A Unified Framework for System-level Design: Modeling and Performance Optimization of Scalable Networking Systems

Hwisung Jung, Massoud Pedram
Department of Electrical Engineering, University of Southern California
{hwijung, pedram}@usc.edu

Abstract

The need to bring high-quality systems to market at ever increasing pace is driving the use of system-level models early in the design process. This paper presents a new unified modeling framework, called the extended queuing Petri net (EQPN), which combines extended stochastic Petri net and G/M/1 queuing models, to realize the design of reliable systems during the design time, while improving the accuracy and robustness of power optimization for high-speed scalable networking systems. The EQPN model is employed to represent the performance behaviors and to minimize energy consumption of the system under performance constraints through mathematical programming formulations. Being able to model the system with the EQPN would enable the users to accomplish the design of reliable and optimized system at the beginning of design cycle. The proposed system model is compared with existing stochastic models under real simulation data. Experimental results demonstrate the effectiveness of the modeling framework and show that our proposed energy optimization techniques ensure robust system-wide energy savings under tight performance constraints.

1. Introduction

System modeling efforts should be pursued from both qualitative and quantitative aspects to elucidate an effective mechanism for controlling a complex electronic system under a set of performance constraints. Furthermore, realistic modeling of a system is an important step toward optimizing the performance and energy consumption of the system, since such a model-based design provides efficiencies in the design of complex system, which in turn enables designers to successfully realize the target system specification early in the design process. Typical scalable networking systems such as Ethernet controllers and switches have particularly tight time-to-market requirement, which requires a highly efficient design cycle. The design of reliable, low-energy, and high performance systems is best accomplished when the system modeling is done with careful consideration of the performance constraints [3].

A look at today's high-speed networking system trends reveals that multiple CPUs reside within a single system to satisfy the high-functionality, high-performance demands of today's applications. However, this trend also translates into high-power densities, higher operating temperatures, and reduced reliability. Furthermore, the ability of the networking protocol of operating system to scale well on a multi-CPU system is becoming requisite to eliminate the receive processing bottleneck. Thereof, receive-side scaling (RSS), a result of Microsoft's scalable networking initiative [2], supports dynamic load balancing to scale with the number of available CPUs in the system, results in parallelism in packet receive processing [1].

As evidenced in the recent literature [4][5][6][7], increasing interest has focused on the realization of target system specification. Stochastic Petri net (SPN) and queuing network (QN) formalisms have been used to model the execution of tasks under precedence constraints in systems that exhibit concurrency, synchronization, and randomness. The work presented in [4] studies the Petri net-based performance modeling with a generalized stochastic Petri net (GSPN) for distributed homogeneous system. In reference [5][6], the authors propose the stochastic modeling of a power-managed system based on the GSPN and continuous-time Markov decision process (CTMDP). Performance modeling based on the timed Petri net for multi-threaded multiprocessors is studied in [7] with the assumption that traffic on links of interconnect network is uniformly distributed.

Although these techniques can handle complex behavior such as concurrency, synchronization, and heterogeneity, GSPN and CTMDP have difficulty capturing the scheduling strategies. In addition, CTMDP only permits exponential distribution functions for the inter-arrival and service times, which are not realistic assumption when modeling a complex real-time system. One way to facilitate the scheduling problem formulation is to combine Petri net models with a queuing network, which provides a mechanism for modeling resource contention and scheduling aspects. The combination of Petri net and queuing network, so-called queuing Petri net, was originally proposed by Falko [8] to model systems with GSPN and M/M/1 models. However, this modeling framework still allows only an exponential distribution function for the inter-arrival times of the system, which continues to be a major shortcoming since an exponential distribution underestimates the occurrence probability of a large request inter-arrival time and so it does not adequately model the request arrival times in the idle periods [9].

In this paper, we overcome these shortcomings by introducing a new unified modeling framework, the extended queuing Petri net (EQPN). EQPN model, characterized by a general distribution function for inter-arrival times, provides a way of optimizing the performance and power consumption. The numerical solution for EQPN is based on a semi-Markov decision process (SMDP). This paper presents a systematic approach for evaluating EQPN under performance constraints. Mathematical programming formulations and dynamic voltage and frequency scaling algorithm are employed to realize more effective solutions during the modeling and performance optimization of scalable networking system. This is precisely the contribution of the present paper.

The remainder of this paper is organized as follows: Section 2 provides a background of the scalable networking system. The details of the modeling framework are given in section 3. Section 4 presents performance optimization techniques. Experimental results and conclusion are given in section 5 and section 6.

2. Background

In this section, we present a background of the scalable networking system briefly, which includes a new Network Driver Interface Specification (NDIS) technology called receiver-side scaling (RSS).

Figure 1 shows a simplified block diagram of the scalable networking system, where the host system uses Ethernet controller, i.e., network interface card (NIC), to send and receive packets. Sending and receiving packets over the local interconnect, e.g., PCI-E bus [19], is handled by the NIC and device driver in the host operating system. In general, NIC typically has a direct memory access (DMA) engine to transfer data to the host system, In addition, NIC includes a medium access control (MAC) unit to implement the link level protocol for the underlying network, and use a signal processing hardware to implement the physical (PHY) layer defined in the network. Details of the NIC architecture and functionality are omitted here to save space. Interested readers may refer to [14].

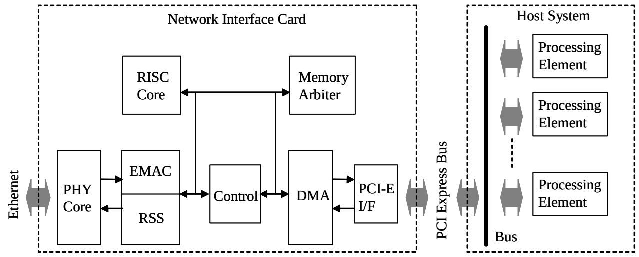

Figure 1. Block diagram of scalable networking system.

In the scalable networking, RSS implemented inside NIC provides dynamic load balancing, while preserving in-order delivery of messages on a per-stream basis. Specifically, RSS enables packets from a single NIC to be processed in parallel on multiple processing elements (e.g., CPUs), while preserving in-order delivery to transmission control protocol (TCP) connections. Furthermore, RSS enables parallel deferred procedure calls, queued in the system in first-in first-out (FIFO) manner, and multiple interrupts if the host system supports it.

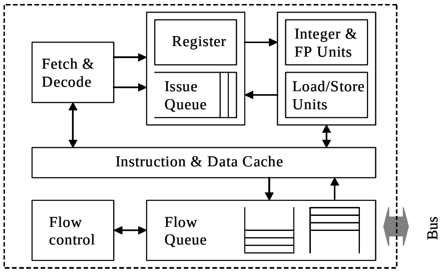

Figure 2. Architecture of processing element.

Figure 2 shows architecture of processing element (PE) in the scalable networking system which improves the processing time by executing multiple threads in parallel across multiple processing elements. A thread consists of a sequence of instructions or data. Each processing element has its own cache hierarchy (i.e., L1 and L2 caches). The data packets are distributed to multiple processing elements by RSS which rebalance the network processing load, while preserving in-order delivery. In this paper, we consider in-order processor architecture to simplify the modeling and analysis steps. Interested readers for the details of multiple processing element architecture may refer to [10][11].

3. A Unified Modeling Framework

In this section, we present a systematic approach for modeling a scalable networking system by introducing a unified modeling framework, i.e., the extended queuing Petri net (EQPN). We focus on the model of the host system shown in Figure 1 to simplify the modeling step, assuming that the data packets are distributed to multiple processing elements by RSS inside NIC. It will not hurt the quality of the paper if we concentrate on the host system side of scalable networking for modeling since it sufficiently exhibits concurrency, synchronization, and heterogeneity behaviors.

3.1 Extended Queuing Petri Net Model

To construct the model of the host system, we first consider the functionality of a PE. The Figure 3 shows the extended stochastic Petri net (ESPN) model [15], an extension of stochastic Petri net, for the PE. Note that an ESPN is formally defined as a six-tuple (P, T, E, M, F, G) such that i) $P = \{P_1 , P_2 , \ldots, P_n \}, n >0$ and is a finite set of places, ii) $T = \{T_1 , T_2 , \ldots, T_n \}, n >0$ and is a finite set of transitions, iii) E is a set of arcs known as a flow relation, iv) $M: P \rightarrow N$ and is a marking, where N is the number of tokens, v) F is the set of firing rates associated with the transitions, and vi) $G: F(M) \times T \rightarrow F$ is a firing function which is associated with an arbitrarily distributed firing time. Pictorially speaking, immediate transitions are drawn as thin bar and timed transitions as thick bar.

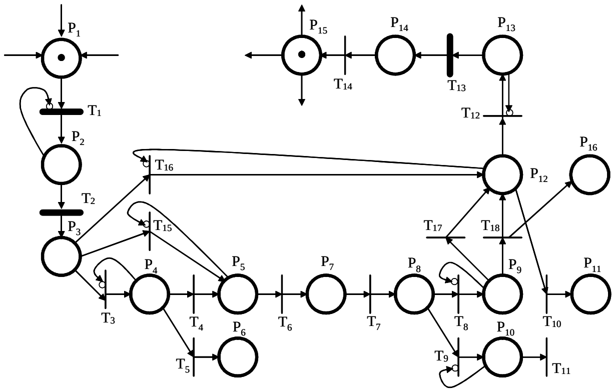

Figure 3. ESPN modeling for processing element.

A job advances in the architecture as follows. First, the inbound switching (i.e., load balancing) is performed at the NIC during P_1. After switching, the incoming traffic is written at the flow queue at P_2. If we consider the instruction fetch at P_4, the processing of fetch is modeled by transition T_4 and T_5 representing cache hit and miss, respectively. We simplify the ESPN model by omitting the detail procedure of cache miss handling. After the instruction cache access at P_5, the PE performs the instruction decoding P_7 and the issue queuing P_8, consecutively. The decode logic takes instructions from the flow queues and decodes them into micro-operations. The execution of data is modeled by transitions T_8 and T_9 with places P_9 and P_{10} representing data memory access and integer / float-point unit access, respectively. If data memory access instruction is presented, either data cache hit or miss occurs, resulting in data cache access at P_{12}. Then, processor needs the cache coherency between L1 and L2 cache if L1 cache is updated by transferring the data through outbound queue at P_{13}. Finally, the stored data at the queue are given to L2 through the outbound switch at P_{15}. In this ESPN model, we consider the write-back scheme as cache update strategy. The definitions for the various places and transitions for this model are summarized in Table 1.

Table 1. Legend for the ESPN model of Figure 3.

Place	Description
P_1	Inbound switching
P_2	Inbound flow queue writing
P_3	Writing done
P_4	Instruction fetch
P_5	Instruction cache accessing
P_6	Instruction cache miss handling
P_7	Instruction decode
P_8	Issue queuing
P_9	Memory inst. executing
P_{10}	Integer & FP unit accessing
P_{11}	Retirement
P_{12}	Data cache accessing
P_{13}	Outbound flow queue writing
P_{14}	Writing done
P_{15}	Outbound switching
P_{16}	Data cache miss handling

Trans	Description
T_1	Incoming switching delay
T_2	Queuing delay
T_3	Immediate transition
T_4	Immediate transition (cache hit)
T_5	Immediate transition (cache miss)
T_6	Immediate transition
T_7	Immediate transition
T_8	Memory access
T_9	Integer & FP unit access
T_{10}	Immediate transition (reg. update)
T_{11}	Immediate transition (reg. update)
T_{12}	Immediate transition
T_{13}	Queuing delay
T_{14}	Immediate transition
T_{15}	Inst. cache update
T_{16}	Data cache update
T_{17}	Immediate transition (cache hit)
T_{18}	Immediate transition (cache miss)

The ESPN model described in Figure 3 can be considered as a controlled Petri net. This is because the movement of tokens is influenced by external enabling conditions called control places, not shown here. For example, the transitions, T={T_4, T_5, T_{10}, T_{11}, T_{17}, T_{18}}, are governed by the control blocks. Since the tokens in the control places are consumed by the firing of controlled transitions in a similar manger to the state places, we omitted the control places to simplify the ESPN model.

Due to the incoming traffic at the NIC, the inbound data are queuing at the flow queue to be fetched into the processor when it is available. Queuing and scheduling mechanisms at the NIC enable the processor to handle the resource contention at the interconnect network. To facilitate the queuing strategy, we use a new modeling formalism, extended queuing Petri net (EQPN). The main idea in the creation of EQPN is to allow the queue to be integrated into the places of an ESPN. An EQPN is defined as follows:

Definition 1: Extended Queuing Petri Net. An EQPN is a triplet *(ESPN, P_Q, W)* with a substitution in *M*, where

1) ESPN is the underlying Extended Stochastic Petri Net.
2) P_Q = {P_{Q1}, P_{Q2}}, where $P_{Q1} \subseteq P$ is the set of timed queuing places and $P_{Q2} \subseteq P$ is the set of immediate queuing places.
3) W = {W_1, W_2}, where $W_1 \subseteq T$ is the set of timed transitions and $W_2 \subseteq T$ is the set of immediate transitions.

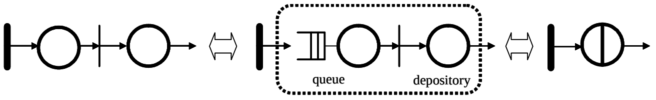

Figure 4. A queuing place and its schematic symbol.

EQPN contains a new type of place, called a *queuing place*. Such a place consists of two components: a queue and a depository [8]. The token that has completed its service at the place, i.e., the queue, immediately moves to the next place, i.e., the depository, as depicted in Figure 4.

Combining the queue network and the ESPN now yields the EQPN model as depicted in Figure 5. In this work, we use the G/M/1 queuing model, where the inter-arrival times are generally distributed and service times are exponentially distributed, which is more realistic model for an actual system than the M/M/1 queuing model used in [8].

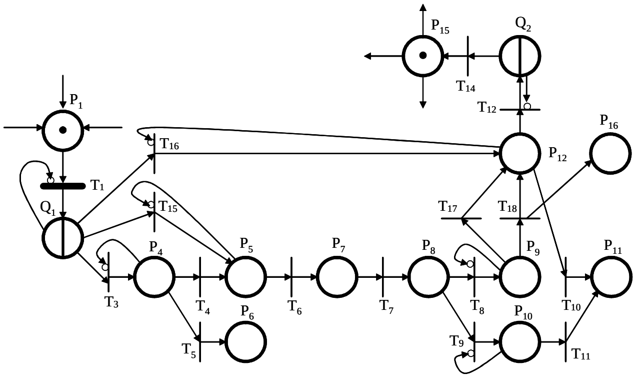

Figure 5. Corresponding EQPN model.

Now that we have obtained the EQPN model of the given system, we can convert it into a semi-Markov decision process (SMDP) model by using the reachability graph, which consists of reachable markings and arcs corresponding to transition firings. Note that a SMDP is a tuple <*S, A, Y, Z, R*>, where *S* is a set of states, *A* is a set of actions, *Y* is the transition probability function, *Z* specifies the probability distribution of transition times for each state-action pairs, and *R* is the expected reward function [12]. Reachability graph, which can be obtained by the reachable marking table, contains two types of markings: vanishing and tangible [13]. A vanishing marking is one that enables only immediate transitions. A tangible marking, denoted by *g* where $g \in M$, is described by the marking process as a function of the time *t*. Notice that an underlying EQPN is a type of semi-Markov process as described in theorem 1.

Theorem 1: The marking process {$g(t),\ t \geq 0$} of an EQPN is equivalent to a semi-Markov process with discrete state space [13].

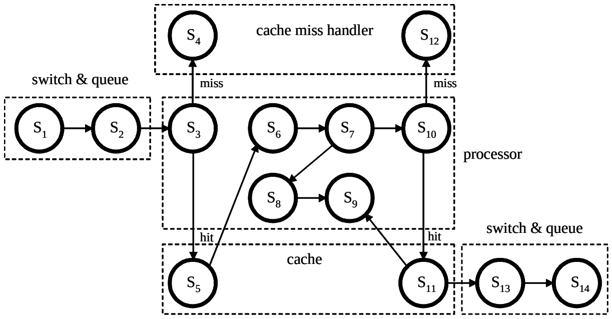

Figure 6. SMDP model of the system.

A SMDP model is constructed from the EQPN model by adding the decision dimension and by removing the vanishing markings from the reachability graph as shown in Figure 6. The system is thus modeled as the SMDP with a state set **S** = {S_1, S_2, …, S_m}, where m is the number of processing modes available to the system. A state represents a marking of the EQPN.

3.2 Analysis of EQPN Model

Regarding the queuing model in the EQPN, as shown in Figure 5, the service time behavior is captured by a given service time distribution for the EQPN when the system (e.g., PE) is in the active state. Similarly, the input request behavior is modeled by some inter-arrival time distribution.

We first analyze the characteristics of the queuing model in the SMDP derived from the EQPN model. Let S_i represent the ith state in the SMDP, and let I_i denote the task (data in the scalable networking) inter-arrival time whose distribution depends only on the present state S_i. Assuming that the inter-arrival times are mutually independent, we can define the arrival process of tasks at time t from state i to state j of the SMDP as follows: [1]

$$a_{ij}(t) = Prob\{S_{i+1} = j,\ I_{i+1} \le t \mid S_i = i\} \tag{1}$$

Let W denote the number of waiting tasks in the PE just before a new task arrives. Assuming an infinite queue size, we have

$$q_n \triangleq Prob\{W = n\} = (1-\gamma)\gamma^n\ ,\ n = 0, 1, ..., \infty \tag{2}$$

where γ is the unique solution of Laplace-Stieltjes transform (LST) of the inter-arrival time distribution function [16]. We assume that the service times in the PEs are exponentially distributed with the mean value of μ^{-1}. Let $T_{W.k}$ and $T_{S.k}$ represent the *waiting time* and the *service time* of the tasks in the k^{th} PE. Since the *response time*, $T_{R.k}$, of the PE is the expected time that the tasks spend in the queue and in the PE, $T_{R.K} = ((1 - \gamma)\mu)^{-1}$. The waiting time spent in the queue is obtained by subtracting the service time from the response time. This yields

$$T_{W.k} = T_{R.k} - \frac{1}{\mu} = \frac{\gamma}{\mu(1-\gamma)} \tag{3}$$

Regarding the traffic in the scalable networking, we consider the utilization of a PE i.e., how much of the computational resource provided by the PE has been employed by the application. The utilization ratio, u_{PE}, is thus defined as the fraction of time that the PE is working, which can be calculated as

$$u_{PE} = BP \big/ (BP + IP) \tag{4}$$

where BP is the duration of the busy period of the PE, and IP is the duration of its idle period. Without presenting the proof, we simply state (cf. [16]):

$$BP + IP = E(T) \big/ (1-\gamma) \tag{5}$$

where $E(T)$ is the expected number of transitions in the SMDP. Thus, considering the number, n, of tasks waiting in the queue, we can calculate BP and IP as follows

$$BP = \frac{E(T)}{1-\gamma} \cdot \sum_{i=1}^{n} q_i\ , \quad IP = \frac{E(T)}{1-\gamma} \cdot q_0 \tag{6}$$

The utilization of the PE is an important factor since it affects the latency and bandwidth of scalable networking system.

As a performance metric, we use the link utilization (LU), a direct measure of traffic workload, to find the under-utilized PE as follows.

$$LU(e) = \sum\nolimits_c^G D(e,c) / N_{clk}\, , \qquad 0 \le LU \le 1 \tag{7}$$

where $D(e,c) = \begin{cases} 1 & \textit{if traffic passes on link e at cycle c} \\ 0 & \textit{otherwise} \end{cases}$, G is the required number of clock cycles at the given frequency for the PE, e is the link path between NIC and PE, and N_{clk} is the number of clock cycles of the link given to the PE.

4. Performance Optimization

In this section, we present the performance (e.g., energy) optimization formulation based on SMDP by developing mathematical programming models.

Let $actpow_{k.\pi}$ and $slpow_k$ represent the power consumption incurred in the k^{th} PE during its active mode running at optimal policy π and during its sleep mode, respectively. Basically, a particular policy tells the system what action (e.g., DVFS) to perform in order to optimize the performance. Incorporating with the SMDP model, we obtain the optimal policy π for power optimization as follows. First, the expected power of the system (i.e., PE) in active mode can be defined as

$$pow_{exp}(s,a) = pow(s) + \frac{1}{\tau(s,a)} \sum_{s' \in S} Prob(s' \mid s, a) ene(s, s') \tag{8}$$

where $pow(s)$ is the power consumption of the system in state s, $Prob(s' \mid s, a)$ is the probability of being in state s' after action a in state s, $ene(s, s')$ is the energy required by the PE to transit from state s to s', and $\tau(s, a)$ is the expected duration of time that the system spent in state s if action a is chosen. Second, let a sequence of states $s^0, s^1, \ldots, s^k$ denote a processing path δ from s^0 to s^k of length h with the property that $p(s^0, s^1), \ldots, p(s^{k-1}, s^k) > 0$, where $p(x, y)$ is the probability that the system moves to state y from state x. For a policy π, we can define the discount cost C of a processing path δ of length h as follows.

$$C^{\pi}(\delta) \triangleq \sum\nolimits_{i=0}^{h} \gamma^{t_i} pow_{exp}(s^i, a^i) \tag{9}$$

where γ is a discount factor, $0 \le \gamma < 1$, and t_i is the time that the system spent in state s^i before action a^i causes a transition to state s^{i+1}. Considering the expectation with respect to the policy π over the set of processing path starting in state s, we can define the expected cost of the system, given that the system starts in state s by

$$actpow^{\pi}_{avg}(s) = EXP[C^{\pi}(\delta)] \tag{10}$$

With the above-mentioned variables and their characteristics, we can write the linear programming as

$$\begin{aligned} \min \quad & \sum\nolimits_{s} \sum\nolimits_{a} actpow^{\pi}_{\text{avg}}(s) \varphi(s,a) \\ \text{s.t.} \quad & \sum\nolimits_{a} \varphi(s,a) - \sum\nolimits_{s'} \sum\nolimits_{a} \varphi(s',a) Prob(s' \mid s, a) = 0 \\ & \sum\nolimits_{s} \sum\nolimits_{a} \varphi(s,a) \tau(s,a) = 1 \\ & \varphi(s,a) \ge 0 \qquad all\ s \in S, a \in A \end{aligned} \tag{11}$$

[1] In this paper, subscripts denote state information whereas superscripts denote time stamp.

where $\varphi(s, a)$ is the frequency that the system is in state s and action a is issued. Note that the optimal solution to the SMDP policy optimization problem belongs to the set of deterministic policy [12].

Now that we have obtained the policy π for optimal active power consumption, the total energy dissipation of the PE is defined as

$$ene_{total.k} = actpow_{avg}^{\pi}(s) \cdot \sum_{l \in L} exe_{l.k.\pi} + slpow_{k}(T_{d} - \sum_{l \in L} exe_{l.k.\pi}) \quad (12)$$

where $exe_{l.k.\pi}$ is the execution time of task l, where $l \in L$ (set of tasks), running under policy π at the k^{th} PE, T_d is the given deadline. Changing the voltage level (and correspondingly the operating frequency) of the PE affects the execution time of the tasks. Clearly, $exe_{l.k.\pi} = T_{W.k.\pi} + T_{S.k.\pi}$. When there is a positive slack for the task to run on a PE, application of DVFS can result in significant energy saving. Thus, our goal is to minimize energy consumption for a PE, subject to performance constraints as follows.

$$\begin{aligned} \min \quad & ene_{total.k} \\ \text{s.t.} \quad & \sum\nolimits_{i=1}^{n} q_i \,/ \sum\nolimits_{i=0}^{n} q_i \geq u_k \\ & \sum\nolimits_{\mathrm{i=1}}^{\mathrm{n}} i \cdot q_i + T_{S.k.\pi} \leq T_d \\ & \sum\nolimits_{i=0}^{n} q_i = 1 \\ & 0 \leq q_i \leq 1, \quad i = 0, \ldots, n \end{aligned} \qquad (13)$$

where $\sum_{i=1}^{n} i \cdot q_i = T_{W.k.v}$, $T_{S.k.v}$ is affected by the DVFS setting, $BP/(BP + IP) = \sum_{i=1}^{n} q_i / \sum_{i=0}^{n} q_i$, and u_k is a lower bound on the PE utilization, which is provided by the developers.

5. Experimental Results

In the experimental setup, we use a network interface card (NIC), i.e., Ethernet controller [20], which includes receiver-side scaling (RSS) capabilities for scalable networking, to obtain the real-application data used to evaluate our EQPN models. Table 2 shows the performance characteristics of the Ethernet controller, obtained by measuring the throughput for streams of various packet sizes, where we use SmartBits 2000 (performance analysis system) [21] to generate various packet streams. Note that we fix the IP packet size in the simulation with the inter-packet gap set to 0.0096us.

Table 2. Performance characteristics of Ethernet controller.

Packet size (bytes)	Service rate (pkt/sec)	Inter-arrival time (sec)	Arrival rate (pkt/sec)	Service time (sec)
1518	84819	12.2E-6	81699	11.7E-6
1024	124936	8.28E-6	120656	8.00E-6
512	245100	4.19E-6	238549	4.08E-6
256	317400	2.14E-6	466417	3.15E-6
128	325200	1.12E-6	892857	3.07E-6
64	338000	0.60E-6	1644736	2.95E-6

The first experiment is designed to demonstrate the effectiveness of our proposed EQPN model as shown in Figure 7 by calculating the number of tasks and comparing these results with both the actual data obtained from real-applications as shown in Table 2 and those obtained based on GSPN model (which assumes an exponential distribution for inter-arrival times). The parameter values for the EQPN and GSPN models, which depend on the arrival rate and the mean service time as variables, are obtained by using the Queue2.0 [18] simulation tool. Note that all values are normalized to the actual data. For example, when the arrival rate is 0.54 and the mean service time is 1.2, the EQPN model gives around 0.3 for the number of task, similar to the actual data, whereas the GSPN model yields around 0.4. We point out that the EQPN model gives accurate performance results compared to the actual data while the GSPN model overestimates the performance loss.

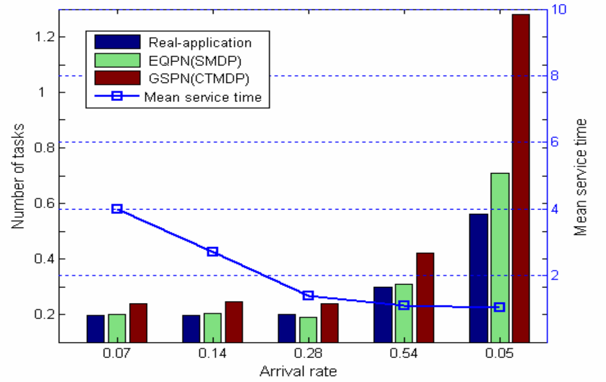

Figure 7. Evaluation of EQPN for the various numbers of tasks.

In the second experiment, we evaluate the proposed performance optimization methods. Considering the processing element, we use the system specification model from UltraSPARC-II*i* processor [17], which consumes 17.6W active power dissipation at 1.7V, 650MHz and 20mW power for sleep mode. It also supports 1/2 and 1/6 clocking frequencies for power-saving modes. Our goal is to minimize the energy consumption of the PE under following scenario:

Scenario: Processing element can accept two streams of data (i.e., tasks): low-priority data and high-priority data with service mean $\mu^{-1} = 1$. We use non-preemptive priority policy [16], where a high-priority data can move ahead of all the low-priority data waiting in the queue.

Different arrival rates of high-priority data are used to generate the multiple rows in Table 3. We set that $T_d = 9$ and $u_k = 0.8$ as performance constraints, and assume that the arrival rate of low-priority data = 0.5 and the service mean $\mu^{-1} = 1$. PE has three operating voltage/frequency levels: 1.7V/650MHz, 1.6V/325MHz and 1.5V/108MHz. For example, in the first row in Table 3, it takes 3.64 times (=2366 clock cycles @650MHz) to proceed both high and low priority data, which consumes 64.1 energy dissipation in the original model (i.e., not using sophisticated energy optimization techniques), including the sleep power, whereas the PE achieves 56.8 energy consumption if running at 325MHz. Note that all values are normalized. Table 3 presents the comparison result by simulation, where the values of frequency $\varphi(s, a)$ are calculated with equation (11). The results report that SMDP-based optimization method can adjust the optimal policy when workload characteristics change, giving the consistent energy dissipation. It also indicates that further reductions in energy savings can be achieved using the SMDP-based optimization when we increase the arrival rate of high-priority threads.

Table 3. Experimental results for SMDP-based optimization.

	Original model			SMDP-based Optimization				
Arrival Rate of High-Priority Data	Waiting Time at High-Priority Queue	Waiting Time at Low-Priority Queue	Energy	Frequency that the system is in state *s* and action *a* (high-priority case) [φ (s, a_1) φ (s, a_2) φ (s, a_3)]	Power for High-Priority Data	Power for Low-Priority Data	Energy	Energy Saving
0.02	0.53	1.11	64.1	0.61 0.31 0.10	13.1	6.6	62.1	3.3%
0.04	0.56	1.22	66.5	0.59 0.28 0.10	12.9	6.0	65.3	1.8%
0.06	0.60	1.35	69.6	0.55 0.27 0.09	11.9	5.6	64.8	6.9%
0.08	0.63	1.50	72.7	0.53 0.26 0.06	11.4	4.8	65.1	10.5%
0.10	0.67	1.67	76.4	0.49 0.25 0.08	10.8	4.1	64.9	15.1%

The third experiment is to demonstrate the robustness of our proposed energy optimization method in the case of dynamic traffic workload on the link between the NIC and PE. To simplify the experimental setup, we assume that the PE accepts only one type of data, and link utilization (LU) on the link between the NIC and PE is changed to 0.7 from 1. Table 4 shows the comparison results for energy consumption in the PE for the cases: 1) LU = 1.0 in the original model, 2) LU = 0.7 in the original model, and 3) LU = 0.7 in our proposed model. Notice that when the arrival rates of data are 0.9 and 0.8, the optimization method is not applicable since it exceeds the response time, which incur performance degradation. From the table, we can see that the proposed method ensures system-wide energy savings (up to 11.5%) with the enhanced utilization factor when the traffic workload is changed by dynamic load balancing.

Table 4. Experimental results for LU-based optimization.

When LU = 1.0			When LU = 0.7							
Original model			Original model				LU-based Optimization			
Arrival Rate of Data	Response Time T_R	Energy	Response Time T_R	Busy + Idle Period	Util. of PE	Energy	Response Time T_R	Util. of PE	Energy	Energy Savings
0.9	7.65	135.0	1.52	2.40	0.63	26.6	1.52	0.63	26.6	0%
0.8	3.10	53.9	1.22	2.18	0.56	21.5	1.22	0.56	21.5	0%
0.7	2.32	41.0	1.02	2.08	0.49	17.9	2.04	0.98	15.9	11.4%
0.6	1.71	29.9	0.89	2.12	0.42	15.7	1.78	0.83	13.9	11.5%
0.5	1.37	24.3	0.79	2.21	0.35	13.9	1.58	0.71	12.3	11.4%
0.4	1.17	20.6	0.74	2.64	0.28	13.0	1.48	0.56	11.6	11.4%
0.3	1.06	18.7	0.71	3.39	0.21	12.5	1.42	0.42	11.2	11.4%
0.2	1.01	17.9	0.70	5.00	0.14	12.4	1.40	0.28	10.9	11.3%
0.1	0.99	17.7	0.70	9.99	0.07	12.5	1.40	0.14	11.1	11.2%

6. Conclusion

We proposed a new unified modeling framework for the system-level design, called the extended queuing Petri net (EQPN), to improve the modeling accuracy of scalable networking system. By modeling the system with EQPN, the parameters in the model become more realistic which allows state durations to have arbitrary distribution. Performance optimizations based on SMDP models are formulated under performance constraints to ensure system-wide energy savings. Experimental results with real-application show that or model gives accurate performance results and has better performance than previous models. The EQPN model clearly enables the user to accomplish the design of reliable and optimized electronic system under performance constraints, early in the design cycle.

References

[1] http://www.sun.com/processors/UltraSPARC-T1/ Technical document.

[2] http://www.microsoft.com/whdc WinHEC 2004 version, Apr. 2004.

[3] L. Spracklen, and S.G. Abraham, "Chip Multhreading: Opportunities and Challenges," *Proc. of 11th HPCA*, Feb. 2005.

[4] N. Lopez-Benitez, "Petri-Net Based Performance Evaluation of Distributed Homogeneous Task Systems," *IEEE Trans. on Reliability*, Vol. 49, No.2, June 2000.

[5] Q. Qiu, Q. Wu, and M. Pedram, "Stochastic Modeling of a Power-Managed System - Construction and Optimization," *IEEE Trans. on Computer-Aided Design*, Vol. 20, No. 10, Oct. 2001.

[6] Al Bogliolo, L. Benini, E. Lattanzi, and G. D. Micheli, "Specification and Analysis of Power-Managed System," *Proc. of the IEEE*, Vol. 92, No. 8, Aug. 2004.

[7] W.M. Zuberek, R. Govindarajan, and F. Suciu, "Timed Colored Petri Net Models of Distributed Memory Multithreaded Multiprocessors," *Proc. of Workshop on Colored Petri Nets and Design*, June 1998.

[8] F. Bause. "Queuing Petri Nets," *In Proc. of the 5th Intl Workshop on Petri Nets and Performance Models*, Toulouse, France, 1993.

[9] T. Simunic, et al., "Event-driven power management," *IEEE Trans. On Computer-Aided Design,* Vol.20, No. 7, Jul. 2001.

[10] D. Bertozzi, and L. Benini, "Xpipes: A Network-On-Chip Architecture for Gigascale Systems-on-Chip," *IEEE Magazine*, 2nd quarters, 2004.

[11] R. Kumar, et al., "Single-ISA Heterogeneous Multicore Architecture: The Potential for Processor Power Reduction," *Proc. of 36th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, Dec. 2003.

[12] M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Publisher, New York, 1994.

[13] J. Wang, *Timed Petri Nets: Theory and Application*, Kluwer Academic Publishers, 1998.

[14] Y. Hoskote, et al, "A TCP Offload Accelerator for 10Gb/s Ethernet in 90nm CMOS," *IEEE Journal of Solid-State Circuits*, Vol. 38, No. 11, Nov. 2003.

[15] J. Dugan, K. Dtrivedi, R. Geist, and V. Nicola, "Extended stochastic Petri nets: applications and analysis", *Proc. of Performance*, Dec. 1984.

[16] D. Gross, and C.M. Harris, *Fundamentals of Queuing Theory*, Wiley, 3rd edition, 1998.

[17] http://www.sun.com/processors/UltraSPARC-IIi/. UltraSPARC doc.

[18] http://www.win.tue.nl/cow/Q2 . Queue 2.0 online tool.

[19] http://www.pcisig.com/specification PCI-Express Base Specification.

[20] http://www.broadcom.com NetXtreme Gigabit Ethernet Controller.

[21] http://www.spirentcom.com SmartBits 2000 Performance Analysis System.