used in this study is TimeNET 4.0 [9].

TimeNET 4.0 [9], written in JAVA, is a discrete event simulator of Petri nets. A standalone GUI program called Platform Independent Editor for Net Graphs (PENG) is used to build the Petri net model. The model is translated to a XML schema file which describes all the places, transitions, tokens, and arcs. Thereafter, an executable is generated to simulate the model. Depending on the model implemented in TimeNET, different solution techniques are applied. For example, TimeNET can compute the steady-state solutions for Petri nets with mutually exclusive nonexponential firing parameters. However, for models that result in more than one deterministic transition becoming enabled at a time, appropriate techniques are applied to maintain statistical steady-state accuracy as the model is simulated over an extended time.

2.2. Related Work. Many techniques have been proposed for modeling embedded systems and minimizing the energy consumption. As discussed earlier, existing method includes stochastic Markov models and programmed simulation methods. There are many proposed methods based on Markov models [1, 10]. In [11], Markov models are used to model active and sleep capabilities of a node and the resulting energy and performance characteristics. Steady-state probability equations are derived that describe the number of packets that must be serviced in different modes of operation.

Another proposed method [12] employs a stochastic queueing model to develop a cross-layer framework. This framework is utilized to find the distribution of energy consumption for nodes for a given time period. When the time period is long, it is demonstrated that the distribution approaches a normal distribution. This information is then used to predict node and network lifetime. This framework is also used to identify relationships between energy consumption and network characteristics such as network density, duty cycle, and traffic throughput.

Jung et al. in [13] proposed the use of Markov models to model nodes in a wireless sensor network. However, Jung's focus was on identifying the power consumption rates between trigger-driven and schedule-driven modes of operation and, as a result, the lifetimes of node using these methods. We feel that the use of Markov models is cumbersome and results in limitation of the model due to the inability of Markov models to account for fixed constant arrival or service rates.

Coleri et al. [14] have demonstrated the use of a Hybrid Automata to model TinyOS and hence the resulting power consumption of the nodes. Coleri was able to analyze the nodes in the network on a much wider scale than what is presented in this paper. By utilizing the TinyOS framework, an Automata model was constructed that resulted in the ability to analyze power dissipation of a node based on its location in the sensor network. Finite Automata have also been used in [15].

Liu and Chou [2] present a model based on tasks, constraints, and schedules. Energy minimization is proposed through the use of scheduling for DVS processors capable of executing at different modes. Other works include [16, 17].

One of most common methods of modeling is through the use of programmed simulation using tools such as NS2, OMNet++, OPNET, and TOSSIM. Each of these tools have their strengths and weakness as described in [18].

In [19], the author propose the use of Petri nets for modeling the behavior and characteristics of WSN nodes in what they define as intelligent wireless sensor networks (IWSNs). Their Petri net models can be used to simulate the actual applications, and they present results of a target tracking system prototype that they implement using a Petri net tool called integrated net analyzer (INA).

We have not found any literature that discusses the use of Petri nets for modeling energy consumption of nodes in a wireless sensor network [7, 8]. We attempt to view the minimization of energy from a systems standpoint rather than just focus on the CPU. Because the CPU is intricately associated with the system, all the other parameters of the system affect the energy consumption associated with the CPU.

3. Evaluation of a CPU with a Markov Model and Petri Net

Intrinsically, embedded systems operating in a wireless sensor network offer great potential for power minimization. Generally, the level of computation required is low and usually interspersed with communication between other nodes in the network. The power consumption of the CPU can be minimized by moving to a low power mode and conserving energy when it is not directly involved in any computation.

3.1. Open versus Close Workload Model. There are two types of workload generators that are widely used for generating jobs for a simulation. Both are used quite frequently depending upon the application. One is called the closed workload generator and the other is called the open workload generator. In a closed workload generator, a new job cannot be generated until the system has completed servicing the current job. This can be used to model schedule-driven systems that poll at given intervals for task requests. Since jobs are not generated until the current job is processed, this workload model is easy to implement and analyze. In open workload generators, on the other hand, jobs arrive independent of the state of the system. These can be used to model trigger-driven systems that service requests when an interrupt occurs. However, since jobs can arrive at any time, a buffer needs to be implemented to store those requests that arrive while the system is busy with another request. This workload model can

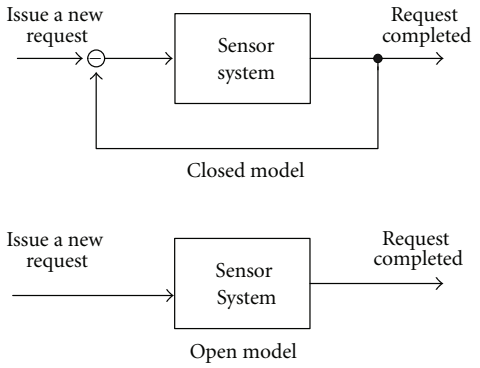


Figure 2: Closed workload generator model and open workload generator model.

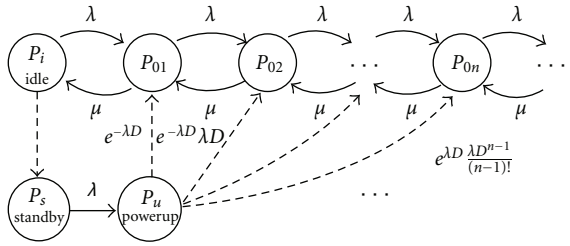


Figure 3: Birth-death process of CPU jobs.

be more difficult to implement and analyze. Figure 2 depicts both closed and open workload generators.

3.2. Modeling Energy Consumption of a CPU Using a Markov Model. An example of a Markov model of a CPU is given in Figure 3. The CPU "power-ups" (p_u) from some low power "standby" mode (p_s) when jobs begin arriving. The Markov model depicts the various increasing states (p_{01}, p_{02}, p_{03}, etc.) the CPU enters as the number of jobs increase under a given job arrival rate λ. The CPU services the jobs at rate μ and strives to move the CPU to lower states and eventually to the "idle state" (p_i). If the processor remains in the idle state for some time interval greater than some threshold, the CPU moves back to the "standby" (p_s) state.

In this example, the following assumptions are made.

(1) The request arrivals follow a Poisson process with mean rate λ.

(2) The service time is exponentially distributed with mean $1/\mu$.

(3) The CPU enters the standby mode (state p_s) if there are no more jobs to be serviced for a time interval longer than T.

(4) The power up process (state p_u) takes a constant time D.

The CPU model consists of a mix of deterministic and exponential transitions. While all transitions shown as solid lines in Figure 3 follow exponential time distributions, the transitions shown as dashed lines are deterministic. This includes the transition from the idle state to the standby state. The CPU enters the standby state after idling for a constant time threshold T. This power down transition depends on its history and is not memoryless. Accordingly, the CPU transitions cannot be modeled directly as a Markov process. Using the method of supplementary variables proposed in [20], we can derive an alterative set of state equations to approximate the transitions for stationary analysis. Let $X = [x_1, x_2]$ denote two age variables representing how long a deterministic transition has become enabled [21]. And let $\mathbf{P}_i(x_1)$ and $\mathbf{P}_u(x_2)$ be the age density functions with respect to x_1 in the idle state and with respect to x_2 in the power up state, respectively.

The state equations for this mixed transition process can be derived by the inclusion of two supplementary variables X. The deterministic transition from the idle state to the standby state can be modeled as below:

$$\begin{aligned} p_i &= \int_0^T \mathbf{P}_i(x_1)dx_1, \\ \frac{d}{dx_1}\mathbf{P}_i(x_1) &= -\lambda \mathbf{P}_i(x_1), \\ \mathbf{P}_i(0) &= \mu p_{01}, \\ \mathbf{P}_i(T) &= \lambda p_s, \end{aligned} \tag{1}$$

where $\mathbf{P}_i$ is an exponential function with coefficient λ (p_i is the steady-state probability of being in the idle state).

The deterministic power up process can be modeled as below:

$$\begin{aligned} p_u &= \int_0^D \mathbf{P}_u\,(x_2)dx_2, \\ \frac{d}{dx_2}\mathbf{P}_u(x_2) &= -\lambda \mathbf{P}_u(x_2), \end{aligned} \tag{2}$$

$$\mathbf{P}_u(0) = \lambda p_s. \tag{3}$$

In addition, when the system is stable, we have

$$(\lambda+\mu)\,p_{01} = \lambda p_i + \mu p_{02} + e^{-\lambda D}\mathbf{P}_u(0), \tag{4}$$

$$\begin{aligned} (\lambda+\mu)\,p_{0n} &= \lambda p_{0,n-1} + \mu p_{0,n+1} \\ &\quad + e^{-\lambda D}\frac{(\lambda D)^{n-1}}{(n-1)!}\mathbf{P}_u(0) \quad \text{for } n \geq 2, \end{aligned} \tag{5}$$

$$1 = \sum_{n=1}^{\infty} p_{0,n} + p_i + p_s + p_u. \tag{6}$$

From (1), we can get

$$\begin{aligned} p_{01} &= \frac{\lambda}{\mu}e^{\lambda T}p_s, \\ p_i &= \left(e^{\lambda T} - 1\right)p_s. \end{aligned} \tag{7}$$

From (2), and (3), we have

$$p_u = \left(1 - e^{-\lambda D}\right)p_s. \tag{8}$$

We define the generating function of $p_{0,n}$ $(n = 1, 2, \ldots)$ as

$$G_0(z) = \sum_{n=1}^{\infty} p_{0,n} z^n. \tag{9}$$

We multiply (4) by z and (5) by z^n, add from $n = 1$ to ∞, use (3), (7), and (9), and get

$$G_0(z) = \frac{\lambda z p_s}{\mu - \lambda z}\left(e^{\lambda T} + \frac{e^{\lambda D(z-1)} - 1}{z - 1} z\right). \tag{10}$$

Thus, we get

$$G_0(1) = \frac{\lambda p_s}{\mu - \lambda}\left(e^{\lambda T} + \lambda D\right). \tag{11}$$

Substituting (7), (8), (9), and (11) into the normalization equation (6) gives

$$p_s = \frac{1 - \rho}{e^{\lambda T} + (1 - \rho)(1 - e^{-\lambda D}) + \rho \lambda D}, \tag{12}$$

where $\rho = \lambda/\mu$.

Consequently,

$$\begin{aligned} p_i &= \frac{(1 - \rho)\left(e^{\lambda T} - 1\right)}{e^{\lambda T} + (1 - \rho)(1 - e^{-\lambda D}) + \rho \lambda D}, \\ p_u &= \frac{(1 - \rho)\left(1 - e^{-\lambda D}\right)}{e^{\lambda T} + (1 - \rho)(1 - e^{-\lambda D}) + \rho \lambda D}. \end{aligned} \tag{13}$$

And the utilization is

$$G_0(1) = \frac{\rho\left(e^{\lambda T} + \lambda D\right)}{e^{\lambda T} + (1 - \rho)(1 - e^{-\lambda D}) + \rho \lambda D}. \tag{14}$$

Let $L(z) = \sum_{n=1}^{\infty} n p_{0n}\, z^n$, then $L(1)$ is the total number of jobs in the system:

$$\begin{aligned} L(z) &= \sum_{n=1}^{\infty} n p_{0n}\, z^n \\ &= z \frac{d}{dz}\left(\sum_{n=1}^{\infty} P_{0n}\, z^n\right) \\ &= z \frac{d}{dz} G_0(z). \end{aligned} \tag{15}$$

Incorporating (10) into (15), we can get the total number of jobs in the system as follows:

$$L(1) = \frac{\rho}{1 - \rho} \frac{e^{\lambda T} + (1/2)(1 - \rho)\lambda^2 D^2 + (2 - \rho)\lambda D}{e^{\lambda T} + (1 - \rho)(1 - e^{-\lambda D}) + \rho \lambda D}. \tag{16}$$

According to the Little's Law, the average latency for each job is

$$\tau = \frac{L(1)}{\lambda}. \tag{17}$$

Thus, the total running time is

$$\begin{aligned} T &= \frac{N}{\lambda} + L(1)\tau \\ &= \frac{N + L(1)^2}{\lambda}, \end{aligned} \tag{18}$$

where N is the total number of jobs.

And the total energy consumption is

$$\begin{aligned} E = &\left(p_i P_{idle} + p_s P_{standby} + p_u P_{powerup} + G_0(1) P_{active}\right) \\ &\times \frac{N + L(1)^2}{\lambda}, \end{aligned} \tag{19}$$

where P_{idle}, $P_{standby}$, $P_{powerup}$, and P_{active} are the power consumption rate in the idle, standby, power up, and active states, respectively; p_i, $p_s p_u$, and $G_0(1)$ are the probability that the system stays in the corresponding state.

3.3. CPU Energy Modeling Using a Petri Net. As shown in the last section, the development of a Markov model for even a simple CPU is mathematically cumbersome especially when dealing with deterministic transitions. Any slight modifications to the model will entail that the equations be rederived again. Petri net on the other hand offers a more flexible approach.

Figure 4 shows an open model of an extended deterministic and stochastic Petri net (EDSPN) [22] modeling a minimizing power consumption system for a processor like the Markov model described earlier. The Petri net models a CPU that starts from some "stand by" state (*Stand_By*) and moves to an "on" state (*CPU_ON*) when jobs are generated. The CPU remains in the "on" state so long as there are jobs in the CPU buffer. If there are no jobs in the CPU buffer for some time interval as given by *Power_Down_Delay*, the CPU then moves to the "stand by" mode (*Stand_By*) to conserve power.

This model uses an open workload generator because when transition $T1$ fires to deposit a task in the *CPU_Buffer*, a token is moved back to place $P0$ which enables transition *Arrival_Rate* and allows another task to be generated. Table 1 lists the parameters of all the transitions in the Petri net. The names of the transitions in Figure 4 are listed in the first column, followed by the type of transition. Transitions that have a specified time parameter are listed in the "Delay" column. The last column indicates the priority of a transition in the event that there is a tie. Transitions with higher priority fire before other transitions.

The CPU is simulated by executing the Petri net using the following steps.

(1) Jobs are generated in place $P1$, when transition *Arrival_Rate* fires randomly in the interval $[0, 1]$ using an exponential distribution. A token in the place $P0$ is moved to $P1$ and enables $T1$.

(2) Transition $T1$ is an immediate transition and fires as soon as it is enabled. Also since $T1$ has the highest priority, it will fire before any other immediate transition if multiple immediate transitions are enabled

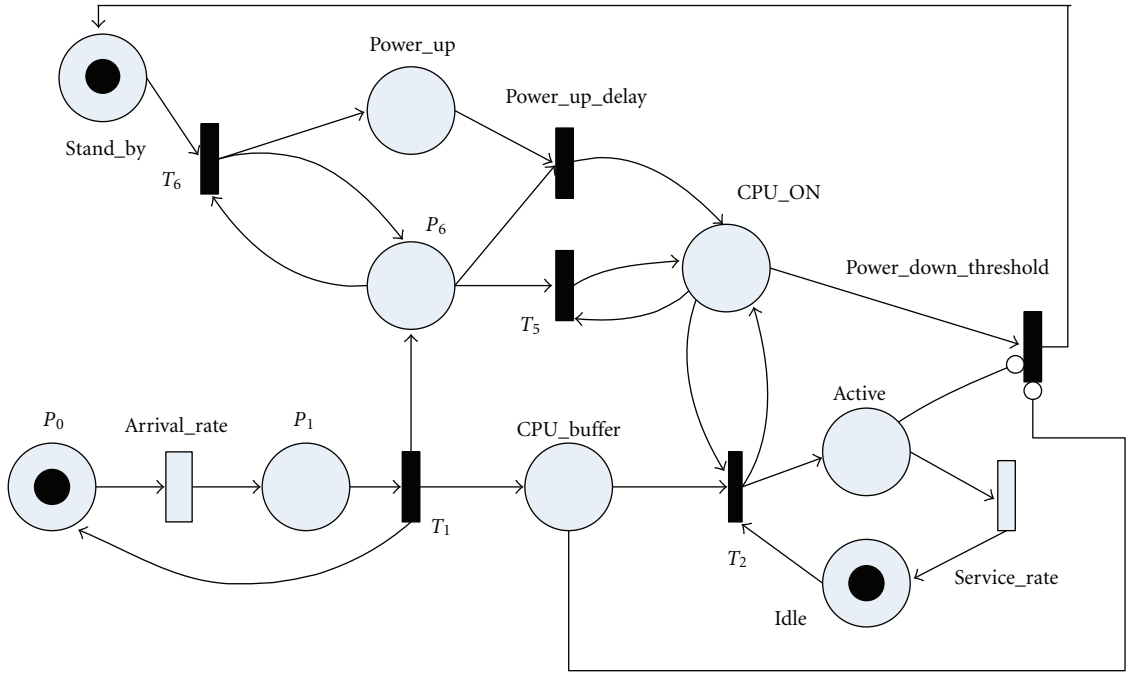


Figure 4: Petri net model of CPU.

Table 1: Petri net transition parameters for CPU jobs.

Transition	Firing distribution	Delay	Priority
AR	Exponential	ArrivalRate	NA
*T*1	Instantaneous	—	4
*T*2	Instantaneous	—	1
SR	Exponential	ServiceRate	NA
PDT	Deterministic	*PDD*	NA
*T*5	Instantaneous	—	2
*T*6	Instantaneous	—	3
PUD	Deterministic	*PUD*	NA

concurrently. When *T*1 fires, a token is removed from *P*1 and three tokens are generated and deposited in places *P*0, *P*6, and *CPU_Buffer*, respectively. Initially, CPU is in the *Stand_By* mode. When a job arrives, a token is deposited in place *P*6 and transition *T*6 is enabled.

(3) When *T*6 fires, one token from *Stand_By* and *P*6 are removed, respectively. Two new tokens are then generated, with one deposited in place *Power_Up* and the other in *P*6. The CPU has now moved to a powering up state. Transition *Power_Up_Delay* is now enabled with a token in *Power_Up* and in *P*6.

(4) Since transition *Power_Up_Delay* has a deterministic delay, the transition fires after a fixed time interval. When this happens, the two tokens from *Power_Up* and *P*6 are removed, and a token is deposited in place *CPU_ON*. The CPU is now "on" and ready to process job events.

(5) When *T*1 fires, a token is placed in *CPU_Buffer* in step 2. A token in *CPU_Buffer*, a token in *CPU_ON*, and a token in *Idle* enable the immediate transition *T*2. When *T*2 fires, one token, respectively from *CPU_Buffer*, *CPU_ON*, and *Idle* are removed and two new tokens are generated, with one deposited back in *CPU_ON* and the other in *Active*. The system is now in the processing state. With a token in *Active*, the transition *Service_Rate* is enabled.

(6) *Service_Rate* is an exponential delay and it will fire randomly after some time in the interval $[0, 0.1]$. After *Service_Rate* fires, the token is removed from *Active* and placed in *Idle*.

(7) In the event that another task arrives while the CPU is still "on" and processing other tasks (steps 1-2), a token will be deposited in *P*6 and *CPU_Buffer*. When the CPU is already "on", having a token in *P*6 will enable *T*5 to fire immediately. The tokens from both *P*6 and *CPU_ON* will be removed and a single token will be placed in *CPU_ON*. This is necessary because tokens cannot be allowed to accumulate infinitely in any place.

(8) The token deposited to *CPU_Buffer* will remain there until the CPU is idle as determined by a token in *Idle*. All jobs that arrive while the CPU is "on" will cause the Petri net to cycle through steps 7 and 8.

(9) However, in the event that the job arrival rate is very slow, the CPU may power down and move to the *Stand_By* state. This happens when there is a token in *CPU_ON* and no tokens in *Active* and *CPU_Buffer*, transition *Power_Down_Threshold* becomes enabled. The small circle at the ends of

Table 2: Simulation parameters.

Total simulated time	1000 seconds
Arrival rate	1 per second
Service rate	1 per second

the arcs from *Active* and *CPU_Buffer* specify this inverse logic. Since *Power_Down_Threshold* is a transition with deterministic delay, it will fire after a specified period *Power_Down_Delay* (*PDD*). When *Power_Down_Threshold* fires, a token from *CPU_ON* will be removed and transferred to *Stand_By*. The CPU has now moved to the *Stand_By* state.

By computing the average number of tokens in a certain place during the duration of the simulation time results in the "steady-state" percentage of time the CPU spends in that state. For example, the average number of tokens in *CPU_ON* will indicate the percentage of time the CPU was "on." The average number of tokens in *Power_Up* will indicate the "steady-state" percentage of time that the CPU was "powering up." Of course, these percentages are determined by the *ArrivalRate*, *Service_Rate*, *Power_Down_Delay*, and *Power_Up_Delay* delays. Once the percentages are obtained, they can be used to compute the total energy consumption of the system over time as given in (20):

$$\text{Total Energy} = \left(P_{standby} \times p_{standby} + P_{powerup} \times p_{ppowerup} + P_{idle} \times p_{idle} + P_{active} \times p_{active}\right) \times \text{Time}, \quad (20)$$

where P_x is the power consumption rate and p_y is the steady-state probability of a specific state.

4. Comparison between Simulation, Markov Models, and Petri Net

We have developed a discrete event simulator that emulates the timings of state transitions of CPU. Equation (20) will be used to compute the total energy for the simulator as well. Table 2 lists the simulation time, arrival rate, and service rate parameters. The PXA271 Intel Processor whose power parameters are given in Table 3 [13] is used in this paper. We will compare the predicted steady-state probabilities and the energy estimates of the event simulator, the Markov model and the Petri net model while the *Power_Down_Threshold* is varied from 0.001 to 1 second and the *Power_Up_Delay* is fixed at 0.001, 0.3, and 10 seconds.

Figure 5 shows the percentage of time the CPU spends in the different states when *Power_Up_Delay* or the time for the CPU to "wake up" is fixed at 0.001 seconds. The *Power_Down_Threshold* is the length of time that the CPU waits in the *Idle* state before it transitions to the *Stand_By* mode.

Figure 5 allows us to study the effects of increasing the *Power_Down_Threshold*. Intuitively, it is obvious that as the *Power_Down_Threshold* increases, the *Idle* time increases appropriately as indicated in the figure. The amount of time in *Stand_By* decreases proportionally, and more and more time is spent in *Idle* and the CPU moves to *Stand_By* fewer times. This means that the time spent in powering up decreases as well because there are fewer times the CPU goes to *Stand_By*. Notice that the *Active* time remains constant indicating that for the most part the *Power_Down_Threshold* does not affect the utilization of the CPU.

Table 3: System model Petri net power parameters.

State	Power rate (mW)
CPU *Stand_By*	17
CPU *Idle*	88
CPU *Power_Up*	192.976
CPU *Active*	193
Radio *Stand_By*	$1.44e-4$
Radio *Idle*	0.712
Radio *Power_Up*	0.034175
Radio *Active*	78

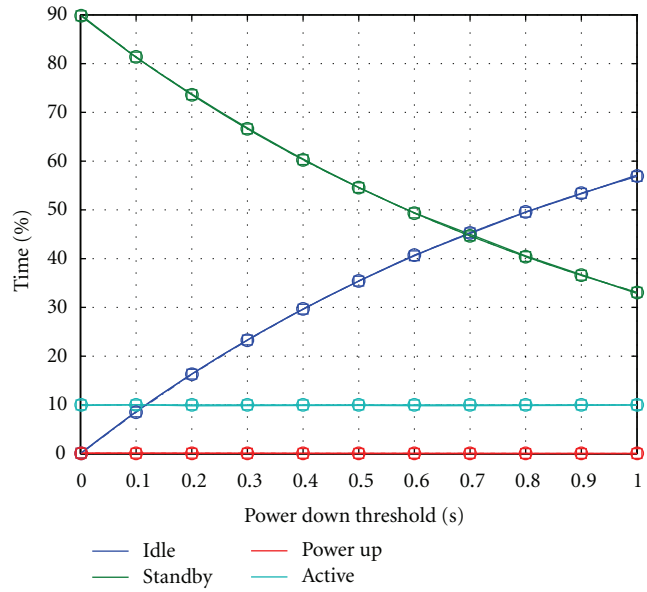


Figure 5: *Power_Up_Delay* = 0.001 seconds.

In all of the figures, the simulator results are given by the solid line. The Markov model is represented by the line with squares, and the Petri net by the line with circles.

Figure 8 shows the energy consumption estimates for each of the three methods. It is interesting to note that the average difference between the Markov model energy estimates compared to the simulator is equal to the average difference between the Petri net and the simulator as shown in Table 4.

Figure 6 depicts the behavior of the CPU when the *Power_Up_Delay* is fixed at 0.3 seconds. Although, the Petri net model seems to overestimate the percentages of each of the four states as compared to the simulator, it tends to be a better indicator of the system than the Markov model. Figure 9 shows the energy consumption estimates for each of the three methods. It is interesting to note that the Petri

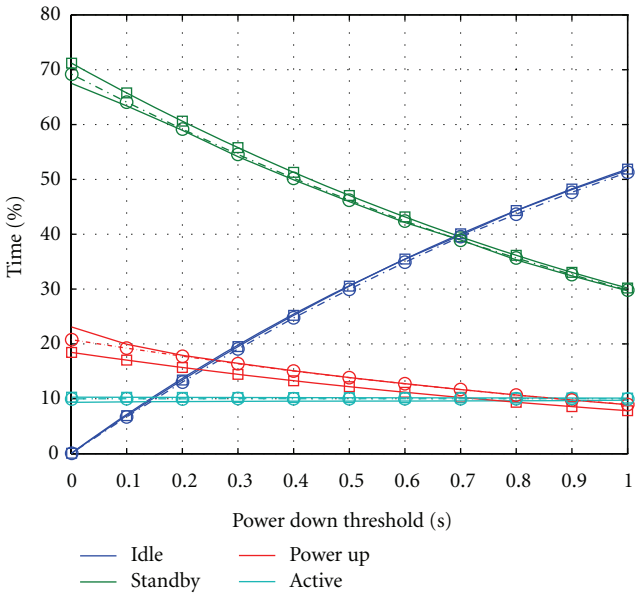


Figure 6: *Power_Up_Delay* = 0.3 seconds.

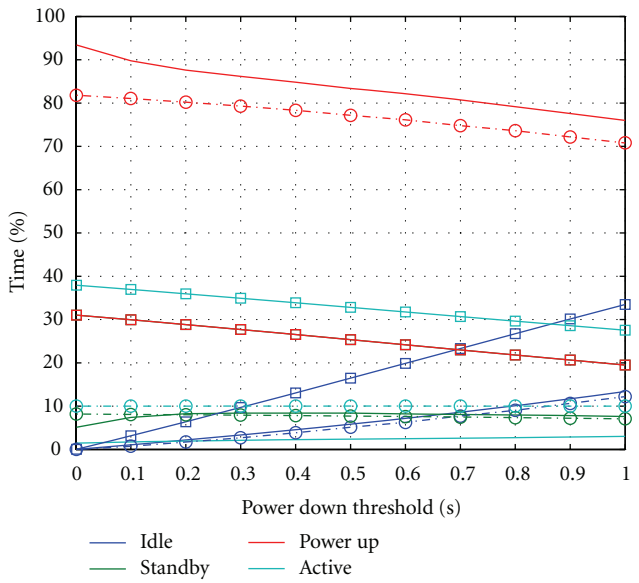


Figure 7: *Power_Up_Delay* = 10 seconds.

Table 4: ΔEnergy (Joules) estimates (*Power_Up_Delay* = 0.001 second).

Power down	ΔSim-Markov	ΔSim-Petri net	ΔMarkov-Petri net
Avg.	7.37	7.37	0.05
Variance	11.88	12.18	0.00
STD DEV	3.45	3.49	0.03
RMSE	8.07	8.08	0.06

net energy estimates are now closer to the simulator results than the Markov model. As Table 5 shows, the average energy estimate difference for all estimates between the simulator and the Markov model is 7.28 Joules, while the difference between the simulator and the Petri net is only 4.99 Joules.

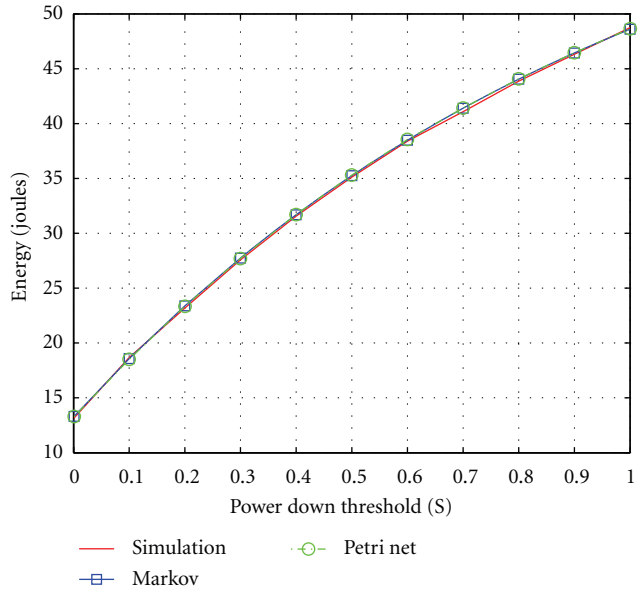


Figure 8: Energy estimates for *Power_Up_Delay* = 0.001 seconds.

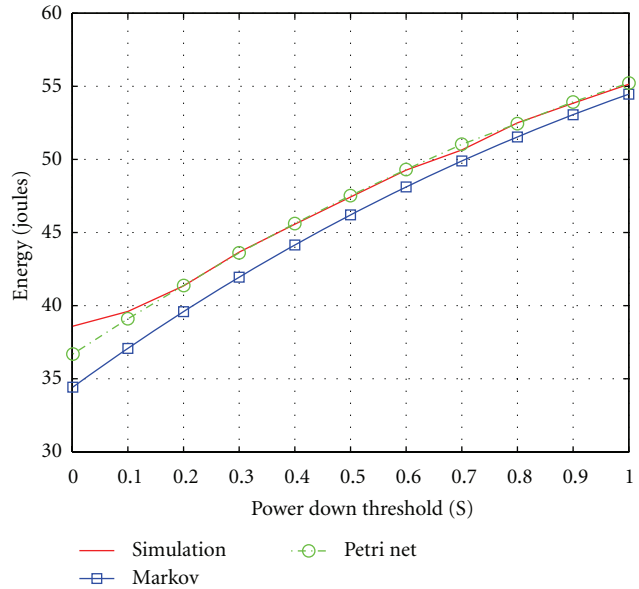


Figure 9: Energy estimates for *Power_Up_Delay* = 0.3 seconds.

Figure 7 depicts the behavior of the CPU of an extreme case when the *Power_Up_Delay* is fixed at 10 seconds. In this scenario, the CPU spends a significant amount of time in *Power_Up* as it "wakes up." In this setting, the Markov model completely fails to estimate the behavior of the simulator. The Petri net on the other hand seems to be in lock step with the simulator results. The energy consumption comparison in Figure 10 and Table 6 further shows that the Petri net model is more accurate than the Markov model.

By comparing three different scenarios (*Power_Up_Delay* of 0.001, 0.4, and 10 seconds) and of which two were extreme

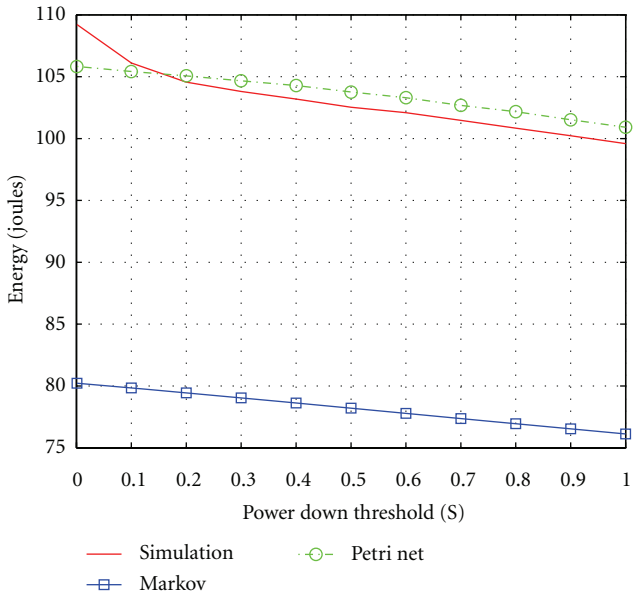


Figure 10: Energy estimates for $Power\_Up\_Delay = 10$ seconds.

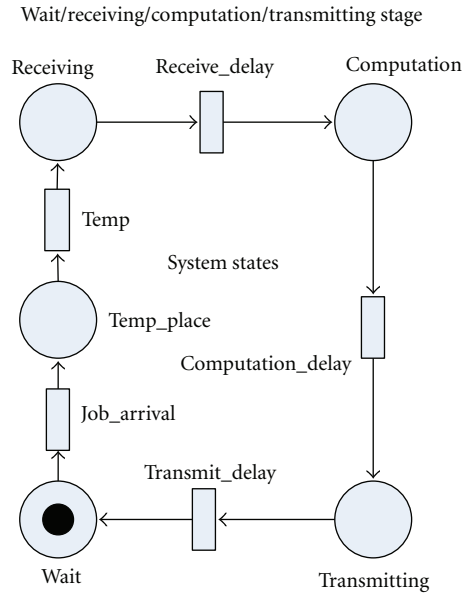


Figure 11: Simple system model of node in wireless sensor network.

Table 5: ΔEnergy (Joules) estimates ($Power\_Up\_Delay = 0.3$ second).

Power down	ΔSim-markov	ΔSim-petri net	ΔMarkov-petri net
Avg.	7.28	4.99	2.29
Variance	6.71	3.55	0.51
STD DEV	2.59	1.88	0.71
RMSE	7.69	5.30	2.39

Table 6: ΔEnergy (Joules) estimates ($Power\_Up\_Delay = 10$ seconds).

Power down	ΔSim-markov	ΔSim-petri net	ΔMarkov-petri net
Avg.	42.41	0.12	42.41
Variance	1.85	0.00	2.00
STD DEV	1.36	0.06	1.41
RMSE	42.43	0.13	42.43

Table 7: Measured power requirements for different IMote2 states.

State	Mean power (mW)
Idle	1.216
Receiving	1.213
Computation	1.253
Transmission	1.028

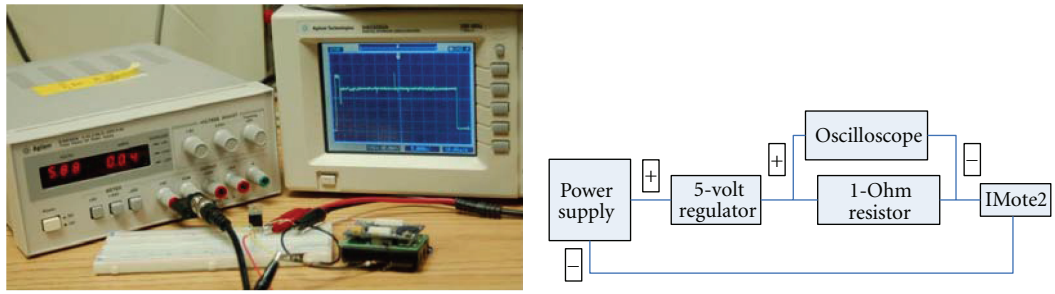
cases (0.001, and 10 seconds), we were able to show that the Petri net is better adept at predicting the behavior of the simulator than the Markov model. The Petri net hence is a better method of modeling a CPU.

Another interesting observation from these experiments is that a *Power_Up_Delay* of 10 seconds results in an energy consumption trend that actually decreases as the *Power_Down_Threshold* increases (see Figure 10). This is because the *Power_Up* power rate is much higher than the *Idle* rate. As the *Power_Down_Threshold* increases, the time spent in *Idle* also increases, and hence decreases the number of time the CPU goes to the *Stand_By* state. As a result, the number of power up transitions decreases, leading to reduced energy usage. From this we can gather that it is more efficient to allow a CPU to idle than to have it repeatedly move from a power down state to active.

5. Evaluation of a Simple Sensor System

In this section, the energy prediction of a Petri net model for a simple sensor system will be compared against real measurements collected from an IMote2 node acting as a sensor node. Figure 11 depicts the generic operating behaviour of a sensor system node. The system remains in a wait state until a random event occurs at which point, a message is received, some computation is required, and then the results are transmitted to some other node. The transition *Job_Arrival* is the only one that fires randomly using an exponential distribution; all others are deterministic transitions. The transition *Temp* and place *Temp_Place* are required in the Petri net to account for the fact that the IMote2 node is not capable of handling events that are less than 1 second apart; the *Temp* transition fires after a fixed 1 second.

Figure 12(b) depicts how a power supply was used to power the IMote2 node. The voltage across a 1.16 Ohm resistor was monitored to determine the current draw of the system. The power consumed by the IMote2 as seen at the battery terminals was measured in four different states of operation: computation, idle, transmission, and receiving. The measured power values listed in Table 7 are the average power consumed in these states. It is interesting to note that the transmission state has the least power consumption, even than that of the Idle case. Although this might seem counterintuitive, it must be borne in mind that while



(a) Setup of data collection for IMote2

(b) Block diagram of data collection for IMote2

Figure 12: IMote2 power collection setup.

Table 8: Petri net transition parameters for a simple system.

Transition	Firing distribution	Delay (sec)	Steady state probability (%)
Job_Arrival	Exponential	3.0	59.8
Temp	Deterministic	1.0	19.7
Receive_Delay	Deterministic	0.00597	0.098
Computation_Delay	Deterministic	1.0274	20.2
Transmit_Delay	Deterministic	0.0059	19.7

Table 9: Steady-state probabilities for a simple system.

State/place	Probability (%)
Wait	59.8
Temp place	19.7
Receiving	0.098
Computation	20.2
Transmitting	19.7

the IMote2 is idling, the receiver is actively "listening" (although it is not receiving anything). The datasheet for the CC2420 radio chip on the IMote2 lists the receive current consumption at 18.8 mA, whereas for transmission, the current draw is 17.4 mA. To obtain the power consumption in the nonidle states, the IMote2 node executed programs that either ran a sort routine repeatedly, transmitted packets, or received packets. This data was collected for the time it took to send or receive 50 packets.

Once the power parameters of the IMote2 were characterized, the energy consumption of the IMote2 as a node in a sensor network was found. This was done by triggering the node randomly for 100 events while the power consumption was monitored. These 100 events took 266.5 Seconds, and resulted in an average power consumption of 1.261 mW. The energy consumption of the IMote2 was found to be 0.336137 J as listed in Table 10.

Using the power parameters collected, the Petri net was simulated until steady-state probability values were obtained. This took about 10 minutes on a 2.80 GHz computer running XP. Table 8 lists the transitions in the Petri net and the delays. Table 9 lists the steady-state probabilities of the places for the given transition parameters in Table 8. Equation (21)

Table 10: Results of actual system and petri net.

IMote2 execution time	266.5 sec
Average IMote2 power	1.261 mW
IMote2 energy usage	0.336137 J
Petri net energy usage	0.326519 J
Percent difference	2.95

was used to compute the energy consumption resulting from these probabilities. As Table 10 indicates, the actual energy consumed by the IMote2 and the energy predicted by the Petri net vary only by about 3 percent.

$$\begin{aligned}\text{Total Energy} = {} & \left(P_{Wait} \times \left(p_{Wait} + p_{Temp\_Place}\right)\right. \\ & + P_{Receiving} \times p_{Receiving} \\ & + P_{Computation} \times p_{Computation} \\ & \left. + P_{Transmitting} \times p_{Transmitting}\right) \times \text{Time}.\end{aligned} \tag{21}$$

6. Modeling a Sensor Node in Wireless Sensor Networks

In this section, stochastic colored Petri nets are used to model the energy consumption of a sensor node in a wireless sensor network using open and closed workload generators as shown in Figures 13 and 14. Generally, the behavior of nodes in a wireless sensor network follows the same basic pattern. First, a node in *Idle* or *Stand_By* is "awoken" by either an external event or a message from another node. This node then proceeds to process the event or message that typically involves some computation. The resulting information is then transmitted to other sensor nodes or a centralized data collector. Finally, the node moves either to *Idle* or *Stand_By* if no more events arrive for some time period. It then "waits" for another event.

Unlike Markov models, the ease with which a Petri net can be designed allows for complicated scenarios to be modeled. Figures 13 and 14 describe Petri net models of a processor capable of servicing multiple tasks. A colored Petri net is capable of assigning numerical values or other attributes

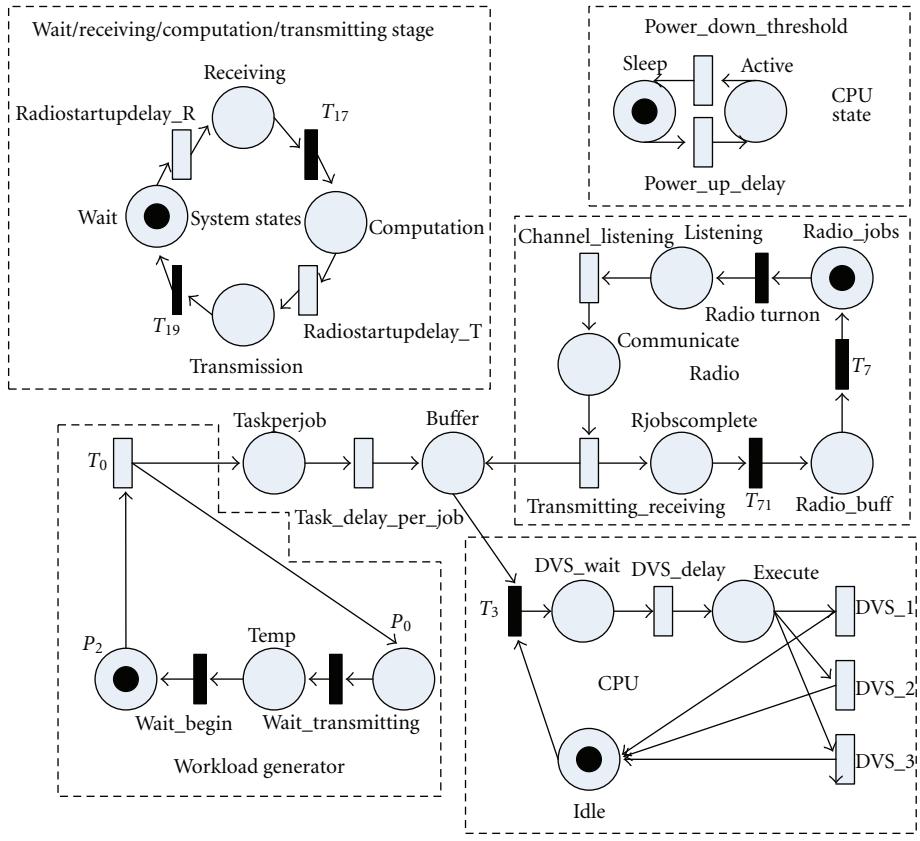


Figure 13: Closed system model of node in wireless sensor network.

to tokens to allow for enhanced decision making capabilities. The characteristics of a token can be checked, using expressions called "local guards," as the token is input to a transition. Local guards can be used to allow or deny tokens from activating a transition. For example, transitions *DVS_1*, *DVS_2*, and *DVS_3* have local guards associated with them and the appropriate transition fires only if the token in its input place (*Execute*) has a corresponding value of 1, 2, or 3 associated with it. This feature allows the model to simulate a DVS processor using a practical variable voltage system where the processor stops executing while changing operating parameters [5]. Tokens of different values result in different execution speeds simulating the change in the operating parameters.

The Petri nets in this section model a system that services jobs of a single type. As soon as a job is generated, a token is placed in either place *P*0 for the closed workload generator (Figure 13) or place *Event_Arrival* for the open workload generator (Figure 14). We use the processor and radio parameters as given in Table 3 [13] for the iMote2 sensor platform to provide realistic analysis.

6.1. Energy Model Using a Closed Workload Generator. Figure 13 demonstrates a stochastic colored Petri net (SCPN) [22] model of a sensor node using a closed workload generator. The portion of the Petri net labeled "*Workload Generator*" generates the job events, while the portions marked "*Radio*" and "*CPU*" refer to those respective components. The system is composed of four states: "*Wait*," "*Receiving*," "*Computation,*" and "*Transmitting*." There are two states associated with the CPU: "*Sleep*" and "*Active*."

In addition, our model implemented based on TimeNET utilize a feature called "global guards" to specify more "global" conditions for the firing of transitions. We use global guards in the forms of expressions at the transitions that remove the need to provide connections using arcs. For example, these conditions can be used to check for the number of tokens in a given place. This simplifies the drawing of the Petri net significantly.

Simulation of the open workload Petri net given in Figure 13 results in the movement of tokens as given below.

Global guards for the Petri net in Figure 13 are given in Table 11.

The Petri nets in this section model a system that services jobs of a single type. As soon as a job is generated, a token is placed in either place *P*0 for the closed workload generator (Figure 13) or place *Event_Arrival* for the open workload generator (Figure 14).

The system then moves from the "*Wait*" state to the "*Receiving*" state using transition *RadioStartUpDelay_R* which simulates the time for the radio to start up. Once the system is in the *Receiving* state, this allows the token in *Radio_Jobs* (Radio) to move to *Listen*. The Petri net begins to simulate "*Channel_Listening*" for an available communication slot. Thereafter, the radio proceeds to "receive" information in the *Communicate* place, and after which, a token is

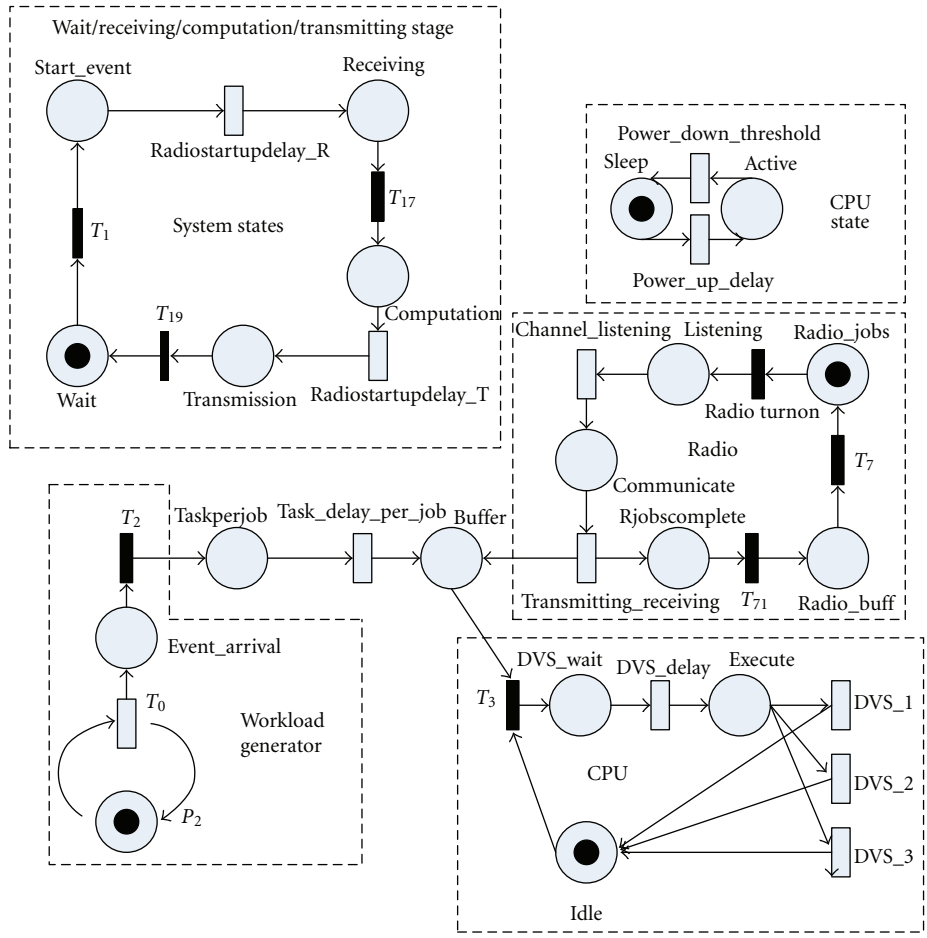


Figure 14: Open system model of node in wireless sensor network.

deposited in the CPU *Buffer* for the purpose of awakening the CPU for error checking the received packet. Any tokens (jobs) in the *Buffer* cause the CPU to transition from the *Sleep* state to the *Active* state.

If the CPU is *Idle*, the token moves from the place *Buffer* to *DVS_Wait*. The transition labeled *DVS_Delay* simulates the time for any overhead to execute a particular task. Finally, the token moves to the *Execute* place to simulate the execution of the job on the CPU. Depending on the value of the token, either *DVS_1*, *DVS_2*, or *DVS_3* is enabled. Once enabled, the transitions will fire after fixed intervals that represent the time to service the appropriate task. Thereafter, the CPU then moves to *Idle*.

Once the CPU has "processed" the received packet, the radio moves to an "idle" mode. The system then moves to the *Computation* state. The token that was generated and placed in *TaskPerJob* is moved to the *Buffer* and the CPU proceeds to service the job simulating any computation required for the event generated. The system then moves to the *Transmitting* state using transition *RadioStartUpDelay_T* to awaken the radio, where the processed information is "transmitted" to some base station using the same steps as for the *Receiving* state. Finally, the state moves back to the *Wait* state. The CPU *Sleep*/*Active* states operate independently of the system states. The CPU is "woken" from sleep if any tokens are placed in the *Buffer*; however, depending upon the CPU *PowerDownThreshold*, the CPU may go back to "sleep" during the communication stage. In which case, the CPU may need to be woken up again.

The Petri net assumes that the radio is put to sleep after the *Transmitting* state. However, between the *Receiving* and *Computation* states the radio is idle. The Petri net also assumes that the radio wake up cost is the same whether the radio is awoken from sleep to active or idle to active; *RadioStartUpDelay_R* = *RadioStartUpDelay_T* = *RadioStartUpDelay*. We will present the simulation results and analysis in Section 7.

6.2. Energy Model Using an Open Workload Generator. Figure 14 demonstrates a stochastic colored Petri net (SCPN) [22] model of a sensor node using an open workload generator. This Petri net is very similar to the one with the closed workload generator presented previously. Many of the transitions and global guards given in Table 11 are also used here. The three additional transitions unique to this model are given in Table 12.

The main difference between the close model (see Figure 13) and the open model (see Figure 14) is that events arrive independently of the state of the system in the Petri net given in the open model. When transition $T0$ fires randomly

Table 11: Closed system model Petri net transition parameters.

Transition	Type	Delay	Global guard
$T0$	DET	AR	$(\#Wait > 0)$
RadioStartUpDelay_R	DET	0.000194	$(\#P0 > 0)$
RadioTurnOn	INST (3)	—	$((\#Receiving > 0) \parallel (\#Transmitting > 0))$
Channel_Listening	DET	0.001	$((\#Receiving > 0) \parallel (\#Transmitting > 0))$
Transmitting_Receiving	DET	0.000576	NA
Power_Up_Delay	DET	0.253	$(\#Buffer > 0)$
$T3$	INST (2)	—	$(\#Active > 0)$
DVS_Delay	DET	0.05	NA
DVS_1	DET	0.03	$dvs1 == 1.0$ (Local Guard)
DVS_2	DET	0.01	$dvs2 == 2.0$ (Local Guard)
DVS_3	DET	0.081578	$Comm == 3.0$ (Local Guard)
$T17$	INST (3)	—	$((\#Buffer == 0)$ && $(\#Idle > 0)$ && $(\#RJobsComplete == ComPackets))$
$T71$	INST (2)	—	$((\#Buffer == 0)$ && $(\#Idle > 0)$ && $(\#RJobsComplete == ComPackets))$
$T7$	INST (1)	—	$((\#Computation > 0) \parallel (\#Wait > 0))$
Task_Delay_Per_Job	DET	0.000001	$\#Computation > 0$
RadioStartUpDelay_T	DET	0.000194	$((\#TaskPerJob == 0)$ && $(\#Buffer == 0)$ &&$(\#Idle > 0))$
$T19$	INST (3)	—	$((\#Buffer == 0)$ && $(\#Idle > 0)$ && $(\#RJobsComplete == ComPackets))$
Power_Down_Delay	DET	PDT	$((\#Buffer == 0)$ && $(\#Idle > 0))$
Wait_Transmitting	INST (3)	—	$(\#Transmitting > 0)$
Wait_Begin	INST (3)	—	$(\#Wait > 0)$

using an exponential distribution, a token is deposited back in place $P2$ and a new token is placed in place *Event_Arrival*. With a token in place $P2$, transition $T0$ can fire again at any time. In order to assure that multiple but closely spaced events each trigger a new system cycle, place *Start_Event* and transition $T1$ were needed.

As mentioned before, the ease of building Petri nets allows one to simulate complex behavior. The Petri net in Figure 14 describes just one particular scenario. Any variation of other scenarios can just as easily be simulated by slight modifications to the Petri net. This flexibility and ease in modeling a system can go a long way towards obtaining an understanding of the system and hence exploiting power saving features.

Using the Petri net in Figure 14, the effects of the *Power_Down_Threshold* of the CPU on the system can easily be studied. The next section explores results obtained from simulating the Petri net.

7. Energy Evaluation of a Sensor Node

Based on the Petri net models presented in the previous section, this section evaluates the energy consumption of a sensor node and discusses the potential applications of our Petri net model. For all experimental results presented in this section, we use our models to estimate the total energy consumption for a time interval of 15 minutes unless specified otherwise.

7.1. Analysis Using Closed Workload. Figure 15 describes the energy characteristics of a wireless sensor node with a closed generator as *Power_Down_Threshold* increases. We aim to use our model to address the question that was posed in Section 1: what is the optimum *Power_Down_Threshold* that yields minimum energy consumption in a wireless sensor network?

In Figure 15, powering down the CPU immediately after the computation does not result in the minimal energy consumption, neither does always keeping the CPU active achieve the optimal energy efficiency. The optimum energy consumption of approximately 2432 Joules occurs at a *Power_Down_Threshold* of 0.00177 seconds. This is a 35% decrease in energy consumption that occurs when the CPU is immediately powered down to a low power state. This is also a 29% decrease in energy consumption that occurs when the CPU is never powered down. Interestingly, it is no coincidence that the minimal energy consumption occurs at this point as will be illustrated.

In the closed model, all transitions are deterministic including transition $T0$ for generating jobs as well as transition *Channel_Listening*. Although, the results of the closed model are predictable, the results from this model can be used to identify four classes of energy values due to three boundaries that arise. These three boundaries delineate shifts in energy consumption trends as *Power_Down_Threshold* increases. The boundaries are generated because there are three points in the system where the CPU can power

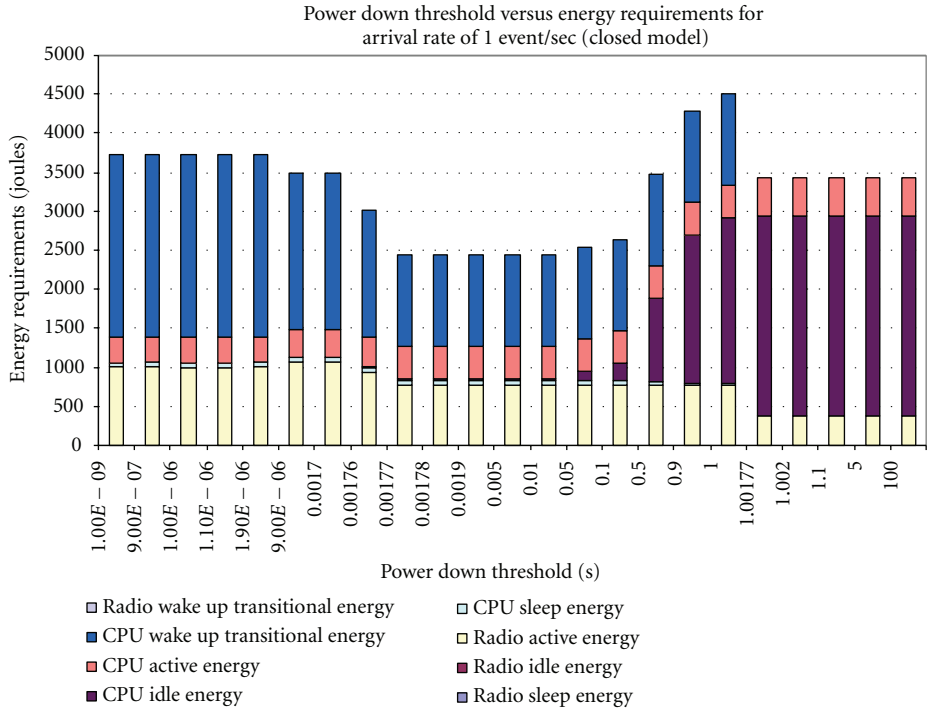


Figure 15: Power down threshold versus energy requirement for a closed model with a job arrival rate of 1 event/second.

down after having been powered up. In other words, the boundaries are the results of three different intervals in the system where the CPU can power down after having been powered up.

The first class of values are those associated when the CPU is powered down after computation at a threshold less than the sum of the least transition times between consecutive CPU usages. Hence, the system is required to power up three times in one system cycle. From Table 11, this happens when *Power_Down_Threshold* < *Tasks_Delay_Per_Job* since this is the transition with the least time between consecutive CPU usages. The second class results when the CPU powers down after some larger threshold and as a result is required to power up twice during the system cycle. In the third class, the CPU powers up once, and in the fourth class the CPU powers up and never powers down again. Each will be examined in detail in the following.

When *Power_Down_Threshold* is smaller than *Tasks_Delay_Per_Job*, that is, it is less than the time between the *Receiving* and *Computation* states, the CPU powers down and is then forced to power up 3 times per system cycle as seen in Figure 16. The red depicts the CPU *Power Up* energy cost, the green depicts the CPU *Active* energy cost, and the aqua depicts the CPU *Idle* energy cost. The purple depicts the radio energy costs.

However, when *Power_Down_Threshold* is larger than *Tasks_Delay_Per_Job*, then the CPU will not power down and will not need to be powered up again between the *Receiving* and *Computation* states. This deterministic transition determines how long the CPU remains idle between the *Receiving* and *Computation* state. At this point, the CPU remains

Table 12: Open system model Petri net transition parameters.

Transition	Type	Delay	Global guard
$T1$	INST (2)	NA	$(\#TaskPerJob > 0)$
$T2$	INST (1)	NA	$(\#Wait > 0)$
$T171$	INST (3)	NA	$(\#Wait == 0)$

powered up between the Receiving and Computation states, and as a result there is one less CPU power up.

Since *Power_Down_Threshold* is larger than *Tasks_Delay_Per_Job*, the CPU is powering down fewer times. Hence, the time it takes to power up the CPU is being saved from the cycle time as depicted in Figure 17.

Again, applying the same principal as above, select the next minimum sum of transitions of deterministic delays. From Table 12, this happens for *RadioStartUpDelay* + *Channel_Listening* + *Transmitting_Receiving* or the sum of time when the radio is awoken, a wireless communication slot found, and data is transmitted.

So when

$$Power\_Down\_Threshold > RadioStartUpDelay + Channel\_Listening + Transmitting\_Receiving, \quad (22)$$

the CPU now remains powered up between the *Computation* and *Transmitting* states. There is no *Power_Up_Delay* in the *Transmitting* state as shown in Figure 18.

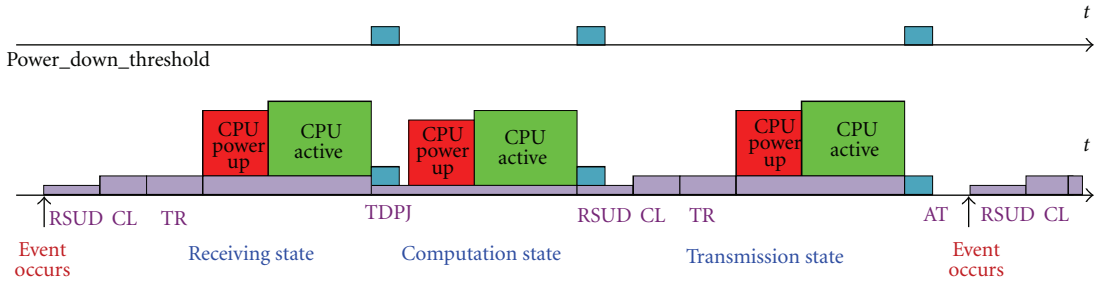


Figure 16: Energy diagram of 3 CPU power ups in one system cycle for closed model (not drawn to scale).

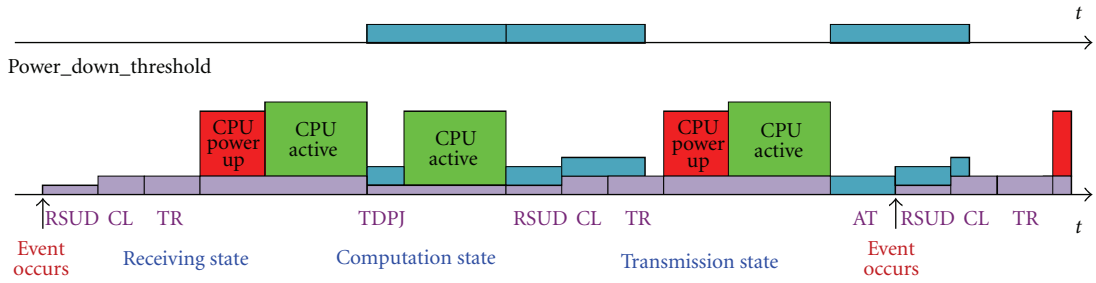


Figure 17: Energy diagram of 2 CPU power ups in one system cycle for closed model (not drawn to scale).

The final set of transitions between two consecutive CPU usage requests are between the end of the current *Transmission* state and the next *Receiving* state. This means that when

$$\begin{aligned} Power\_Down\_Threshold > {} & Job\_Arrival\_Rate \\ & + RadioStartUpDelay \\ & + Channel\_Listening \\ & + Transmitting\_Receiving, \end{aligned} \tag{23}$$

the CPU now is always powered up. The CPU no longer powers down between the end of the previous *Transmitting* state and the next subsequent event.

The difference between this and the previous cases is that *Power_Up_Delay* is no longer a factor. *Power_Down_Threshold* is sufficiently large that the CPU does not power down after it powers up for the first time between events for this arrival rate as shown in Figure 19.

Each of these cases with the appropriate cycle times is presented in Table 13.

As Table 13 indicates, a sudden shift in energy consumption can be expected when

$$\begin{aligned} Power\_Down\_Threshold > {} & RadioStartUpDelay \\ & + Channel\_Listening \\ & + Transmitting, \end{aligned} \tag{24}$$

that is, *Power_Down_Threshold* > 0.00177 seconds. This can be verified in Figure 15. When *Power_Down_Threshold* is 0.00176 second, the system energy consumption is 3007.827 Joules. However, when *Power_Down_Threshold* is set to 0.00177 second, the system energy consumption is 2431.95 Joules. This is a drop of 19.15% in energy consumption just by increasing *Power_Down_Threshold* by 0.00001 seconds. This decrease in energy consumption results from the fact that the CPU is powering up only once per cycle as opposed to twice. The CPU is saving the heavy *Power Up* energy penalty.

Table 13: *Power_Down_Threshold* cases for a closed system and their associated cycle times.

Power down threshold criteria	Delay (second)	Power up delays	Cycle time (second)
=0	0	3	2.00254
>*TDPJ*	0.000001	2	1.74954
>*RSUD*+*CL*+*TR*	0.00177	1	1.49654
>*AT*+*RSUD*+*CL*+*TR*	1.00177	0	1.24354

However, as *Power_Down_Threshold* continues to increase, the system energy consumption increases steadily. Upon closer inspection, the CPU *Power Up* energy remains constant; however, it is the CPU *Idle* energy that contributes to the overall increase in the system consumption. As *Power_Down_Threshold* increases, the one *Power Up* penalty is being avoided; however, the time the CPU is in the *Idle* state is being increased and hence the greater system energy consumption is.

This increase continues and even results in a maximum energy consumption of 4501.96 Joules at a *Power_Down_Threshold* of 1 second and then drops suddenly to 3429.92 Joules when *Power_Down_Threshold* increases by 0.00177 seconds. This abrupt change is, of course, due to the CPU

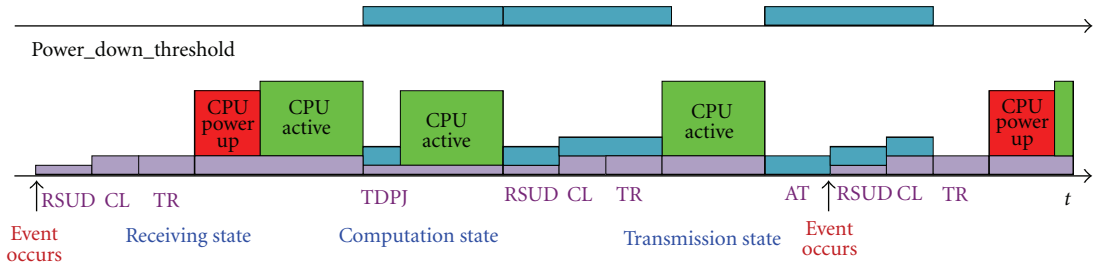


Figure 18: Energy diagram of 1 CPU power up in one system cycle for closed model (not drawn to scale).

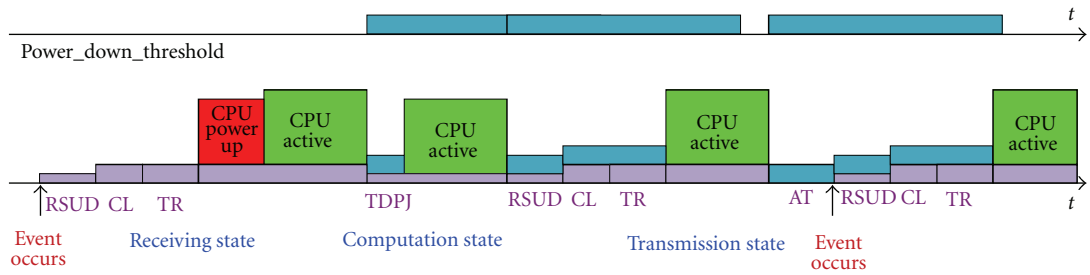


Figure 19: Energy diagram of no CPU power ups (after the first one) in one system cycle for closed model (not drawn to scale).

remaining always on as opposed to powering down once per cycle as indicated in Table 13. Although the CPU *Idle* energy increased during this transition, the saving from the heavy powering up penalty is sufficient to result in the system energy consumption that is less than when the CPU was powering down immediately after it finished computation or when *Power_Down_Threshold* was zero.

We can also see how the tradeoff between the CPU *Power Up* and *Idle* energy consumption of only one power up delay results in the minimum energy consumption at a *Power_Down_Threshold* of 0.00177 seconds. This is when

$$\begin{aligned} Power\_Down\_Threshold > {} & RadioStartUpDelay \\ & + Channel\_Listening \\ & + Transmitting. \end{aligned} \tag{25}$$

We can see from this exercise that powering down the CPU immediately after it completes processing is not the best solution. A careful analysis of the system parameters should be conducted before deciding upon a *Power_Down_Threshold* that will minimize overall energy consumption.

An interesting point to note is that as *Power_Down_Threshold* is steadily increased, the *Radio Active Energy* decreases. The reason for this is apparent because the energy consumption of the *Radio* in the active state is the time spent in the *Receiving* and *Transmitting* states. However, during these states, after communication, the "received" packet needs to be processed by the CPU for integrity checking of the checksum. If the CPU has powered down by then, the *Radio* needs to wait in an active state consuming the active energy rate while the CPU powers up and then processes the packet. This decrease in the *RadioActiveEnergy* as *Power_Down_Threshold* is decreased is indicative of the fact that the CPU is powering down fewer and fewer times during each cycle, and as a result the *Radio* needs to spend less time in an active state while the CPU powers up. When *Power_Down_Threshold* increases to the point where the CPU is always on, the energy consumption of 368.24 Joules is just for the communication involved without the powering up time overhead.

Although the cases where the system CPU moves from powering up twice to only one time

$$\begin{aligned} Power\_Down\_Threshold > {} & RadioStartUpDelay \\ & + Channel\_Listening \\ & + Transmitting\_Receiving \end{aligned} \tag{26}$$

and powering up once to always staying on

$$\begin{aligned} Power\_Down\_Threshold > {} & Arrival\_Time \\ & + RadioStartUpDelay \\ & + Channel\_Listening \\ & + Transmitting\_Receiving \end{aligned} \tag{27}$$

can be clearly identified in Figure 15, it is a little harder to identify the first case where *Power_Down_Threshold* > *Task_Delay_Per_Job* or $1e^{-6}$ Seconds. This is because Time NET uses the float type and since it is a very small value used during simulation, the value of the floating value and the rounding error may have skewed the results of the simulation.

By analyzing the closed model case, an understanding of the internal workings of the Petri net and its resulting

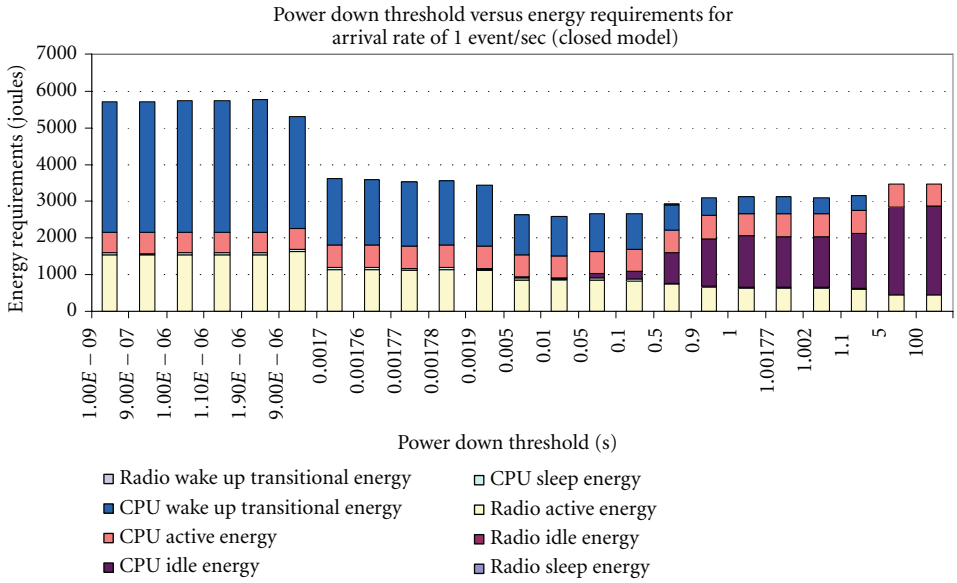


Figure 20: *Power_Down_Threshold* versus energy requirements of an open model for an arrival rate of 1 event/second.

behavior was obtained. This understanding can be applied to the open model as well, however, while acknowledging that the four cases in Table 13 are not as distinct for the open model.

7.2. Analysis Using Open Workload. As Figure 20 shows, having the CPU go to sleep immediately after processing a task is not beneficial for this system either. This can be seen when the *Power_Down_Threshold* is $1.00e-9$ seconds or almost zero. The *Power Up* transitional energy to wake the CPU becomes prohibitive. However, it is not beneficial to always keep the CPU "on" either as indicated when *Power_Down_Threshold* is five seconds or more. However, when *Power_Down_Threshold* is approximately 0.01 seconds, the energy requirement is approximately 2589 Joules which is almost 55% less than the energy consumed when the CPU is shut down immediately after every task. This is also less than the energy consumed when the CPU is always on by almost 26%.

Figure 20 is not meant to present the minimum energy level that can possibly be obtained, but this figure does indicate the energy consumption in the four distinct classes. Although Figure 20 is not plotted at even intervals of *Power_Down_Threshold*, the energy trends seem to be relatively smooth.

Except for transitions $T0$ and *Channel_Listening*, all other transitions are deterministic that fire after fixed intervals. An exponential distribution was used for transitions $T0$ and *Channel_Listening* due to the random and sporadic nature of these events. Assuming that the firing time of these exponent transition can be treated as a random variable, this will result in an average firing rate of $(1/ArrivalRate)$. Since the *Channel_Listening* time is so small, it will be used as it is.

In the open model, transitions $T0$ and *Channel_Listening* fire randomly in the specified time interval as given by the exponential distribution. A randomness is introduced that blurs the sharp boundaries in the energy consumption as *Power_Down_Threshold* increases as was seen for the closed model.

Table 14: *Power_Down_Threshold* cases for an open system and their associated cycle times.

Power down threshold criteria	Power up delays	Cycle time (second)
0	3	1.00254
$>TDPJ$	2	0.74954 (1.0)
$>RSUD + CL + TR$	1	0.49654 (1.0)
$>AT + RSUD + CL + TR$	0	0.24354 (1.0)

With open workload, there may not be a delay between the current cycle and the next cycle because the transition that dictates the arrival rate now fires randomly according to an exponential distribution. As Figure 14 and Table 12 indicates after transition $T0$ fires, the token is returned back to place $P2$ again which enables the transition again. $T0$ may fire again immediately after it has just fired or it may fire after waiting the full length of the interval. When $T0$ will fire will be determined by the probabilistic nature of the exponential distribution.

It is possible that when the system finishes servicing the current "event" a new event is waiting for it. Therefore, there is no *Arrival_Time* delay between the current and the next "event." This can happen when the *Arrival_Time* is less than the *Cycle Time* minus the *Arrival_Time* as was expressed before. With an *Arrival_Rate* of 1, event occurs every 1.0 seconds on average $E[X] = 1/\lambda$ where λ is the *Arrival_Rate*. Table 13 can be used and the "Cycle Time" values can be modified by removing the average *Arrival_Rate* from the original cycle times.

The cycle times have been adjusted in Table 14 to show the average cycle times of the system given

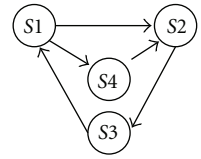


Figure 21: Wireless sensor network.

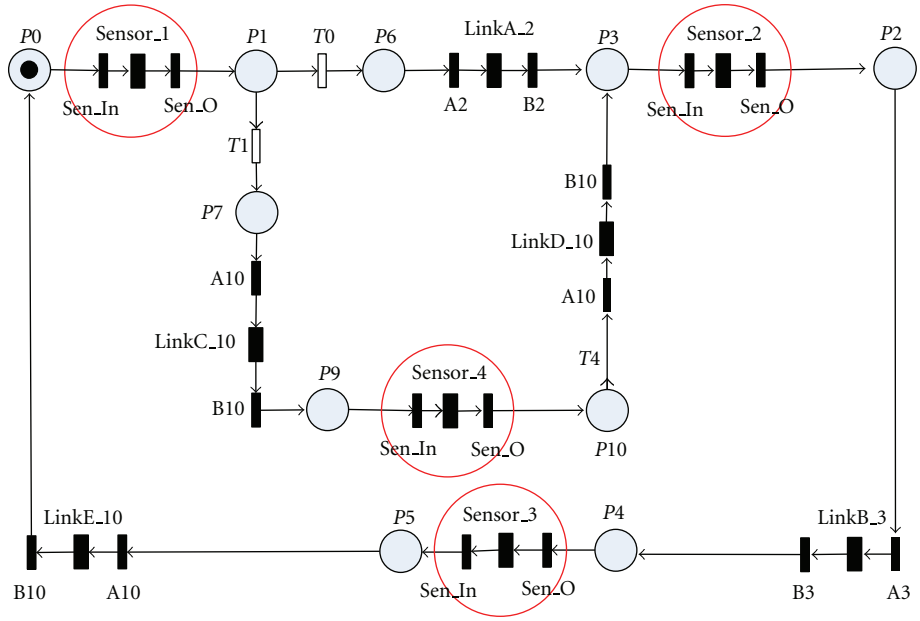


Figure 22: Petri net of wireless sensor network.

Table 15: *Power_Down_Threshold* cases for an open system and their associated down times.

Power down threshold criteria	Cycle time (second)	Down time (second)
0	1.00254	0
$>TDPJ$	0.74954 (1.0)	0.25046
$>RSUD + CL + TR$	0.49654 (1.0)	0.50346
$>AT + RSUD + CL + TR$	0.24354 (1.0)	0.75646

Power_Down_Threshold. However, it should be noted, that since transition $T0$ fires every 1.0 seconds on average, when *Power_Down_Threshold* is equal to 0, events are being generated at a faster rate than can be serviced as indicated by the cycle time. The system is slightly saturated.

In the next three cases, the cycle time is shorter than the average *Arrival_Rate* implying that the average *Arrival_Rate* dictates how often events are serviced. In the latter three cases, the system completes servicing the previous event and waits for the next event to be generated. The length of the time that the system waits in idle is given in the last column "Down Time" in Table 15. In the closed model of the system, since events always arrive after a fixed interval after the system has completed servicing the previous event, the "Down Time" is always 1.0 seconds. Unlike the closed model results in Figure 15 where the minimum energy consumption at the corresponding *Power_Down_Threshold* is apparent, identification of this point for an open model system can be challenging and can fall at a *Power_Down_Threshold* that has not been simulated. The open model of the system presented in this paper contains only two transitions that use exponentially distributed firing rate, and already their results are unpredictable in terms of arriving at an expression to analytically explain them.

If a larger combination of transitions using deterministic and exponential distribution firing rates are used, the resulting outputs can be expected to be even more difficult to analytically derive. This indicates that Petri nets are a very important tool in modeling and analyzing systems.

8. Modeling Wireless Sensor Networks

The Petri net models constructed for the sensor nodes can also be used to create a network of sensors. TimeNET has the capability of modularizing a Petri net so that a hierarchy of Petri nets can be designed. For example, Figure 21 depicts a sensor network composed of four nodes $S1$, $S2$, $S3$, and $S4$ with communication links as described. Slightly modifying the Petri net shown in Figure 13 and substituting it for each of the nodes the sensor network can be created as shown in Figure 22. The filled-in box labeled *Sensor_1* corresponds to $S1$, and so forth. The box labeled *LinkA_2* corresponds to a Petri net simulating the distance link between $S1$ and $S2$

which also incorporates bit error rate, propagation delay, and channel availability.

Using this platform, the energy consumption of a wider range of areas such as routing, network communications, and even high level sensing applications can be modeled and simulated. With such a detailed model from the physical layer to the network layer, our platform allows for research that require knowledge at all these levels such as energy-aware cross-layer routing. In fact, this platform can be used for almost an endless number of research areas.

9. Conclusion

Using stochastic colored Petri nets, this paper develops a detailed and flexible energy model for a wireless sensor node. The experimental results indicate that this model is more accurate than the one based on Markov models. This is due to the fact that a Markov model requires the modeled systems have memoryless states. A wireless sensor node that relies on time to dynamically change its power state does not satisfy the Markov chain's memoryless requirements. In addition, the Petri net model is much more flexible than the Markov model and can easily accommodate changes.

Further, in this paper, we have successfully demonstrated using our model that immediately powering down a CPU after every computation is not an energy minimal option and nor is never powering down the CPU. However, using our model, it is possible to identify a *Power_Down_Threshold* that results in large power savings. Through this example, we were able to show that our model is very useful and provides a valuable platform for energy optimization in wireless sensor networks.

The drawbacks of our Petri net model is that simulating Petri nets can be computationally intensive and require relatively long simulation time to achieve steady-state probabilities. The models presented in this paper required between 10 and 15 minutes of simulation on a 2.8 GHz computer running Windows XP to stabilize. In comparison, evaluation of closed-form Markov equations is almost instantaneous.

Acknowledgments

This work was supported in part by NSF Grants (GK-12 no. 0538457, IGERT no. 0504494, CNS no. 1117032, EAR no. 1027809, IIS no. 091663, CCF no. 0937988, CCF no. 0737583, CCF no. 0621493).

References

[1] A. Seyedi and B. Sikdar, "Modeling and analysis of energy harvesting nodes in wireless sensor networks," in *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing*, pp. 67–71, September 2008.

[2] J. Liu and P. H. Chou, "Idle energy minimization by mode sequence optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 4, article 38, 2007.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," in *Proceedings of the IEEE International Solid-State Circuits Conference 47th Annual (ISSCC '00)*, pp. 294–295, February 2000.

[4] T. Pering, T. Burd, and R. Brodersen, "Simulation and evaluation of dynamic voltage scaling algorithms," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pp. 76–81, August 1998.

[5] G. Qu, "What is the limit of energy saving by dynamic voltage scaling?" in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '01)*, pp. 560–563, November 2001.

[6] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "Dynamic voltage scaled microprocessor system," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, 2000.

[7] A. Shareef and Y. Zhu, "Energy modeling of processors in wireless sensor networks based on petri nets," in *Proceedings of the 37th International Conference on Parallel Processing Workshops (ICPP '08)*, pp. 129–134, Portland, Ore, USA, September 2008.

[8] A. Shareef and Y. Zhu, "Energy modeling of wireless sensor nodes based on Petri nets," in *Proceedings of the 39th International Conference on Parallel Processing (ICPP '10)*, pp. 101–110, San Diego, Calif, USA, September 2010.

[9] A. Zimmermann, M. Knoke, A. Huck, and G. Hommel, "Towards version 4.0 of TimeNET," in *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB '06)*, pp. 477–480, 2006.

[10] P. Hu, Z. Zhou, Q. Liu, and F. Li, "The HMM-based modeling for the energy level prediction in wireless sensor networks," in *Proceedings of the 2007 2nd IEEE Conference on Industrial Electronics and Applications (ICIEA '07)*, pp. 2253–2258, May 2007.

[11] Y. Zhang and W. Li, "An energy-based stochastic model for wireless sensor networks," in *Proceedings of the Wireless Sensor Network*, 2011.

[12] Y. Wang, M. C. Vuran, and S. Goddard, "Stochastic analysis of energy consumption in wireless sensor networks," in *Proceedings of the 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '10)*, Boston, Mass, USA, June 2010.

[13] D. Jung, T. Teixeira, A. Barton-Sweeney, and A. Savvides, "Model-based design exploration of wireless sensor node lifetimes," in *Proceedings of the 41rth European Conference on Wireless Sensor Networks (EWSN '07)*, 2007.

[14] S. Coleri, M. Ergen, and T. J. Koo, "Lifetime analysis of a sensor network with hybrid automata modelling," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, pp. 98–104, ACM, New York, NY, USA, September 2002.

[15] S. Kellner, M. Pink, D. Meier, and E. O. Blaß, "Towards a realistic energy model for wireless sensor networks," in *Proceedings of the 5th Annual Conference on Wireless on Demand Network Systems and Services (WONS '08)*, pp. 97–100, January 2008.

[16] N. Kamyabpour and D. Hoang, "A task based sensor-centric model for overall energy consumption," in *Proceedings of the 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '11)*, pp. 237–244, 2011.

[17] N. Kamyabpour and D. B. Hoang, "Modeling overall energy consumption in wireless sensor networks," submitted, http://arxiv.org/abs/1112.5800.

[18] M. Korkalainen, M. Sallinen, N. Kärkkäinen, and P. Tukeva, "Survey of wireless sensor networks simulation tools for demanding applications," in *Proceedings of the 5th International Conference on Networking and Services (ICNS '09)*, pp. 102–106, April 2009.

[19] X. Fu, Z. Ma, Z. Yu, and G. Fu, "On wireless sensor networks formal modeling based on petri nets," in *Proceedings of the 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '11)*, pp. 1–4, 2011.

[20] D. R. Cox, "The analysis of non-markovian stochastic processes by the inclusion of supplementary variables," *Proceedings Cambridge Philosophical Society*, vol. 51, no. 3, pp. 433–441, 1955.

[21] R. German, "Transient analysis of deterministic and stochastic petri nets by the method of supplementary variables," in *Proceedings of the 3rd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '95)*, pp. 394–398, IEEE Computer Society, Washington, DC, USA, 1995.

[22] TimeNET 4.0 A Software Tool for the Performability Evaluation with Stochastic and Colored Petri Nets, User Manual.