Transient Analysis of Real-Time Systems With Soft Deadlines

Varsha Mainkar*
AT&T Bell Laboratories,
Holmdel, NJ 07733, USA.

Kishor S. Trivedi†
Center for Advanced Computing and Communications
Dept. of Electrical and Computer Engineering,
Duke University,
Durham, NC 27708, USA.

February 8, 1996

Abstract

Performance analysis of real-time systems offers unique challenges in analytical modeling because of system characteristics such as periodic arrivals of tasks and stringent response time requirements of these tasks. Consequently, evaluation of real-time systems has been largely limited to discrete event simulations or worst-case bounds. In this paper, we use an analytical model to evaluate a real-time system in which tasks have soft deadlines. The design of such systems does not require *guaranteed* meeting of deadlines, and therefore shows some "stochastic" behavior, which is suitably modeled using stochastic models. By making some reasonable assumptions, we model such a real-time system using *Deterministic and Stochastic Petri Nets* and solve the model by applying the theory of Markov regenerative processes. We study the time-dependent behavior of soft real-time systems, and also numerically compute response time distributions for the tasks.

Keywords : soft real-time systems, timed Petri nets, transient analysis, response time distribution

1 Introduction

Performance evaluation of real-time systems offers unique challenges to analytical modeling techniques. Real-time software systems typically consist of certain number of tasks arriving

*An earlier version of this paper appeared in the Proceedings of the Int'l Workshop on the Quality of Communication-Based Systems, Kluwer Academic Publishers, 1994. Most of this work was done at Duke University and was sponsored by IBM under the IBM/Duke University Research Agreement # RAL-R93010-00

†This research was sponsored by IBM under the IBM/Duke University Research Agreement # RAL-R93010-00

at regular intervals, while others that arrive sporadically, as a result of some external event. Some of these tasks can have very stringent response-time requirements. In case of critical real-time systems with *hard* deadlines, missing task completion deadlines can lead to serious system failure with catastrophic results. On the other hand, in *soft* real-time systems missing a deadline provides degraded service but does not lead to system failure [22]. Examples of such systems are bank transaction processing systems, airline reservation systems, and telephone switching systems. In a telephone switching system, a telephone call should be switched and connected within 2-3 seconds. However, if this response time requirement is not met, it only implies degradation of service to the customer.

Real-time systems with such soft response time requirements typically have a task system that is simple, and is designed to meet the requirements "almost all the time". However, there is no need to provide guarantees, and hence the design leaves some uncertainties, and some "randomness" in the behavior. Stochastic models, are therefore appropriate for modeling such behavior. When we make certain simplifying, but reasonable assumptions, we can derive a model which can be solved using some existing analytic/numerical techniques.

The real-time task system under consideration in this paper is one in which there are a set of periodic tasks, and some aperiodic tasks. The periodic tasks arrive at regular (deterministic) intervals, while the aperiodic tasks arrive according to some random process. The tasks are assigned static priorities, and they are non-preemptive. The characteristics of this system are very typical of existing real-time systems with soft constraints, and hence are quite realistic. The difficulties in this model lie in the incorporation of the periodic and aperiodic arrivals from many different sources with different service requirements. Using *deterministic and stochastic Petri nets*, we are able to make the description of this model simple; and using the theory of Markov regenerative processes, we are able to solve this model analytic/numerically. We can study the *time-dependent* behavior of the system and also numerically compute the *response time distributions* of tasks.

The existing literature largely shows use of models to compute average measures, worst-case bounds or approximate measures. Worst-case analysis techniques are appropriate for hard real-time systems, where the aim is to guarantee a certain degree of performance. By bounding the worst case performance, this goal can be achieved. In case of tasks with soft deadlines, approximate methods such as slowing down the server to account for some tasks in the system, so that response time for one important task can be found, have been used to provide simple performance measures [12]. The key contribution in this paper is to enhance these models for soft real-time systems, by incorporating much more detail, while still maintaining analytical tractability. Deterministic and stochastic Petri nets have been used earlier to model medium access protocols [9], for evaluating their applicability to real-time systems. A model for analyzing the performability of soft real-time systems was also proposed in [8] which uses the probability of a task meeting its deadline as an input. Our model in this paper provides this and other useful and detailed performance measures, all of which have practical significance, and provide a mathematical insight into the behavior of soft real-time systems.

The rest of the paper is as follows: in Section 2 we provide a brief introduction to DSPNs. In Section 3 we introduce the basic characteristics of real-time systems and describe in detail the features of the real-time system that we consider. In Section 4 we present its DSPN model. In Section 5 we describe how the transient solution of this DSPN can be obtained. In Section 6 we present a realistic example of a soft real-time system analyzed using our model. In Section 7 we present the DSPN model used to compute the response time distribution of an aperiodic task. We conclude the paper in Section 8.

2 Overview of DSPNs

A Petri net [19] is a directed bipartite graph with two types of nodes called **places** (represented by circles) and **transitions** (represented by rectangles or bars). Directed arcs (represented by arrows) connect places to transitions, and vice versa. If an arc exists from a place (transition) to a transition (place), then the place is called an input (output) place of that transition. Places may contain **tokens** (represented by dots or by a non-negative integer inside a place). The state of a Petri net is defined by a vector of the number of tokens in each place, called a **marking** of the Petri net. The notation $\#(p, m)$ is used to denote the number of tokens in a place p in a marking m. When the context is clear, the "m" is dropped from the notation. Typically, places represent conditions or resources, and transitions represent events or choices. Tokens may flow along the directed arcs of the Petri net according to some rules, thus reflecting the changes in the system due to various events.

A **multiplicity** is a non-negative integer that may be associated with an input or output arc. A transition is said to be **enabled** if each of its input places contains at least as many tokens as that input arc's multiplicity. An enabled transition can **fire**. When it fires, as many tokens as an input arc's multiplicity are removed from the corresponding input place, and as many tokens as an output arc's multiplicity are deposited in the corresponding output place. A set of transitions is said to be **conflicting** when the firing of one disables the rest. Transitions may be assigned priorities that can be used to resolve conflicts between transitions.

Structural extensions to Petri nets include **inhibitor** arcs (denoted by an arc with a circle instead of an arrow head), which connect places to transitions. A transition can be enabled only if the number of tokens in its inhibitor place is less than the multiplicity of the inhibitor arc.

Timed Petri nets associate a time delay with transitions. The **generalized stochastic Petri net** (GSPN) is a type of Petri net that allows transitions to have an exponentially distributed time delay (**timed** transitions, represented by rectangles) or a zero time delay (**immediate** transitions, represented by bars). A marking of a GSPN is said to be **vanishing** if at least one immediate transition is enabled in it and is said to be **tangible** otherwise. Conflicts among immediate transitions in a vanishing marking may be resolved by assigning probabilities to conflicting sets of immediate transitions. A GSPN can be solved analytically by generating its underlying stochastic process (termed the **marking process**) which is a CTMC. This CTMC is the one generated by the tangible markings of the GSPN. Several extensions have been proposed under the formalism of **stochastic reward nets** which keep the underlying stochastic process of the Petri net a CTMC, but make specification easier [4]: The firing rate of the timed transitions may be marking-dependent. The probability of firing of an immediate transition may also be marking-dependent. Multiplicities of arcs may be marking-dependent. Such arcs are termed **variable cardinality** arcs. Further, enabling functions or **guards** may be associated with transitions. Guards are marking-dependent predicates which must be satisfied for transitions to be considered enabled.

Deterministic and stochastic Petri nets [1] are timed Petri nets that allow immediate transitions and timed transitions, where the timed transitions may have either exponentially distributed firing time (termed EXP transitions) or deterministic firing time (termed DET transitions). The condition is that at any time *at most one deterministic transition may be enabled.* When this condition is met, it can be shown that the stochastic process corresponding to the tangible markings is the **Markov regenerative process** [3, 10] (also called semi-regenerative process in [2]). A Markov regenerative process can be described informally as a process in which we can find an embedded sequence of time points at which the past history of the evolution of the process may be "forgotten". The future evolution of the process depends only on the current state at these embedded time points. Note that between these **Markov regeneration epochs** the state of the process may change but this change does not imply a Markov regeneration. This is more general than the CTMC where at *any* time the past history is summarized in its current state, and the semi-Markov chain, where each state-change implies regeneration (in the Markovian sense).

For a formal definition of an MRGP we first need to define a Markov renewal sequence:

Definition 1 [2, 10]
A sequence of bivariate random variables $\{(Y_n, T_n), n \geq 0\}$ *is called a* **Markov renewal sequence** *if*

1. $T_0 = 0,\ \forall\ n > 0,\quad T_{n+1} > T_n$ *and* $Y_n \in \Omega \subset \mathbb{Z}$, *the set of integers*

2. $\forall\ i, j \in \Omega$,

$$\begin{aligned}
&P\{Y_{n+1} = j,\ T_{n+1} - T_n \leq t \mid Y_n = i, T_n, Y_{n-1}, T_{n-1}, ..., Y_0, T_0\} \\
&= P\{Y_{n+1} = j,\ T_{n+1} - T_n \leq t \mid Y_n = i, T_n\} \qquad \textit{(Markov Dependence)} \\
&= P\{Y_1 = j,\ T_1 \leq t \mid Y_0 = i\} \qquad\qquad \textit{(Time Homogeneity).}
\end{aligned} \tag{1}$$

The MRGP is defined as follows:

Definition 2 [2, 10]
A stochastic process $\{Z(t),\ t \geq 0\}$ *is called a* **Markov regenerative process** *(or a semi-regenerative process), if there exists a Markov renewal sequence* $\{(Y_n,\ T_n),\ n \geq 0\}$ *of random variables such that all the conditional finite dimensional distributions of* $\{Z(T_n + t),\ t \geq 0\}$ *given* $\{Z(u), 0 \leq u \leq T_n, Y_n = i\}$ *are the same as those of* $\{Z(t),\ t \geq 0\}$ *given* $Y_0 = i$.

This definition implies that

$$P\{Z(T_n + t) = j \mid Z(u), 0 \leq u \leq T_n, Y_n = i\} = P\{Z(t) = j \mid Y_0 = i\}. \tag{2}$$

This means that the Markov regenerative process $\{Z(t),\ t \geq 0\}$ does not have the Markov property in general, but there is a sequence of embedded time points

$(T_0, T_1, ..., T_n, ...)$ such that the states $(Y_0, Y_1, ..., Y_n, ...,)$ respectively of the process at these time points satisfy the Markov property. These time points are the Markov regeneration epochs. At time T_n, it does not matter what states the process $\{Z(t),\ t \geq 0\}$ has visited until reaching Y_n at time T_n. The state $Z(T_n)$ is the only needed information for the future evolution of $Z(T_n + t),\ t \geq 0$. Note that the stochastic process $\{Y_n, n = 0, 1, \ldots\}$ is a discrete time Markov chain. Further, let $N(t) = \sup\{n \geq 0 : T_n \leq t\}$. Then the stochastic process $\{X(t), t \geq 0\}$ where $X(t) = Y_{N(t)}$ is a semi-Markov process.

The following conditional probabilities are necessary to be defined for the analysis of an MRGP:

$$K_{ij}(t) = P\{Y_1 = j,\ T_1 \leq t \mid Y_0 = i\} \qquad i, j \in \Omega. \tag{3}$$

and

$$E_{ij}(t) = P\{Z(t) = j, T_1 > t \mid Y_0 = i\}. \tag{4}$$

The matrix $K = [K_{ij}(t)]$ is termed the **global kernel** [3] and is the joint conditional probability of the time to the next Markov regeneration and the state right after the next Markov regeneration given the state at the current Markov regeneration. The matrix $E_{ij}(t)$ (called the **local kernel**) describes the evolution of the MRGP between two Markov regeneration epochs. The matrices K and E will be used in computing the transient probability :

$$P_{ij}(t) = P\{Z(t) = j \mid Z(0) = Y_0 = i\} \tag{5}$$

Let $K*P$ denote a matrix whose (i, j)th element is $\sum_u K_{iu} * P_{uj}(t)$ where

$$K_{iu} * P_{uj}(t) = \int_0^t dK_{iu}(x) P_{uj}(t - x).$$

Then the transient probability $[P_{ij}(t)]$ satisfies the Markov renewal equation [2]:

$$P(t) = E(t) + K*P(t). \tag{6}$$

where $P = P_{ij}(t)$.

It was shown in [3] that if $\{Z(t), t \geq 0\}$ is the stochastic process corresponding to the tangible markings of the DSPN, then $Z(t)$ is an MRGP. The Markov regeneration epochs T_i were defined as :

- the time of the next state change, if no deterministic transition is enabled in the current state.
- the time of the firing of the deterministic transition, if one deterministic transition is enabled in the current state.

The matrices $K(t)$ and $E(t)$ corresponding to these Markov regeneration epochs were defined for any general DSPN in [3]. In this paper we shall use these equations and simplify them for the DSPN model of the real-time system.

3 System Description

3.1 Real Time System Characteristics

The design of a real-time system is significantly different from a normal computer system. A real-time system must be designed so that its behavior is *predictable* because of the strict timing constraints that it must satisfy. In case of *soft* real-time systems, this is usually done mainly by having a simple static priority structure in which the more important tasks get higher priority. This ensures (to a limited extent) that the critical tasks in a system will be done without much queueing delay. Also, if missing deadlines does not result in a catastrophe, a non-preemptive priority discipline is usually chosen, for its simplicity.

The components of a real-time system that we have abstracted out as being relevant to our performance analysis are the processor, and a set of periodic and aperiodic tasks. Each periodic task i, arrives to the system every τ_i time units. Periodic tasks typically perform some routine monitoring of the system. Arrival of aperiodic tasks is usually caused by external sources. Task execution times may vary a little, and are hence stochastic.

For us to evaluate this system analytically, we must make some assumptions. These are listed next.

3.2 Assumptions

The real-time system considered in this paper has the following features:

- There are m periodic tasks, $1, 2, \ldots, m$. Each periodic task k has a cycle time of $c_k \tau$, and a priority p_k. Thus *the interarrival times of each of these tasks are integer multiples of a positive number.*

- The timer for each periodic task begins at the same time, i.e. the arrival times of the periodic tasks are synchronized.

- The execution time of the task can be *phase-type* [18], however, assume for simplicity that it is exponentially distributed with mean $1/\mu_k$.

- There are l aperiodic task sources, $m+1, m+2, \ldots, m+n$. An aperiodic task of type $m+k$ arrives to the real-time system from a Poisson source at the rate λ_k. The execution time of the task can be phase-type, however, assume for simplicity that it is exponentially distributed with mean $1/\mu_{m+k}$. The aperiodic task also has an assigned priority of p_{m+k}.

- Each task has its own queue. Tasks of the same type are served with the first-come-first-serve discipline.

- The buffer for each task is finite. Let that buffer size be denoted by b_k for task k. Periodic tasks are not scheduled by the system when this buffer is full, and if an arriving aperiodic task finds that its buffer is full, it is lost.

- There is a single processor.

- Scheduling is static and fixed priority. Whenever a task completes, the next instance of a task with the highest priority (including periodic and aperiodic tasks) is executed. The scheduling is *non-preemptive.*

- Processing time required by the scheduler to choose the next task for execution is negligible.

4 The DSPN Model of the system

In this and the next section, we present the DSPN model of the real-time system, and present equations for transient analysis that are based on the theory developed in [3] but that have been simplified to a great extent because of the special nature of our problem.

With the assumptions outlined in the earlier section, a DSPN that models the state of the system can be built. Figure 1 shows a DSPN for a system with two periodic tasks and one aperiodic task. In this DSPN, we have used many of the *structural* extensions defined for stochastic reward nets (such as marking dependent multiplicities, marking dependent firing rate for transitions with exponentially distributed firing time) which make our task of specification easier, while still maintaining the analytical tractability of the DSPN.

In the figure, a token in place P_{avail} models an idle processor. Transition *timer* represents a clock tick which is used to schedule the periodic tasks. The time, τ, of this transition is equal to the greatest common divisor of the cycle times of the periodic tasks. Places P_1 and P_2 represent periodic tasks 1 and 2. When c_1 tokens accumulate in place P_1, it implies that the timer has fired c_1 times, and thus an instance of task 1 is created and is waiting to be scheduled. Similarly c_2 tokens accumulated in place P_2 denotes the arrival of periodic task 2. The cycle time of a periodic task i is enforced by setting the multiplicity of the arc from P_i to immediate transition t_i equal to c_i. Thus transition t_i is enabled only if there are at least c_i tokens in P_i. Transition t_{arrive} and place P_3 represent the arrival of an aperiodic task. Transition t_{arrive} has a firing rate λ_3. Transitions *timer* and t_{arrive} are always enabled. Further, transition t_i has priority p_i which models the static fixed priority scheduling policy.

When transition t_i fires, it puts i tokens in the place P_{exec} and removes that token from P_{avail}. This represents the non-preemptive nature of the discipline. Thus while the server is occupied by one task, it is unavailable to other tasks. This keeps track of the task that is executing in the processor. The rate of transition T_{exec} is marking-dependent and is equal to μ_k when there are k tokens in place P_{exec}. The arc from P_{exec} to T_{exec} is a variable cardinality arc, whose multiplicity is given by $\#(P_{exec})$ when $\#(P_{exec}) > 0$ and 1 otherwise. Thus when T_{exec} fires it, removes all the tokens from P_{exec}.

The transitions f_i model the limit on the number of waiting tasks in the system. Thus any arrival which occurs when the limit has been reached are "rejected". This is modeled by setting the multiplicity of the arc from P_i to f_i equal to c_i ($c_3 = 1$), and a guard for transition f_i which is true when $\#(P_i) = (m_i + 1)c_i$ and false otherwise. Thus if the buffer limit of task 2 is 2 and the period is 2τ, 2 tokens will be immediately "flushed out" when the number of tokens in P_2 reaches 6. This models the rejection of the task. The priority of transitions f_i is higher than all the other transitions in the net.

If tasks have phase-type execution time, it is very easy to extend the DSPN in Figure 1 to represent phase-type distributions. The transition T_{exec} will have to be replaced by a subnet for each task representing the phase-type execution time.

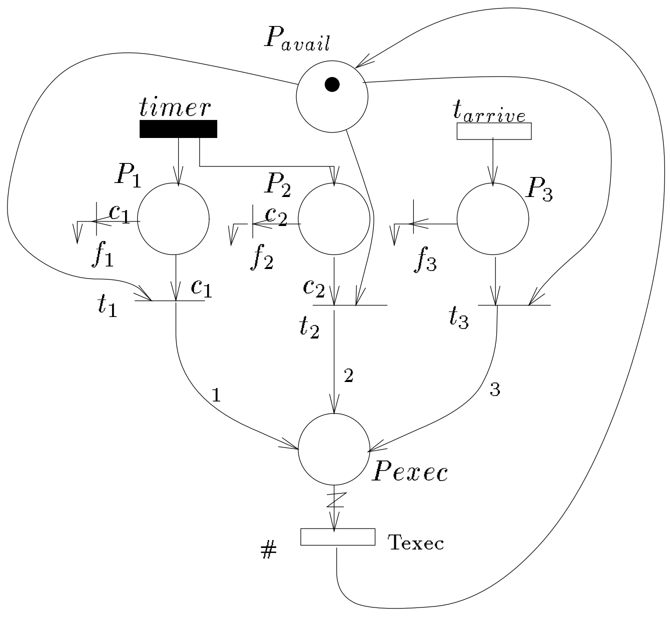

Figure 1: DSPN model of the real-time system

5 Transient Solution of the DSPN Model

The tangible markings of the DSPN of Figure 1 define a stochastic process. Since the DSPN satisfies the condition of at most one DET transition being enabled at any time, this marking process is the Markov regenerative process $Z(t)$. The transient solution of this MRGP is simplified by the fact that there is one deterministic transition (the *timer*) enabled all the time which is not competitive with any other transition. Thus the Markov regeneration epochs of $Z(t)$ are the times of the firing of the transition *timer*, i.e., $0, \tau, 2\tau, \ldots$. Between two firings of the *timer*, the two other events that may bring about a state change are arrival of an aperiodic task and completion of a task on the processor. The timings of these events are exponentially distributed and may be described by a CTMC (continuous time Markov chain) generator matrix. In particular, let Q denote the infinitesimal generator matrix corresponding to all the transitions between tangible markings caused by the firing of EXP transitions. For instance, consider the tangible marking $M_1 = (\#(P_{avail}), \#(P_1), \#(P_2), \#(P_3), \#(P_{exec})) = (1,0,0,0,0)$. This tangible marking directly reaches tangible marking $M_2 = (0,0,0,0,3)$ by the firing of transition t_{arrive} which has exponentially distributed firing time. Therefore, $Q_{12} = \lambda_3$.

The state changes due to the firing of the deterministic transition *timer* are recorded in a matrix Δ. Then, Δ is defined as:

$$\Delta_{ij} = \Pr\{\text{next tangible marking is } M_j \mid \text{current tangible marking is } M_i \text{ and } timer \text{ fires}\}.$$

For instance, suppose $c_1 = 1$ and $c_2 = 2$; then the tangible marking reached from marking M_1 defined above, due to firing of transition *timer* is $M_3 = (0, 1, 0, 0, 1)$, with probability one. Therefore, $\Delta_{13} = 1$.

Now, suppose we are given the initial probability vector, $p(0)$ of the system. Suppose we need to find the probability that the system is in state j at time t, where $t < \tau$. Obviously, this probability is simply $p_j(t)$, where $p(t) = [p_j(t)]$ is given by,

$$p(t) = p(0)e^{Qt}. \tag{7}$$

This is because the only state changes that can occur within the interval $[0, \tau)$ are the ones caused by EXP transitions.

The state probability vector at time $t = \tau + s$, where $0 \leq s < \tau$, can be derived as follows: the transient probability vector of the system just before the timer first fires is given, according to Equation (7), by $p(\tau^-) = p(0)e^{Q\tau}$. The probability vector at the just after the timer fires, is given by $p(\tau^+) = p(0)e^{Q\tau}\Delta$. Next, the evolution over the subsequent time period $(\tau, \tau + s)$ is again due to only exponential transitions and will be described by the Q matrix. Finally, the transient probability vector at time t is given by :

$$p(t) = p(\tau + s) = p(0)e^{Q\tau}\Delta e^{Qs}. \tag{8}$$

In the notation described in Section 2, the matrix $E(t)$ corresponds to e^{Qt} for $t < \tau$ and zero otherwise, while the matrix $K(t)$ corresponds to $e^{Q\tau}\Delta$ for $t \geq \tau$ and is zero otherwise.

In general, for any time t, let $t = n\tau + s$, where $n \geq 0$ and $0 \leq s < \tau$. Then,

$$\begin{aligned} p(t) &= p(n\tau + s) \\ &= p(n\tau)e^{Qs} \\ &= p((n-1)\tau)e^{Q\tau}\Delta e^{Qs} \\ &= p(0)(e^{Q\tau}\Delta)^n e^{Qs}. \end{aligned} \tag{9}$$

The term $e^{Q\tau}\Delta$ represents the one-step transition probability matrix of the DTMC, Y_n, embedded at the Markov regeneration epochs. If we assume that $c_1 = 1$ and $c_2 = 2$, then this DTMC is periodic with a period 2. For this DTMC, let $\Pi_0 = \lim_{n\to\infty}(e^{Q\tau}\Delta)^{2n}$ and let $\Pi_1 = \lim_{n\to\infty}(e^{Q\tau}\Delta)^{(2n+1)}$. Then, if $t = 2n\tau + s$,

$$\lim_{n\to\infty} p(2n\tau + s) = p(0)\Pi_0 e^{Qs}, \tag{10}$$

and if $t = (2n + 1)\tau + s$,

$$\lim_{n\to\infty} p((2n + 1)\tau + s) = p(0)\Pi_1 e^{Qs}. \tag{11}$$

Once the state probability vector is computed we can find various measures by expressing them as weighted sums of state probabilities. For instance, Figure 2 shows the processor utilization by periodic tasks, when priorities are assumed to be $p_1 > p_2 > p_3$. The values used are $\lambda_3 = 0.5, \mu_1 = 6.0, \mu_2 = 3.0, \mu_3 = 2, m_1 = m_2 = m_3 = 2$, and $\tau = 1$ and we assume that initially the processor is idle (time units are in seconds). As can be seen the probability exhibits periodic behavior as $t \to \infty$, with period $2\tau = 2$.

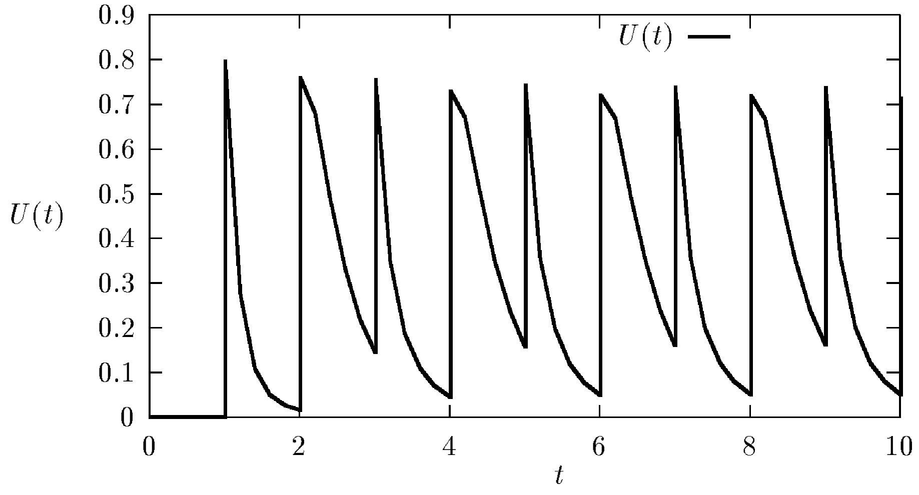

Figure 2: Transient probability that the processor is running periodic tasks.

Since in the long run, utilization is periodic, we must compute time-averages if we need the long-run fraction of processing time utilized by periodic tasks. Such a quantity has been used in estimating performance of aperiodic tasks by using the method of slowing the server down [12]. The time averaged probability vector at time t is defined as:

$$l(t) = \frac{1}{t}\int_0^t p(x)dx. \tag{12}$$

Let $t = n\tau + s, 0 \le s < \tau$. Then we can derive $l(t)$ as follows [14]. First,

$$\begin{aligned}\int_0^t p(x)dx &= \int_0^{\tau^-} p(x)dx + \int_{\tau}^{(2\tau)^-} p(x)dx + \int_{(n-1)\tau}^{(n\tau)^-} p(x)dx + \int_{n\tau}^{n\tau+s} p(x)dx \\ &= \sum_{i=0}^{n-1} p(0)(e^{Q\tau}\Delta)^i \int_0^{\tau^-} e^{Qx}dx + p(0)(e^{Q\tau}\Delta)^n \int_0^s e^{Qx}dx.\end{aligned}$$

We would like to find $\lim_{t\to\infty} l(t)$. For this, we first note that the limit of the discrete time averaged probability of the embedded DTMC is given by [10]:

$$\lim_{n\to\infty} \frac{\sum_{i=0}^{n-1} p(0)(e^{Q\tau}\Delta)^i}{n} = \Pi,$$

where Π is given by the solution to the system of equations:

$$\Pi = \Pi e^{Q\tau}\Delta, \qquad \sum_j \Pi_j = 1.$$

For a periodic DTMC such as this one which has a period of 2, Π_j can also be computed as [10]:

$$\Pi_j = \frac{[\Pi_0]_{ij} + [\Pi_1]_{ij}}{2}.$$

Now,

$$\begin{aligned}
\lim_{t\to\infty} l(t) &= \lim_{n\to\infty} \left[\frac{\sum_{i=0}^{n-1} p(0)(e^{Q\tau}\Delta)^i \int_0^{\tau^-} e^{Qx}dx + p(0)(e^{Q\tau}\Delta)^n \int_0^s e^{Qx}dx}{n\tau + s} \right] \\
&= \lim_{n\to\infty} \left[\frac{\frac{\sum_{i=0}^{n-1} p(0)(e^{Q\tau}\Delta)^i}{n} \int_0^{\tau^-} e^{Qx}dx}{\tau + s/n} \right] + \lim_{n\to\infty} \left[\frac{p(0)(e^{Q\tau}\Delta)^n \int_0^s e^{Qx}dx}{n\tau + s} \right] \\
&= \frac{\Pi}{\tau} \int_0^{\tau^-} e^{Qx}dx. \qquad (13)
\end{aligned}$$

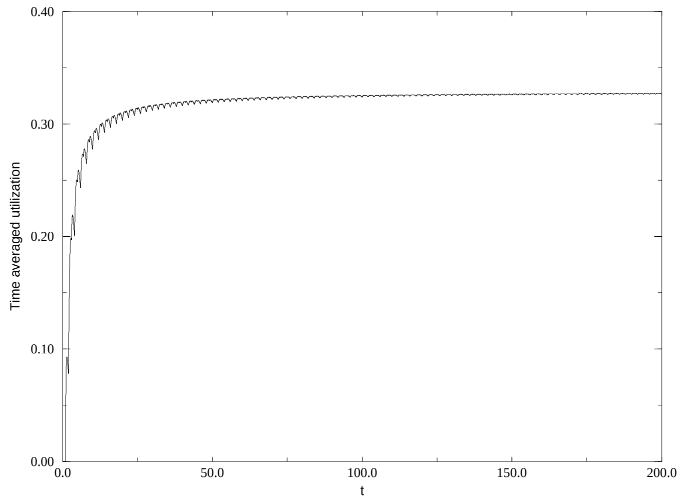

Figure 3: Time averaged utilization by periodic tasks.

The time-averaged utilization by periodic tasks for the earlier example is shown in Figure 3. The long-run utilization converges to 0.33.

5.1 Computational and Complexity Issues

The number states of the MRGP generated by the above DSPN is $O((m+l)N\prod_{i=1}^{m+n} b_i)$ [13], where N is the least common multiple of the c_i's.

In the Q matrix, there are at most $l+2$ non-zero entries in each row. Therefore, Q can be stored as a sparse matrix with $\eta = O(l(m+l)N\prod_{i=1}^{m+l} b_i)$.

There are several ways to numerically compute the matrix exponential [16, 20] in Equation (9). One method that is numerically accurate is the *uniformization* method [5, 6]. If $q = -\max\{Q_{ii}\}$, then uniformization uses the matrix $Q^* = Q/q+I$ to compute the product $p(0)e^{Q\tau}$ as follows :

$$p(0)e^{Q\tau} = \sum_{i=0}^{\infty} p(0)(Q^*)^i e^{-q\tau}\frac{(qt)^i}{i!}. \tag{14}$$

If the matrix Q has η non-zero entries, then this computation has a time complexity of $O(\eta q\tau)$ [21].

For computing $p(t)$ for $t = n\tau+s$, a uniformization routine and a matrix-vector multiplication must be carried out $n+1$ times. Then the final complexity is $O(nl(m+l)N\prod_{i=1}^{m+l} b_i)$. Note that if we want to compute $p(t)$ for a range of successive values of t, we can reuse some of the computation for $p(t)$ for computing $p(t+x), x \geq 0$.

5.2 Advantages of using the DSPN model

Using the DSPN/MRGP approach to solve this problem gives us some distinct advantages over other approaches.

As is apparent from the model description, the underlying state-space of this model can be large. Thus even if we had recognized the underlying stochastic process of the system by simply identifying the regeneration epochs, actually constructing the state diagram would have been tedious. Using a DSPN specification we could take advantage of the mappings from DSPN to MRGP derived in [3] and "automate" that process. This makes experimenting with different parameters of the model easier.

The DSPN/MRGP approach also gives us the ability to evaluate a system model in an analytic/numeric fashion even though the expressions for the performance measures cannot be written in closed-form. The advantage of this ability is significant. Practising modelers often tend to use one of two approaches; (1): Deriving closed-form equations by greatly simplifying the model or (2): when this is not possible, write a discrete-event simulation. However, in many practical situations, assumptions of exponential distributions for various events are made even in simulation models, simply because there are no measurements/data of the actual system that might indicate otherwise. In this case, an "intermediate" approach, in which we do not do either (1) or (2) but instead (3): generate the underlying state space using Petri net based specification tools and solve the underlying stochastic process, is justified. As compared with simulation, this approach can have significantly shorter computation times.

Further, the computation time in this method are independent of the parameters that do not affect the state space. However, in case of simulation, it is a well known fact that the simulation time, necessary to generate results with sufficient confidence, increases rapidly as system utilization increases [11].

From this argument, one may suggest using a simple continuous time Markov chain model (generated using an SRN) instead of an MRGP, because phase expansions can always be applied to approximate non-exponential distributions [18]. In our application, however, the results are very sensitive to such approximations. Figure 4 shows the loss probability of an aperiodic task computed by a stochastic reward net model, where the firing time distribution of transition *timer* is approximated by an Erlang distribution with K stages. Observe that with 100 stages, the plot still does not approximate the DSPN model well. With 100 stages, the CTMC underlying the SRN had 9800 states (whereas the DSPN model has 98 states). Thus this approximation resulted in state-space explosion, while still not giving a good approximation.[1]

6 Call Processing Example

The above model can be used to represent many soft real-time systems such as transaction processing systems, airline reservation systems or telephone call processing systems. Consider the third example, of a switching system, that must process arriving telephone calls and connect them within a certain amount of time. In a real switching system there will be several processes running at different priority levels. For simplicity, we consider three main tasks, at three priority levels. Such a simplification not only makes the model easier but also has practical significance, because it provides faster answers as opposed to a detailed discrete-event simulation model.

Consider a switching system with three tasks. The highest priority is a periodic task (task 1) that provides functions such as preventive checking for faults, overload control of the switch and other critical functions. The lowest priority task (task 2) is also a periodic task that does all routine functions such as audits. The higher priority periodic task is scheduled to run more frequently than the lower priority task. The second priority task (task 3) is created when a call arrives and does all call processing functions.

Suppose the high priority periodic task is scheduled to run every τ seconds. Task 3 arrives according to a Poisson process with mean interarrival time $1/\lambda$ and task 2 arrives every 2τ seconds. Let $\tau = 1$, and the mean processing times of task 1,2 and 3 be $0.01, 0.125$ and 0.1 seconds respectively. Further let the buffer sizes for each task be 1, 1 and 20 respectively. The buffer size for task 1 and 2 is one because these are routine monitoring tasks, and if one instance of such a task has not executed there is not much gain in scheduling another instance of the same task that does the identical job.

There can be many questions we can ask about such a system.

Capacity Determination : The call processing capacity can be determined using this

[1]This argument also implies that the assumption of phase-type execution time may also have a significant impact on the results. However, results from the literature [7] suggest that fitting phase-type distributions to service times do provide fairly good approximations. The same does not apply to approximating a periodic arrival by a phase-type renewal process, because periodic arrivals result in periodicity in the steady-state probabilities. It is this periodicity that is hard to duplicate using phase-type approximations.

model. Figure 5 shows the long range time averaged throughput of the call processing task as a function of offered call arrival rate. The long range time averaged throughput, Λ of the call processing task is given by

$$\Lambda = \lim_{t \to \infty} \frac{\int_0^t \mu_3 U_3(x) dx}{t}, \tag{15}$$

where μ_3 is the execution rate of the call processing task and $U_3(t)$ is the probability that the processor is executing task 3 at time t.

As can be seen the attained throughput saturates at approximately 9.9 calls per second, which is the therefore the maximum capacity of the switch.

Call Processing Overload Suppose the lowest priority task is "deferrable" only to a certain degree; i.e., if the call processing load suddenly increases, the system sanity can be preserved if the lowest priority task does not execute in *every* cycle that it is scheduled. However, it has been determined that this task must occupy at least 2 % of the time in an interval of 6 seconds. If not, then the system is considered to be in a critical overload. We can ask, for example, how large a transient overload can be sustained for how long without getting into a critical overload. [2]

For example, suppose the call processing load on the switch has been 8 calls per second. We assume that, the load changes to 10 calls per second at time $t = 20$. We can compute the system state probabilities at time $t = 20$, and use them as initial probabilities for the next "phase" of the stochastic process. Let $q(x)$ denote the state probability vector of the system x seconds after the system load increased. Then, we can set $q(0) = p(20)$ and $\lambda = 10$ and solve for $q(x)$ using the Equation (9). Using $q(x)$, we can compute the time averaged utilization in an interval as before.

The occupancy of the low priority task in any interval is given by the time averaged utilization of that task in that interval. Using this $q(x)$, we can find this time averaged probability at $x = 6$ seconds, to determine if the system is in critical overload. Figure 6 shows the time averaged occupancy of the low priority task, over the interval of 6 seconds after the load increases. It can be seen that an increase of load to 10 calls/second does not push the time averaged occupancy of the low priority task below 0.02, however, a load of 12 calls/per second or greater does do so. Therefore a transient overload of 12 calls/per second or more will push the system into overload, in 6 seconds. If this same requirement was for 3 seconds instead of 6 seconds, it would take a 15 calls/second load to push the system into an overload state.

This example brings out the key advantage of transient analysis. Such an analysis will not be possible if we observe the system at discrete epochs.

[2]When a call processing system goes into an overload, it will block/delay new arriving calls, until the threshold occupancy level of low priority tasks is regained. It is therefore, very important to be able to quantify the impact of transient overloads on such a system. Note, however, that we are not actually modeling overload control in this paper.

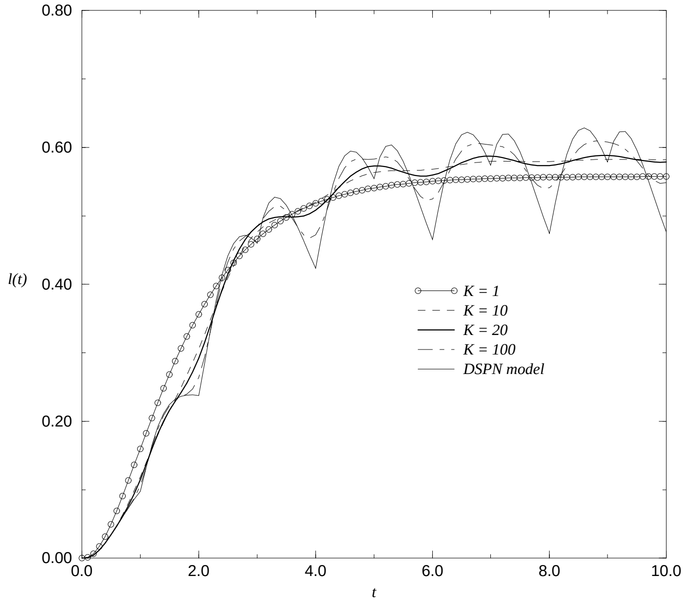

Figure 4: Comparison of SRN model *vs.* DSPN model.

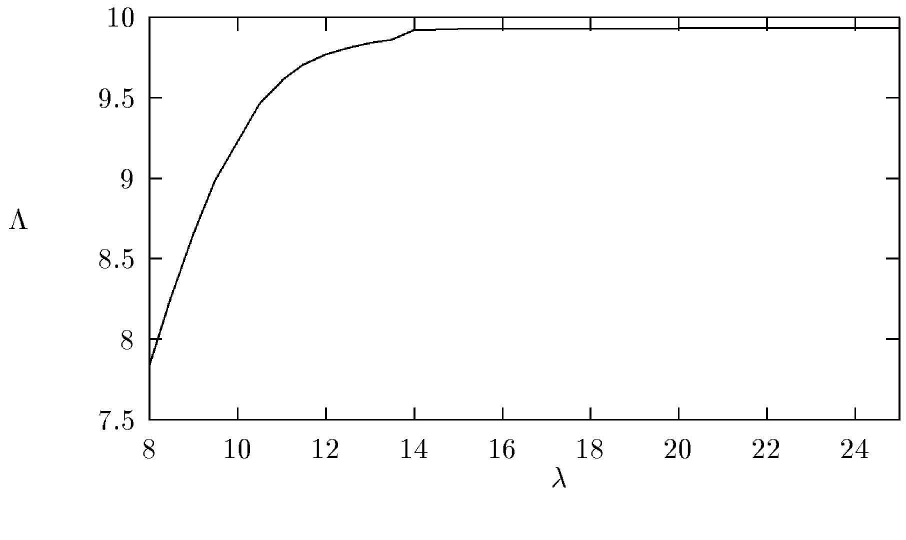

Figure 5: Long Range Throughput *vs.* Call Arrival Rate.

In this example the underlying MRGP has 358 states and the computation to generate the plot in Figure 6 with 40 points took 25 minutes, for each curve, on a SparcStation 1+, with 16MB memory.

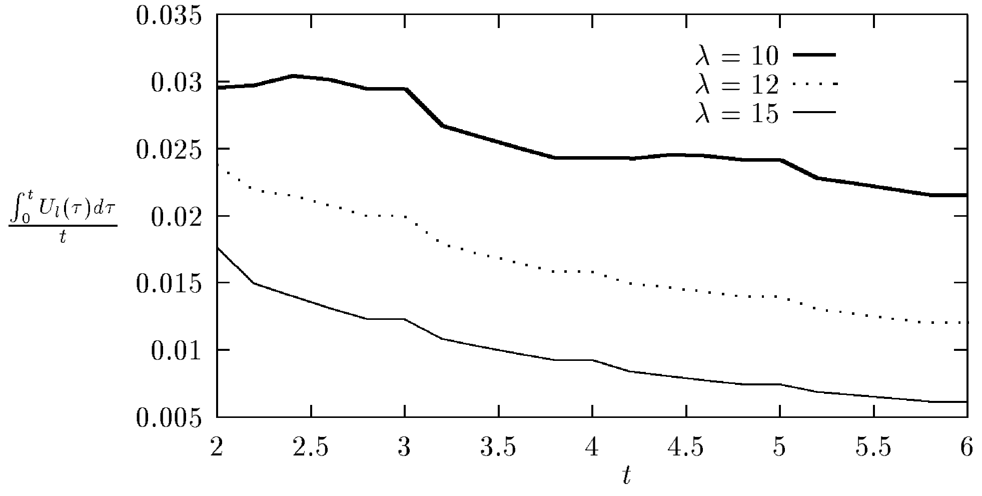

Figure 6: 6-second occupancy of low priority calls

Buffer Sizing There may also be a requirement on blocking of calls. Suppose the requirement is that at any time the probability that an incoming call is blocked is less than 2%. Figure 7 shows loss probabilities for various buffer sizes. Among the sizes shown, clearly a buffer of size equal to or greater than 20 is required.

7 The DSPN model for response time distribution

An important measure for tasks which have deadlines is the probability that a task meets it deadline. In this section we modify and augment our earlier model so as to compute the response time distribution of the *aperiodic task* in our 3-task system. Note that here too, $c_1 = 1$ and $c_2 = 2$, therefore the results from the earlier section suggest that the response time distribution of a task that arrives at some time $t = n(2\tau)+s$, $0 \leq s < 2\tau$ depends on s as $t \rightarrow \infty$. Thus there is no single "limiting response time distribution". In the long range, the response time distribution depends on the time of arrival to the system as measured from the beginning of the current period.

In this section we use the *tagged task approach* [15, 17] in computing the response time distribution given its time of arrival with respect to the time of the beginning of the current period, *and* given the fact that it is not rejected by the system.

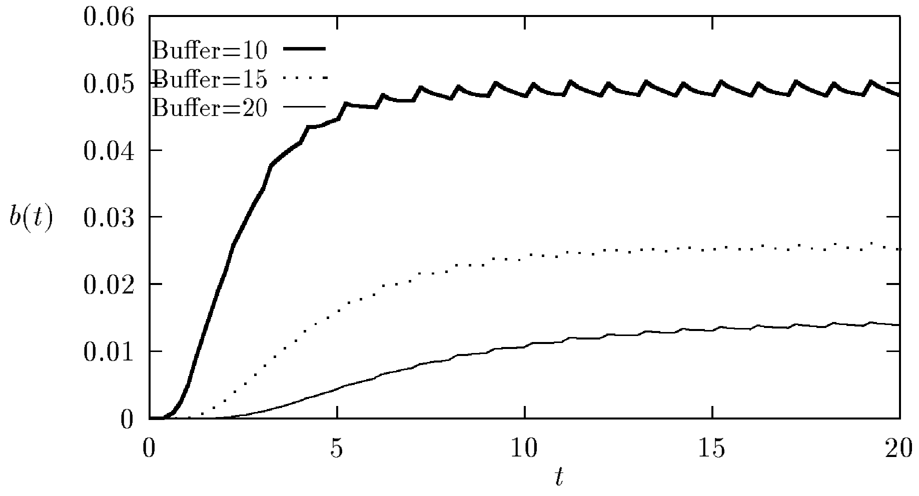

Figure 7: Buffer Sizing For Incoming Calls.

Suppose a task arrives at time $t = 2n\tau + s, 0 \leq s < \tau$, where t is large. Since aperiodic task arrivals are Poisson, the state probability vector of the system as seen by the arriving task is $p(0)\Pi_0 e^{Qs}$ [23]. Similarly, if a task arrives at time $t = (2n+1)\tau + s, 0 \leq s < \tau$, the state probability vector of the system as seen by the arriving task is $p(0)\Pi_1 e^{Qs}$. Since we restrict our tagged jobs to those that are not rejected, these probabilities must be normalized to reflect the condition that the task is accepted by the system. Let $S(t)$ denote the state of the system as seen by a task arriving at time t that gets accepted. Then for large t,

$$\begin{aligned}\Pr[S(t) = M_i \ \mid \ t = 2n\tau + s, \#(P_3, M_i) < b_3] \\ = \frac{[p(0)\Pi_0 e^{Qs}]_i}{\sum_{\{j \mid \ \#(P_3,M_j)<b_3\}}[p(0)\Pi_0 e^{Qs}]_j}, \qquad & \text{if } \#(P_3, M_i) < b_3 \\ = 0 \qquad & \text{otherwise.}\end{aligned} \tag{16}$$

$$\begin{aligned}\Pr[S(t) = M_i \ \mid \ t = (2n+1)\tau + s, \#(P_3, M_i) < b_3] \\ = \frac{[p(0)\Pi_1 e^{Qs}]_i}{\sum_{\{j \mid \ \#(P_3,M_j)<b_3\}}[p(0)\Pi_1 e^{Qs}]_j}, \qquad & \text{if } \#(P_3, M_i) < b_3 \\ = 0 \qquad & \text{otherwise.}\end{aligned} \tag{17}$$

Let this probability be denoted by a_i. Thus a_i is the conditional probability that the arriving task will find the system in state i given that the arriving task will be accepted.

Figure 8 shows the DSPN model for computing response time distribution of an aperiodic task. In the DSPN model, we now add a place that denotes a *tagged task*, P_{tagged}. The arc multiplicities represent the execution of this task separately, by putting 4 tokens in place P_{exec} when transition t_{tagged} fires. The arc from T_{exec} to P_{done} has variable multiplicity defined as follows: it is 1 if $\#(P_{exec}) = 4$, and is 0 if $\#(P_{exec}) \neq 4$. The inhibitor from P_3 to t_{tagged} makes sure that the aperiodic tasks that are already there when the tagged task arrives, are completed before the tagged task. When a token appears in place P_{done} the DSPN "halts", and reaches an absorbing state. This state denotes the completion of the tagged task. The initial distribution of markings of this DSPN is as follows: A marking $(M_i, \#(P_{tagged}), \#(P_{done}))$ where $M_i = (\#(P_{avail}), \#(P_1), \#(P_2), \#(P_3), \#(P_{exec}))$ has probability a_i as defined by Equations (16) and (17) above, if $\#(P_{tagged}) = 1$ and $\#(P_{done}) = 0$ and 0 otherwise.

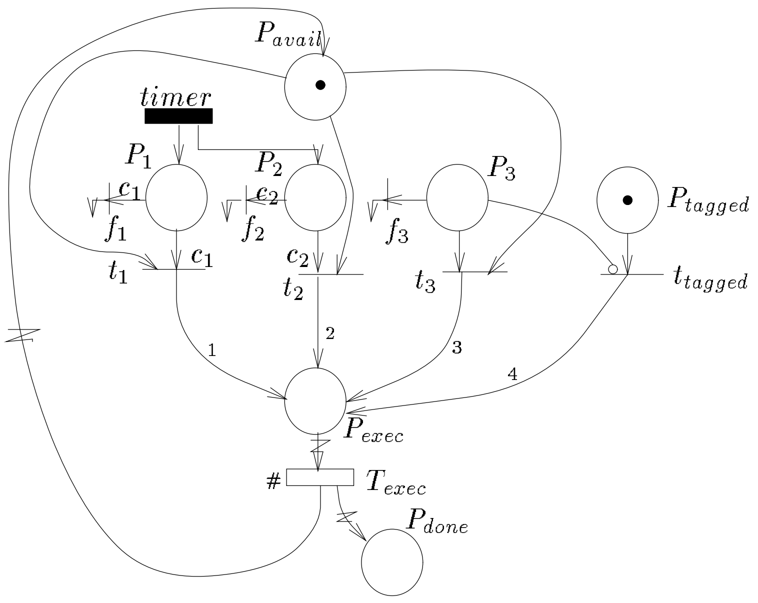

Figure 8: DSPN model for response time distribution

With this initial marking probability distribution, transient analysis can be carried out on this DSPN in a way similar to the one described in Section 5. The distribution of conditional response time, R, of the aperiodic task arriving at time $t = 2n\tau + s$ where $0 \leq s < 2\tau$ is given by the following

$$\Pr[R \leq x \;\; | S = s, \#(P_3, M(t)) < b_3] = \sum_{\{i | \#(P_{done}, M_i(x)) = 1\}} p_i^{(R)}(x), \tag{18}$$

where the superscript (R) is used to denote the fact that this is the transient probability corresponding to the response time DSPN model of Figure 8, and S is the random variable corresponding to the time s. Now, since the aperiodic task arrival process is Poisson, we know that the S is uniformly distributed in the interval $[0, 2\tau]$, and we can also compute the transient probability of the buffer being full. Since this probability is periodic with period 2τ, $b(t) = b(2n\tau + s) = b(s)$. Using these two probabilities we can find the unconditional response time distribution of an aperiodic task as :

$$\Pr[R \leq x] = \frac{1}{2\tau}[\int_0^{2\tau} \Pr[R \leq x \mid S = s, \#(P_3, M(2n\tau + s)) < b_3]b(s)]ds. \tag{19}$$

7.1 Call Processing Example

In the call processing example of Section 6, the most important requirement of the switching system, will be the response time of arriving calls, which we interpret as the response time of the aperiodic task that gets created when the call arrives. Let the requirement be that at least 90 % of the calls that are accepted must get processed within 0.5 seconds.

We can solve the system as described above to compute this probability. For parameters given in Section 6, the probability that a call is processed within 0.5 seconds is 0.540471. If the computation for the aperiodic task is sped up by a factor of 1.5, that probability goes up to 0.950232.

The time to generate the distribution in the interval $[0, 1]$ at 10 points, took 10 minutes on a SparcStation 4, with 32MB of memory.

8 Conclusions

In this paper we used Markov regenerative theory to analyze the transient behavior of a soft real-time system. DSPNs were used as a specification method. Though the abstracted model of the real-time system had several assumptions which made it mathematically tractable, we believe a majority of those were justified for soft real-time systems. The analytical technique used gave us insight into the time-dependent behavior of the real-time system. Using a realistic call-processing example we showed that our analytic model could provide quick answers to questions that would otherwise require a long time with a discrete-event simulation model. We numerically quantified the effects of the periodic behavior of a real-time system. The response-time distribution of an aperiodic task was also computed.

The assumption of exponential distribution for soft real-time systems can be relaxed (to phase-type distributions) at the expense of making the state-space larger. Also, pre-emptive priority scheduling can be incorporated easily only if we assume execution time distribution to be exponential. These are some of the important limitations of the model which will be explored in our future work.

Acknowledgements

We wish to acknowledge the help and encouragement provided by Pam Binns of Honeywell Technology Center. We would also like to thank Dimitris Logothetis and Sandy Wang for useful discussions.

References

[1] M. Ajmone-Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Lecture Notes in Computer Science*, volume 266, pages 132–145. Springer-Verlag, 1987.

[2] E. Çinlar. *Introduction to Stochastic Processes.* Prentice-Hall, Englewood Cliffs, NJ, U.S.A., 1975.

[3] H. Choi, V. Kulkarni, and K. Trivedi. Transient analysis of deterministic and stochastic Petri nets. In *Proc. of The 14th Intl. Conf. on Application and Theory of Petri Nets*, pages 166–185, Chicago, U.S.A., Jun. 21-25 1993.

[4] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proc. Int. Conf. on Petri Nets and Performance Models*, pages 142–150, Kyoto, Japan, Dec. 1989.

[5] D. Gross and D. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Oper. Res.*, 32(2):926–944, Mar.-Apr. 1984.

[6] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skand. Aktuarietidskr.*, 36:87–91, 1953.

[7] M. A. Johnson and M. R. Taffe. An investigation of phase-distribution moment-matching algorithms for use in queueing models. *Queueing Systems*, 8(2):129–147, 1991.

[8] K. M. Kavi, H. Y. Youn, and B. Shirazi. A performability model for soft real-time systems. In *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, 1994.

[9] C. Kelling and G. Hommel. Modeling priority schemes with timed Petri nets. In *Proceedings of the Second Workshop on Parallel and Distributed Real-Time Systems*, 1994.

[10] V. G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman Hall, 1995.

[11] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis.* McGraw Hill, Inc., 1991.

[12] J. P. Lehocky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the 13th Real-Time Systems Symposium*, pages 110–123, Phoenix, Arizona, U.S.A., 1992.

[13] D. Logothetis, V. Mainkar, and K. Trivedi. Transient analysis of non-markovian queues via markov regenerative processes. In Preparation.

[14] D. Logothetis and K. Trivedi. Transient analysis of the leaky bucket rate control scheme under Poisson and ON/OFF source. In *Proc. of the 13th IEEE INFOCOM*, Toronto, Canada, June 1994.

[15] B. Melamed and M. Yadin. Numerical computation of sojourn-time distributions in queueing networks. *J. ACM.*, 31(4):839–854, Oct. 1984.

[16] C. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–835, Oct. 1978.

[17] J. K. Muppala, K. S. Trivedi, V. Mainkar, and V. Kulkarni. Numerical computation of response time distributions using stochastic reward nets. *Annals of Operations Research: Special Issue on Queuing Networks*, 48:155–184, 1994.

[18] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastic Models.* Johns Hopkins, Baltimore, MD, U.S.A., 1981.

[19] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, U.S.A., 1981.

[20] A. Reibman, R. Smith, and K. S. Trivedi. Markov and Markov reward model transient analysis: An overview of numerical approaches. *European J. Oper. Res.*, 40:257–267, 1989.

[21] A. L. Reibman and K. S. Trivedi. Numerical transient analysis of Markov models. *Comput. Oper. Res.*, 15(1):19–36, 1988.

[22] J. A. Stankovic and K. Ramamritham. *Hard Real-Time Systems.* IEEE Computer Society Press, Los Angeles, CA, 1988.

[23] R. W. Wolff. Poisson arrivals see time averages. *Oper. Res.*, 30(2), Mar.-Apr. 1982.