Energy Modeling of Wireless Sensor Nodes Based on Petri Nets

Ali Shareef, Yifeng Zhu

Department of Electrical and Computer Engineering
University of Maine, Orono, USA
Email: {ashareef, zhu}@eece.maine.edu

***Abstract*—Energy minimization is of great importance in wireless sensor networks in extending the battery lifetime. Accurately understanding the energy consumption characteristics of each sensor node is a critical step for the design of energy saving strategies. This paper develops a detailed probabilistic model based on Petri nets to evaluate the energy consumption of a wireless sensor node. The model factors critical components of a sensor node, including processors with emerging energy-saving features, wireless communication components, and an open or closed workload generator. Experimental results show that this model is more flexible and accurate than Markov models. The model provides a useful simulation platform to study energy-saving strategies in wireless sensor networks.**

***Keywords*-Wireless Sensor Networks, Petri Nets, Markov Models, Modeling, Simulation, Minimizing Energy Consumption**

I. Introduction and Motivations

Wireless sensor networks are becoming increasingly prevalent in a wide range of areas from surveillance [1] to monitoring temperature, humidity, and other environmental parameters [2], [3], [4], [5]. A sensor network typically comprises of individual nodes operating with some limited computation and communication capabilities, and powered by batteries. Furthermore, these networks are situated in locations where they may not be easily accessible. As a result, there is a pressing need to have the sensor nodes operate for as long as possible to minimize maintenance and replacement costs. However, in order to propose methods by which power consumption can be minimized in wireless sensor networks, it is first necessary to gain an accurate understanding of their power consumption characteristics.

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 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 a 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* [6].

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 [7], [8]. Currently the two types of DVS systems available are those that stop executing while changing voltage and frequency and those that are capable of changing its operating parameters at run time [9], [10].

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. Specifically, this paper studies two methods of energy modeling: Markov chains and Petri nets. This paper makes the following contributions:

- We successfully model a processor using a Markov model based on *supplementary variables*; we also model a processor using Colored Petri nets. 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.
- We developed a model of a wireless sensor node that can accurately estimate the energy consumption using Petri nets. We used this model to identify the optimum *Power Down Threshold* for a given wireless sensor application.

The paper is organized as follows. Section II gives a short introduction to Markov models and Petri nets and discusses related work. Section III presents the CPU energy models and Section IV validate the CPU models. Section V demonstrates the effectiveness of Petri nets. Section VI presents a model for a wireless sensor node and Section VII uses this model to study the energy optimal *Power Down Threshold*. Section VIII concludes this paper.

II. Background and Related Work

A. Introduction to Markov models and Petri nets

Markov chains can be used for modeling, 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 the Markov property; meaning that the next state in a Markov model is only dependent upon the current state instead of previous ones. As will be demonstrated shortly, Markov models cannot be used to model transitions between states that require fixed deterministic intervals.

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 equations. However, the task of obtaining the equations relevant to the system can be difficult, if not impossible.

Petri nets, on the other hand, can be utilized to build statistical models of complicated systems that would otherwise be very difficult. The limitations of Markov models can be overcome using Petri nets. Petri nets is a simulation based approach for modeling using a directed graph of nodes and arcs referred to as places and transitions respectively.

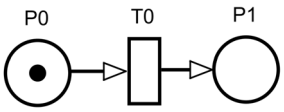


Figure 1. Example of Petri net

An example of a 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 token is enabled, it will fire according to different timing parameters. 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 models the behavior of a system by 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 of Petri nets. The software that is used in this study is TimeNet 4.0 [11].

B. Related Work

Many techniques have been proposed for modeling embedded systems and minimizing energy consumption. Jung et al in [12] 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 lifetime of nodes using these modes. 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 [13] 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.

Liu et al [6] 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 DVS modes.

In [14], we discuss the use of Petri nets to model a processor in a WSN. In this paper, we extend the use of Petri nets for modeling the WSN nodes as well. We attempt to view the minimization of energy from a systems standpoint rather than just focus on the CPU. Because the CPU is an integral part of the system, all the other parameters of the system effect the energy consumption of the CPU.

III. 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. We will review key ideas from [14] first, and then use them to develop models for sensor nodes. The following example illustrates the utilization of a Markov chain and Petri net to model the behavior of a CPU utilizing a simple power minimization philosophy. This example will allows us to compare the ease of use of both approaches.

A. Modeling Energy Consumption of CPU using a Markov model

In Figure 2 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 moves 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.

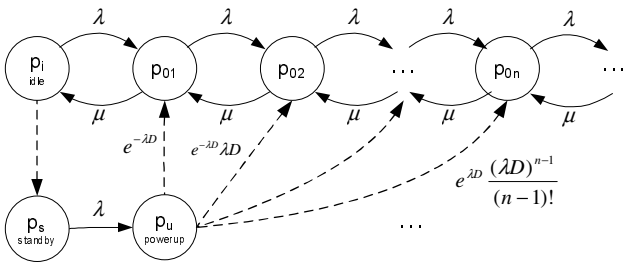


Figure 2. Birth-Death Process of CPU Jobs.

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 $\frac{1}{\mu}$.
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 2 follow exponential time distributions, the transitions shown as dashed lines are deterministic. 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 can not be modeled directly as a Markov process. Using the method of *supplementary variables* proposed in [15], we can derive an alternative set of state equations to *approximate* the transitions for stationary analysis. The derivation of these equations can be found in [14] and will not be repeated here.

The steady state probabilities of p_i, p_s p_u, and $G_0(1)$ are given by the following equations.

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

where $\rho = \frac{\lambda}{\mu}$.

$$p_i = \frac{(1-\rho)(e^{\lambda T}-1)}{e^{\lambda T} + (1-\rho)(1-e^{-\lambda D}) + \rho\lambda D} \tag{2}$$

$$p_u = \frac{(1-\rho)(1-e^{-\lambda D})}{e^{\lambda T} + (1-\rho)(1-e^{-\lambda D}) + \rho\lambda D} \tag{3}$$

$$G_0(1) = \frac{\rho(e^{\lambda T} + \lambda D)}{e^{\lambda T} + (1-\rho)(1-e^{-\lambda D}) + \rho\lambda D} \tag{4}$$

$$L(1) = \frac{\rho}{1-\rho}\frac{e^{\lambda T} + \frac{1}{2}(1-\rho)\lambda^2 D^2 + (2-\rho)\lambda D}{e^{\lambda T} + (1-\rho)(1-e^{-\lambda D}) + \rho\lambda D} \tag{5}$$

and the total energy consumption is

$$\begin{aligned} E &= (p_i P_{idle} + p_s P_{standby} + p_u P_{powerup} \\ &\quad + G_0(1) P_{active}) \frac{N + L(1)^2}{\lambda} \end{aligned} \tag{6}$$

where N is the total number of jobs. P_{idle}, $P_{standby}$, $P_{powerup}$, and P_{active} are the power consumption rates in the idle, standby, power up, and active states respectively.

B. 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 re-derived again. Petri nets on the other hand offer a more flexible approach.

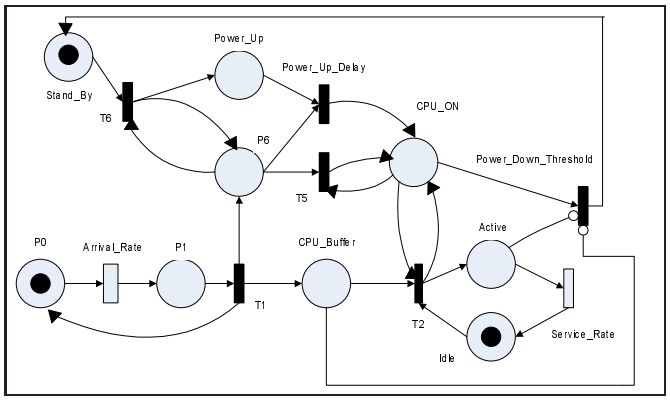


Figure 3. Petri net model of CPU

Table I
PETRI NET TRANSITION PARAMETERS FOR CPU JOBS

Transition	Firing Distribution	Delay	Priority
AR	Exponential	λ	NA
$T1$	Instantaneous	–	4
$T2$	Instantaneous	–	1
SR	Exponential	μ	NA
PDT	Deterministic	PDT	NA
$T5$	Instantaneous	–	2
$T6$	Instantaneous	–	3
PUD	Deterministic	PUD	NA

Figure 3 shows an open model of an Extended Deterministic and Stochastic Petri Net (EDSPN) [16] modeling a processor using the same energy minimizing behavior as 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 arrive. 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 $Power\_Down\_Threshold$, the CPU then moves to the "stand by" state ($Stand\_By$) to conserve power.

This model uses an open workload generator because when transition $Arrival\_Rate$ 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 I lists the parameters of all the transitions in the Petri net. The execution steps of this Petri net are detailed in [14].

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 $ArrivalRate$, $ServiceRate$, $Power\_Down\_Delay$, 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 Equation 7.

$$\begin{aligned} TotalEnergy &= (P_{standby} \times p_{standby} \\ &+ P_{powerup} \times p_{powerup} \\ &+ P_{idle} \times p_{idle} \\ &+ P_{active} \times p_{active}) \times Time \end{aligned} \tag{7}$$

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

IV. 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 7 will be used to compute the total energy for the simulator as well. The PXA271 Intel Processor whose power parameters are given in Table III [12] 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.

Table II
Simulation Parameters

Total Simulated Time	1000 seconds
Arrival Rate	1 per second
Service Rate	.1 per second

Figure 4 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 second. 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 4 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 effect the utilization of the CPU.

Table III
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

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 7 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 IV.

Figure 5 depicts the behavior of the CPU when the *Power_Up_Delay* is fixed at 0.3 second. Although, the Petri net model seems to over-estimate 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 8 shows the energy consumption estimates for each of the three methods. It is interesting to note that the Petri net energy estimates are now closer to the simulator results than the Markov model. As Table V 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.

Table IV
Δ Energy (Joules) Estimates ($Power\_Up\_Delay = 0.001$ second)

Power Down	Δ Sim-Markov	Δ Sim-Petri net	Δ Markov-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

Figure 6 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

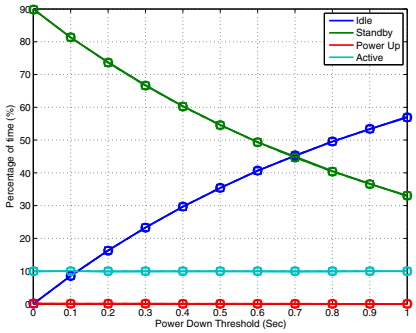


Figure 4. Power_Up_Delay = 0.001 Seconds

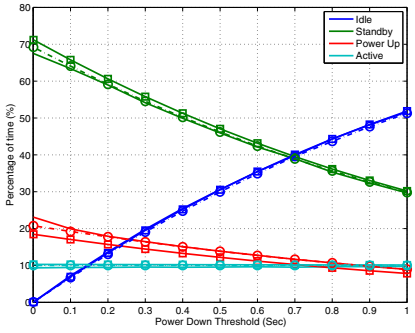


Figure 5. Power_Up_Delay = 0.3 Seconds

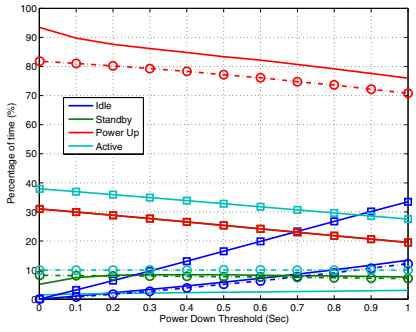


Figure 6. Power_Up_Delay = 10 Seconds

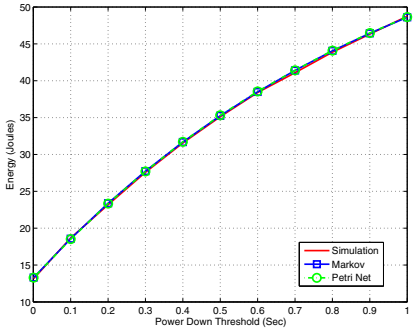


Figure 7. Energy Estimates for Power_Up_Delay = 0.001 Seconds

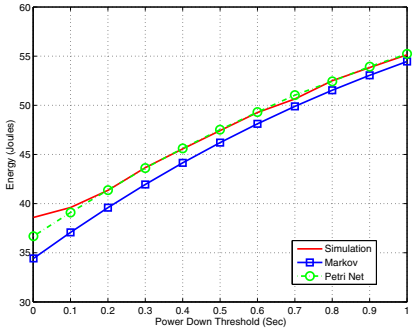


Figure 8. Energy Estimates for Power_Up_Delay = 0.3 Seconds

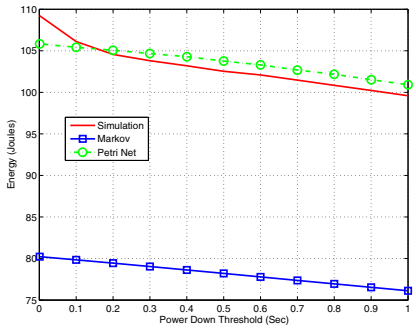


Figure 9. Energy Estimates for Power_Up_Delay = 10 Seconds

Table V
Δ ENERGY (JOULES) ESTIMATES ($Power\_Up\_Delay = 0.3$ SECOND)

Power Down	Δ Sim-Markov	Δ Sim-Petri net	Δ Markov-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 VI
Δ ENERGY (JOULES) ESTIMATES ($Power\_Up\_Delay = 10$ SECONDS

Power Down	Δ Sim-Markov	Δ Sim-Petri net	Δ Markov-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

the simulator results. The energy consumption comparison in Figure 9 and Table VI 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 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 9). 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.

V. 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 10 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 11(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

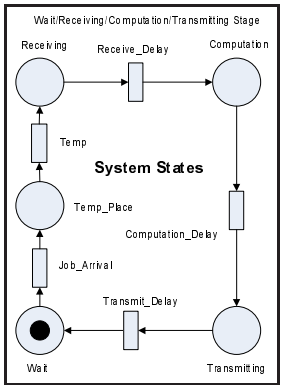
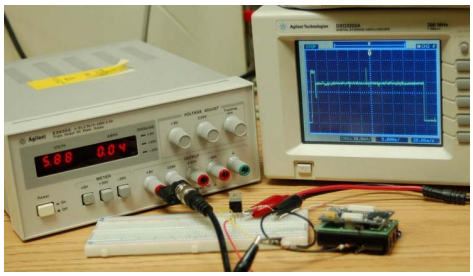
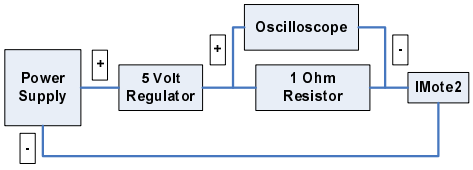


Figure 10. Simple system model of node in Wireless Sensor Network



(a) Setup of data collection for IMote2



(b) Block Diagram of data collection for IMote2

Figure 11. IMote2 power collection set-up

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 VII 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 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 non-idle 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

Table VII
MEASURED POWER REQUIREMENTS FOR DIFFERENT IMOTE2 STATES

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

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 X.

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 VIII lists the transitions in the Petri net, and the delays. Table IX lists the steady state probabilities of the places for the given transition parameters in Table VIII. Equation 8 was used to computed the energy consumption resulting from these probabilities. As Table X indicates, the actual energy consumed by the IMote2 and the energy predicted by the Petri net vary only by about 3 percent.

Table VIII
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 IX
STEADY STATE PROBABILITIES FOR A SIMPLE SYSTEM

State/Place	Probability (%)
$Wait$	59.8
$Tempplace$	19.7
$Receiving$	0.098
$Computation$	20.2
$Transmitting$	19.7

$$\begin{aligned} TotalEnergy &= (P_{Wait} \times (p_{Wait} + p_{Temp\_Place}) \\ &+ P_{Receiving} \times p_{Receiving} \\ &+ P_{Computation} \times p_{Computation} \\ &+ P_{Transmitting} \times p_{Transmitting}) \times Time \end{aligned} \quad (8)$$

VI. 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 both open and closed workload

Table XI
CLOSED SYSTEM MODEL PETRI NET TRANSITION PARAMETERS

Transition	Abbreviation	Type	Delay	Global Guard
$T0$	-	DET	AR	$(\#Wait > 0)$
$RadioStartUpDelay\_R$	$RSUD\_R$	DET	0.000194	$(\#P0 > 0)$
$RadioTurnOn$	RTO	INST (3)	–	$((\#Receiving > 0)\|\|(\#Transmitting > 0))$
$Channel\_Listening$	CL	DET	0.001	$((\#Receiving > 0)\|\|(\#Transmitting > 0))$
$Transmitting\_Receiving$	TR	DET	0.000576	NA
$Power\_Up\_Delay$	PUD	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)	–	$((\#Computation > 0)\|\|(\#Wait > 0))$
$Task\_Delay\_Per\_Job$	$TDPJ$	DET	0.000001	$\#Computation > 0$
$RadioStartUpDelay\_T$	$RSUD\_T$	DET	0.000194	$((\#TaskPerJob == 0)\&\&(\#Buffer == 0)\&\&(\#Idle > 0))$
$T19$		INST (3)	–	$((\#Buffer == 0)\&\&(\#Idle > 0)$ $\&\&(\#RJobsComplete == ComPackets))$
$Power\_Down\_Threshold$	PDT	DET	PDT	$((\#Buffer == 0)\&\&(\#Idle > 0))$
$Wait\_Transmitting$	WT	INST (3)	–	$(\#Transmitting > 0)$
$Wait\_Begin$	WB	INST (3)	–	$(\#Wait > 0)$

Table X
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

generators as shown in Figures 12 and 13. Generally, the behavior of nodes in a wireless sensor network follows the same basic pattern as described earlier. 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 sink. Finally, the node moves either to $Idle$ or $Stand\_By$, depending on the $Power\_Down\_Threshold$, 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 12 and 13 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 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 [9]. Tokens of different values result in different execution speeds simulating the change in the operating parameters.

In addition, our model implementation also utilizes 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 and this removes the need to provide connections using arcs. This simplifies the construction of the Petri net significantly.

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 $P0$ for the closed workload generator (Figure 12) or place $Event\_Arrival$ for the open workload generator (Figure 13). We use the processor and radio parameters as given in Table III [12] for the iMote2 sensor platform to provide realistic analysis.

A. Energy Model Using a Closed Workload Generator

Figure 12 demonstrates a Stochastic Colored Petri net (SCPN) [16] model of a sensor node using a closed workload generator. Global guards for the Petri net in Figure 12 are given in Table XI. The portion of the Petri net labeled "*Workload Generator*" generates the 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*".

A job is generated and placed in place $P0$. The system then moves from the "$Wait$" state to the "$Receiving$" state

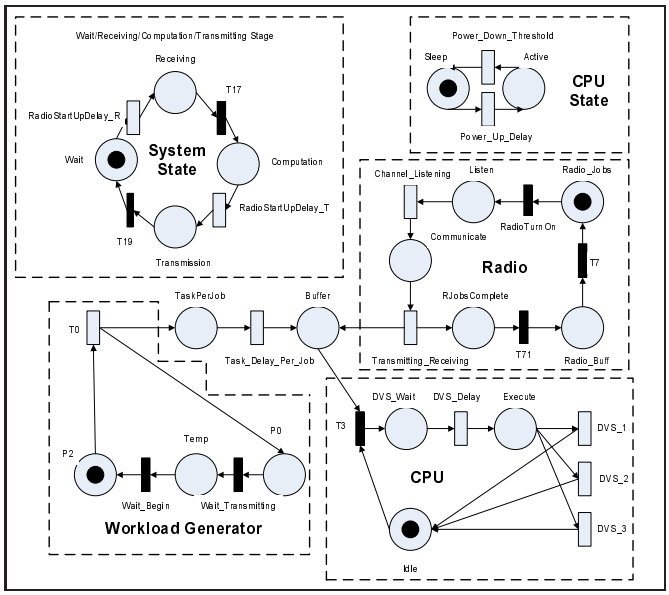


Figure 12. Closed system model of node in Wireless Sensor Network

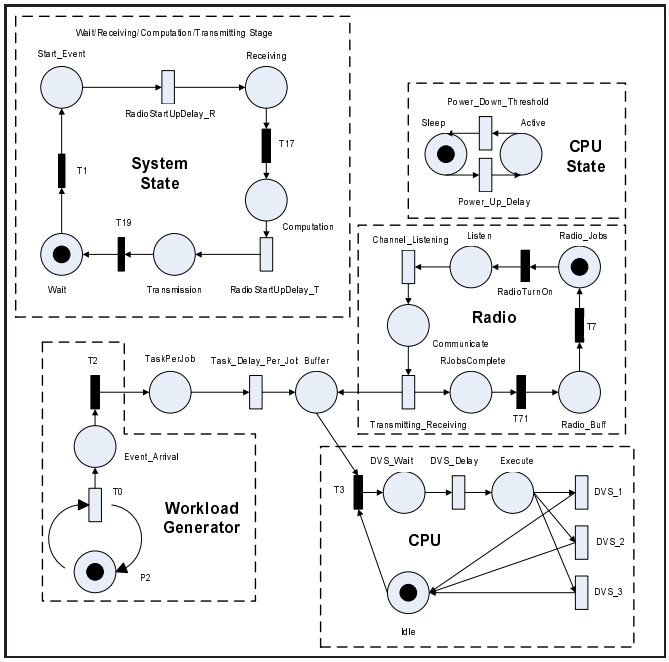


Figure 13. Open system model of node in Wireless Sensor Network

Table XII
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	$(\#Wait == 0)$

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 deposited in the CPU $Buffer$ for the purpose of awakening the CPU for error checking the received packet. Any tokens (jobs) in the $Buffer$ causes 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 by 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 system moves back to the $Wait$ state. The CPU $Sleep$/$Active$ states operates 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 VII.

B. Energy Model Using an Open Workload Generator

Figure 13 demonstrates a Stochastic Colored Petri net (SCPN) [16] 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 XI are also used here. The three additional transitions unique to this model are given in Table XII.

The main difference between the closed model (see Figures 12) and the open model (see Figure 13) is that events

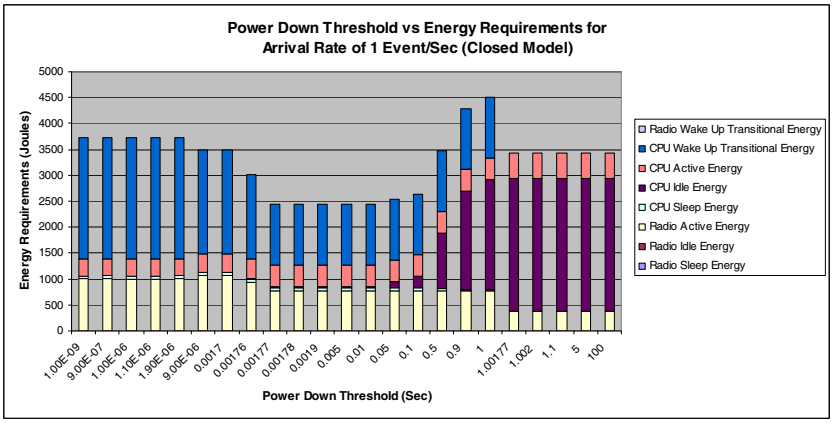


Figure 14. $PowerDownThreshold$ *vs* Energy Requirements for a closed model with a job arrival rate of 1 Event/Second

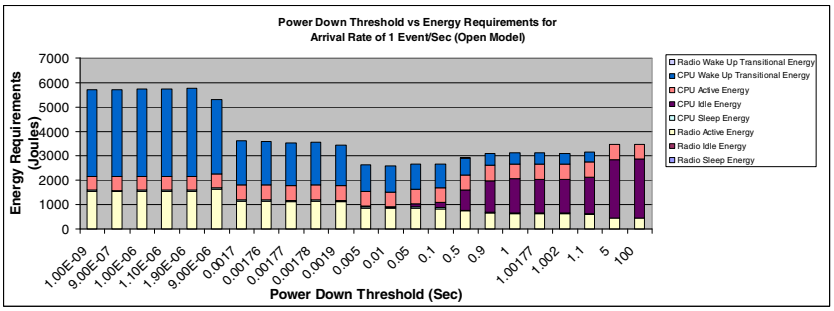


Figure 15. $PowerDownThreshold$ *vs* Energy Requirements for an open model with a job arrival rate of 1 Event/Second

arrive independently of the state of the system. When transition $T0$ fires randomly 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 nets in Figures 12 and 13 describes just two scenarios. Any other scenario 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 nets in Figures 12 and 13, 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 nets.

VII. 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.

A. Analysis Using Closed Workload

Figure 14 describes the energy characteristics of a wireless sensor node with a closed workload generator as $PowerDownThreshold$ increases. We will use our model to answer the question that was posed in Section I: what is the optimum $PowerDownThreshold$ that yields minimum energy consumption in a wireless sensor network?

In Figure 14, we see that powering down the CPU immediately after the computation does not result in the minimal energy consumption. Neither does always keeping the CPU "on" achieve the optimal energy efficiency. The optimum energy consumption of approximately 2432 Joules occurs at a $PowerDownThreshold$ of 0.00177 seconds. This is 35% less than the energy needed if the CPU is immediately powered down. This is also 29% less than the energy needed if the CPU is never powered down.

B. Analysis Using Open Workload

As Figure 15 shows, having the CPU power down immediately after processing a task is not beneficial for this system either. 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 $PowerDownThreshold$ is five seconds or more. When $PowerDownThreshold$ is approximately 0.01 seconds, the energy requirement is approximately 2589 Joules which is

almost 55% less than the energy needed if the CPU is shut down immediately after every task. This is also 26% less than the energy needed if the CPU is always "on".

In the event that a larger combination of transitions using deterministic and exponential distribution timing parameters is needed, the resulting system can be even more difficult to analyze with Markov chains. This indicates that Petri nets are a very important tool in modeling and analyzing these kinds of systems.

VIII. Conclusion

This paper proposed a detailed and flexible energy model for a CPU and a wireless sensor node using Stochastic Colored Petri nets. The experimental results indicate that the Petri net model of a CPU is more accurate than one based on Markov chains. This is due to the fact that a Markov model requires the modeled system 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 be modified.

We also showed that the energy estimates from a Petri net were within 3 percent of the actual energy needs of an IMote2 system triggered by random events.

We have also successfully demonstrate using our model that immediately powering down a CPU after every computation is not an energy minimal option. Nor is it energy efficient to leave the CPU "on" all the time. However, it is possible to identify a $PowerDownThreshold$ that results in large power savings. Through this example, we were able to show that this method of modeling is very useful and provides a valuable platform for energy optimization in wireless sensor networks.

However, one drawback of Petri net models is the relatively long simulation time to achieve steady state probabilities of complicated systems. The Petri net models in Figures 12 and 13 took an hour to stabilize. Depending on the desired accuracy, the simulation time can be even longer.

Acknowledgments

This work was supported in part by NSF Grants (GK-12 #0538457, IGERT #0504494, IIS #0916663, CCF #0937988, CCF #0621493, CCF #0754951, EPS #0904155).

References

[1] T. He, P. Vicaire, T. Yan, Q. Cao, G. Zhou, L. Gu, L. Luo, R. Stoleru, J. A. Stankovic, and T. F. Abdelzaher, "Achieving long-term surveillance in vigilnet," in *INFOCOM*, IEEE, 2006.

[2] J. Panchard and S. Rao, "Wireless sensor networks for applied research on rainfed farming in india: an exploratory user experiment," tech. rep., LCA, 2008.

[3] R. Cardell-Oliver, K. Smettem, M. Kranz, and K. Mayer, "Field testing a wireless sensor network for reactive environmental monitoring," tech. rep., University of Western Australia and Australian National University, 2004.

[4] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless sensor networks*, (Norwell, MA, USA), pp. 399–423, Kluwer Academic Publishers, 2004.

[5] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, (Atlanta, GA), September 2002.

[6] J. Liu and P. H. Chou, "Idle energy minimization by mode sequence optimization," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 4, p. 38, 2007.

[7] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," in *Proceedings of IEEE International Solid-State Circuits Conference*, Nov. 2000.

[8] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," pp. 76–81.

[9] G. Qu, "What is the limit of energy saving by dynamic voltage scaling?," in *ICCAD*, pp. 560–, 2001.

[10] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," in *IEEE Journal of Solid-State Circuits*, pp. 1571–1580, 2000.

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

[12] D. Jung, T. Teixeira, A. Barton-Sweeney, and A. Savvides, "Model-based design exploration of wireless sensor node lifetimes," in *Proceedings of the Fourth European Conference on Wireless Sensor Networks*, EWSN 2007, Jan. 2007.

[13] S. Coleri, M. Ergen, and T. J. Koo, "Lifetime analysis of a sensor network with hybrid automata modelling," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, (New York, NY, USA), pp. 98–104, ACM, 2002.

[14] A. Shareef and Y. Zhu, "Energy modeling of processors in wireless sensor networks based on petri nets," in *ICPPW'08: Proceedings of the 2008 International Conference on Parallel Processing - Workshops*, (Portland, Oregon, USA), pp. 129–134, IEEE Computer Society, 2008.

[15] 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.

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