Real-Time-Systems Performance in the Presence of Failures

Jogesh K. Muppala, Duke University
Steven P. Woolet, IBM
Kishor S. Trivedi, Duke University

The growing use of real-time systems in such diverse areas as transaction processing, air-traffic control, process control, and mission-critical control has created a need for effective analysis techniques to model these systems. Real-time systems are characterized by stringent deadlines and high-throughput and high-reliability requirements. Traditional analysis techniques, which rely mostly on mean values of the measures, cannot capture all the nuances of these systems.

When analyzing a transaction processing system, we often wish to compute the response time for a transaction. A typical performance requirement for transaction processing systems is a 1-second response for at least 95 percent of the transactions (usually referred to as the 95th percentile). Because failure to meet the deadline results in losses but may not be catastrophic, such systems are called *soft* real-time systems.

A mission-critical system is best characterized by how fast it can respond to a change in input stimuli or the environment. Typically we would require a guaranteed response within a short period of time commonly referred to as a *hard deadline*. Thus, such systems are called *hard* real-time systems, since the consequences of failing to meet the deadline could be catastrophic. Here, it is important to know the probability that the response time will be less than the deadline; this implies that the response-time distribution must be computed.

This unified methodology for modeling real-time systems uses techniques that combine the effects of performance, reliability/availability, and deadline violation into a single model.

Queueing networks[1] have been used traditionally to study the performance of computer systems. They are useful in representing contention for resources, as occurs in transaction processing systems. Computing the response-time distribution for queueing networks using closed-form expressions is extremely difficult even for networks with a simple structure.[2] Numerical evaluation of the response-time distribution, however, is not difficult, as we will show.

Failures and repairs of components in a real-time system can noticeably affect performance. Thus, any realistic system model must also incorporate the effect of failures and repairs. Meyer[3] introduced a useful framework called *performability* for combining performance and reliability. In this case, the performance levels can be incorporated into the failure-repair model to study overall system behavior. When the failure-repair behavior is Markovian in nature, the approach results in a Markov reward model.[4]

Shin and Krishna[5] defined some interesting performance measures for hard real-time multiprocessor systems. Failure to meet a hard deadline is assumed to be catastrophic, leading to system failure. They compute the dynamic failure probability, taking into account hard-deadline failures. A cost function associated with the tasks in the model is used as an optimization criterion for designing real-time systems. Al-

0018-9162/91/0500-00037$01.00 © 1991 IEEE

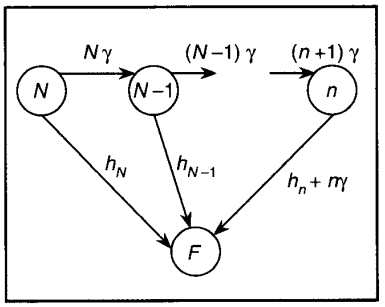


Figure 1. Markov chain model used by Shin and Krishna.

though Shin and Krishna's performance measures are general, their example is restricted to a single-resource model. Their approach complements Meyer's.

In this article we present a unified methodology for modeling both soft and hard real-time systems, combining Meyer's approach with that of Shin and Krishna. We use an on-line transaction processing system as an example to illustrate our modeling techniques. We consider dynamic failures due to a transaction's violating a hard deadline by incorporating additional transitions in the Markov chain model of the failure-repair behavior, as described by Shin and Krishna. We also take into account system performance in the various configurations by using throughput and response-time distribution as reward rates. Since the Markov chains used in computing the distribution of response time are often very large and complex, we use a higher level interface based on a variation of stochastic Petri nets called stochastic reward nets.[6]

A motivating example

Shin and Krishna present an interesting model for a real-time multiprocessor system incorporating hard-deadline failures. The performance of a multiprocessor system with m processors is modeled using an *M/M/m* queue with Poisson arrival of tasks at the rate λ. The system's initial configuration has N processors. The processors are allowed to fail, and the time to failure of

Markov chains and Markov reward models

A Markov chain is a state-space-based method for modeling systems. It is composed of states, and transitions between states. The states of a Markov chain can be used to represent various entities associated with a system — for example, the number of functioning resources of each type, the number of tasks of each type waiting at a resource, the number of concurrently executing tasks of a job, the allocation of resources to tasks, and states of recovery for each failed resource. A transition represents the change of the system's state due to the occurrence of a simple or a compound event, such as the failure of one or more resources, the completion of executing tasks, or the arrival of jobs. A transition thus connects one state to another.

A Markov chain is a special case of a discrete-state stochastic process in which the current state completely captures the past history pertaining to the system's evolution. Markov chains can be classified into discrete-time Markov chains and continuous-time Markov chains, depending on whether the events can occur at fixed intervals or at any time — that is, whether the time variable associated with the system's evolution is discrete or continuous. This article is restricted to continuous-time Markov chains. Further information on Markov chains is available in the literature.[1]

In a graphical representation of a Markov chain, states are denoted by circles with meaningful labels attached. Transitions between states are represented by directed arcs drawn from the originating state to the destination state. Depending on whether the Markov chain is a discrete-time or a continuous-time Markov chain, either a probability or a rate is associated with a transition.

To illustrate these concepts further, let's consider an example based on a multiprocessor system comprising two dissimilar processors, P1 and P2. When both processors are functioning, the time to occurrence of a failure in processors P1 and P2 is assumed to be a random variable with the corresponding distribution being exponential with rates γ_1 and γ_2, respectively. We also consider a common-mode failure whereby both processors can fail simultaneously. The time to occurrence of this event is also assumed to be exponentially distributed with rate γ_C. When only one processor is functioning, the rate of failure could be correspondingly altered to reflect the fact that a single processor carries the load of both processors. When only P1 is functioning, we assume its failure rate is γ_1', and when only P2 is functioning, its failure rate is γ_2'. We could consider repair of the processors where the time to repair the processors is also exponentially distributed with rates δ_1 and δ_2, respectively. We also assume that when both processors are waiting for repair, P1 has priority over P2.

The behavior of this system can be represented by a continuous-time Markov chain, shown in the figure in this sidebar. In this figure the label (i, j) of each state is determined as follows: $i = 1$ if P1 is up, and $i = 0$ if P1 is failed; similarly, $j = 0$ or 1, depending on whether P2 is failed or up. Solving this Markov chain involves computing either $P_{i,j}(t)$, the probability of being in state (i, j) at time t, or $\pi_{i,j}$, the steady-state probability of being in state (i, j).

Now let's consider the formal notation for a Markov chain. Let $\{Z(t), t \geq 0\}$ represent a homogeneous finite-state continuous-time Markov chain with state space Ω. Typically, we are interested in computing $P_i(t)$, the unconditional probability that the continuous-time Markov chain will be in state i at time t. The corresponding row vector $\mathbf{P}(t) = [P_i(t)]$ represents the *transient state probability vector* of the continuous-time Markov chain. Let π_i be the steady-state probability of state i of the continuous-time Markov chain. Then $\pi = [\pi_i]$ is the corresponding steady-state probability vector.

A Markov reward model is obtained by associating a *reward rate* r_i with each state i of the continuous-time Markov chain. Let $X(t) = r_{Z(t)}$ be the random variable corresponding to the reward rate at time t. Then the expected reward rate $E[X(t)]$ can be computed as follows:

$$E[X(t)] = \sum_i r_i P_i(t)$$

The expected reward rate in steady state can be computed as

$$E[X] = E[X(\infty)] = \sum_i r_i \pi_i$$

each processor is exponentially distributed with rate γ. The system is assumed to be nonrepairable, and at least n processors are required for the system to be stable. Figure 1 shows the corresponding Markov model. The hard deadline is assumed to be a random variable. To model hard-deadline failures, Shin and Krishna use additional transitions from each up state to the failure state. In this figure, h_i $(n \le i \le N)$ is the product of the task arrival rate and the probability that a task violates the hard deadline; that is, $h_i = \lambda \int_0^\infty [1 - F_{MMi}(t)] dF_d(t)$, where $F_{MMi}(t)$ is the response-time distribution of an *M/M/i* queue, and $F_d(t)$ is the cumulative distribution function of the random variable corresponding to the hard deadline. It is assumed that the queue is stable in state i, that is, $\lambda < i\mu$, where μ is the service rate of a single processor.

For this example Shin and Krishna compute the dynamic failure probability, given by the instantaneous probability $P_F(t)$, of state F. The dynamic failure probability is computed as a function of the number of processors as well as the number-power product (number of processors times the service rate of each processor). By defining a cost function for this model, the mean cost over the mission's lifetime was also computed.

For the same example, we might want to compute the amount of work processed until hard-deadline failure. This can be done using Markov reward models. The probability that a job will violate the deadline in state i has already been specified as $\int_0^\infty [1 - F_{MMi}(t)] dF_d(t)$. Thus, the rate of the arriving jobs that will meet the deadline in state i is given by $\lambda(1 - \int_0^\infty [1 - F_{MMi}(t)] dF_d(t))$. This is the effective throughput of the system in state i. If we assign this as the reward rate in state i and compute $E[Y(t)]$, it yields the expected number of jobs completed successfully until time t, while $E[Y(\infty)]$ yields the expected number of jobs processed until hard-deadline failure.

Thus we see that combining Meyer's work with that of Shin and Krishna makes possible the analysis of real-time fault-tolerant systems, so that we can consider different types of failures in a single model. We can consider soft deadlines by

where X is the random variable corresponding to the steady-state reward rate. For a detailed discussion of using Markov reward models for evaluating computer systems, see Smith, Trivedi, and Ramesh.[2] As an example, by assigning a reward rate of 1 to *up* states and 0 to *failure* states, $E[X(t)]$ yields the instantaneous system availability and $E[X]$ the steady-state system availability. Alternatively, we might wish to compute the accumulated reward $Y(t)$ over the interval $[0,t)$, where $Y(t)$ is given by

$$Y(t) = \int_0^t X(\tau) d\tau = \int_0^t r_{z(\tau)} d\tau$$

It is possible to compute the expected accumulated reward $E[Y(t)]$ by using the following expression:

$$E[Y(t)] = \sum_i r_i \int_0^t P_i(\tau) d\tau$$

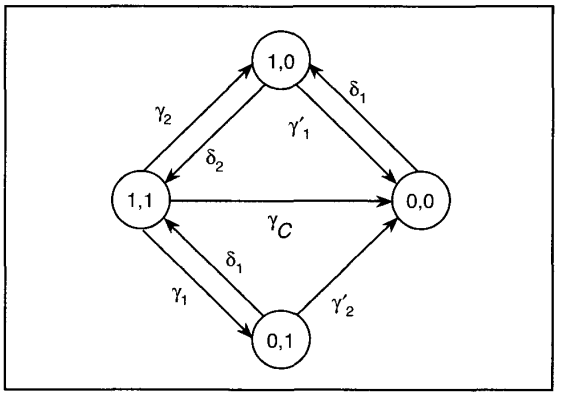
This lets us determine how much total work has been done by time t. We can also consider the time-averaged measure $E[Y(t)/t]$.

Computing the distribution of $Y(t)$ is comparatively difficult.[2] On the other hand, if we consider a system with absorbing failure states, we could compute the distribution of accumulated reward until absorption, $P[Y(\infty) < y]$, using a technique proposed by Beaudry.[3] In this technique we divide each transition rate emanating from a state by the reward rate associated with that state. The distribution of time to absorption for this transformed Markov chain yields the distribution of $Y(\infty)$ for the original Markov chain.

The question remaining is, what constitutes an appropriate reward assignment? We saw earlier that binary reward assignment (0 and 1) yields traditional reliability/availability measures. We could assign the performance levels as reward rates. In the above example let's assume that the service rate of the two processors P1 and P2 is μ_1 and μ_2, respectively. A capacity-based reward rate assignment so that $r_{i,j} = i\mu_1 + j\mu_2$ yields the computational availability.[3] Alternatively, if we assume a more general structure for the performance model, we can compute the throughput in each system state. If we set $r_{i,j}$ to be the system's throughput in state i, j, then we can obtain the throughput-oriented availability.[4]

References

1. K.S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice Hall, Englewood Cliffs, N.J., 1982.
2. R.M. Smith, K.S. Trivedi, and A.V. Ramesh, "Performability Analysis: Measures, an Algorithm, and a Case Study," *IEEE Trans. Computers*, Vol. C-37, No. 4, Apr. 1988, pp. 406-417.
3. M.D. Beaudry, "Performance-Related Reliability Measures for Computing Systems," *IEEE Trans. Computers*, Vol. C-27, No. 6, June 1978, pp. 540-547.
4. J.F. Meyer, "Closed-Form Solutions of Performability," *IEEE Trans. Computers*, Vol. C-31, No. 7, July 1982, pp. 648-657.



Continuous-time Markov chain for the two-processor system.

combining throughput and response-time distributions into the reward rates, and we can consider dynamic failures due to deadline violations. Failure due to imperfect coverage can also be modeled. The method applies to multiple-resource models as well. Nevertheless, for successful analysis, we must be able to generate and solve large, complex Markov chains. In the next section we describe our method, using an example.

We realize that we must compute the response-time distributions for tasks in a system with multiple resources. Computing the response-time distribution in general is difficult. Closed-form expressions are available only for simple queueing systems. For networks of queues, numerical solution using Markov chains appears to be the only possible method. To automatically generate and solve the Markov model for computing response-time distribution, we use stochastic Petri nets. Finally, the assignment of rewards to the failure-repair Markov model is essential to our approach. Thus, we must automatically generate reward rates for the Markov chain that is generated automatically from a stochastic Petri net. We use an extension of stochastic Petri nets, called stochastic reward nets, for this purpose.

Modeling an on-line transaction processing system

On-line transaction processing systems have become a major application area for computers. They are needed when many users require instant access to information such as records in large databases. Examples include airline reservation systems and automated bank-teller systems. These systems are characterized by high-throughput and high-availability requirements.

Figure 2 shows a typical architecture for

Stochastic reward nets

A stochastic reward net is an extension of a stochastic Petri net. The latter, in turn, is an extension of a Petri net. We will briefly introduce Petri nets and stochastic Petri nets and then describe some structural and stochastic extensions.

Basic terminology. A Petri net is a bipartite diagraph whose nodes are divided into two disjoint sets called places and transitions. Directed arcs in the graph connect places to transitions (input arcs) and transitions to places (output arcs). A *multiplicity* may be associated with these arcs. A marked Petri net is obtained by associating tokens with places. Marking a Petri net means distributing tokens in its places. In a graphical representation of a Petri net, circles represent places, bars represent transitions, and dots or integers in the places represent tokens. Input places of a transition are the set of places that are connected to the transition through input arcs. Similarly, output places of a transition are those places connected to the transition by output arcs.

A transition is considered *enabled* in the current marking if the number of tokens in each input place is at least equal to the multiplicity of the input arc from that place. The firing of a transition is an atomic action in which one or more tokens are removed from each input place of the transition and one or more tokens are added to each output place of the transition, possibly resulting in a new marking of the Petri net. When the transition is fired, the number of tokens deposited in each of its output places is equal to the multiplicity of the output arc. Each distinct Petri net marking constitutes a separate state of the Petri net. A marking is reachable from another marking if there is a sequence of transition firings starting from the original marking that results in the new marking. The reachability set (graph) of a Petri net is the set (graph) of markings reachable from the other markings. In any Petri net marking, a number of transitions may be simultaneously enabled.

Another type of arc in a Petri net is the *inhibitor* arc. An inhibitor arc drawn from a place to a transition means that the transition cannot fire if the place contains at least as many tokens as the multiplicity of the inhibitor arc.

Extensions to Petri nets have been considered by associating firing times with the transitions. By assuming the firing times of the transitions to be exponentially distributed, we get the stochastic Petri net. The underlying reachability graph of a stochastic Petri net is isomorphic to a continuous-time Markov chain. Further generalization of stochastic Petri nets has been introduced by Marsan et al.,[1] allowing transitions to have either zero firing times (immediate transitions) or exponentially distributed firing times (timed transitions), in turn giving rise to the generalized stochastic Petri net. In the figures accompanying this article, hollow rectangles represent timed transitions, while thin bars represent immediate transitions. The markings of a generalized stochastic Petri net are of two types: A marking is *vanishing* if only immediate transitions are enabled in the marking and *tangible* if only timed transitions or no transitions are enabled in the marking. Conflicts among immediate transitions in a vanishing marking are resolved using a random switch.[1]

Structural extensions. Ciardo et al.[2] introduced several structural extensions to Petri nets that increased their modeling power. These include enabling functions, general marking dependency, and variable cardinality arcs.

Enabling function. A Boolean enabling function $E(.)$ is associated with each transition. Whenever a transition satisfies all the input and inhibitor conditions in a marking M (that is, when the input places contain enough tokens to enable the transition, and the number of tokens in the inhibitor places is less than the multiplicity of the corresponding inhibitor arc), the enabling function is evaluated. The transition is considered enabled only if the enabling function $E(M)$ = true. Enabling functions are very useful in expressing complex interdependencies, simplifying the model structure, and implementing state truncation.

Variable cardinality arc. The multiplicity of an arc in a Petri net can be defined as a function of the Petri net marking. This facility is useful in simplifying the Petri net's structure. It is especially useful if an input place needs to be flushed. This facility can be used with input arcs, output arcs, and inhibitor arcs.

Marking dependency. The rates and probabilities of transi-

an on-line transaction processing system. The system's front end comprises a transaction generator — a terminal or a bar code reader — and transaction processors that analyze the submitted transaction to determine the information needed from the databases and to provide error recovery capabilities. The system's back end consists of a set of database processors that read and update the records in the databases according to the requests submitted by the transaction processors. The transactions visit the transaction processors and database processors in succession until the necessary processing is completed. Thus, a good measure of performance for an on-line

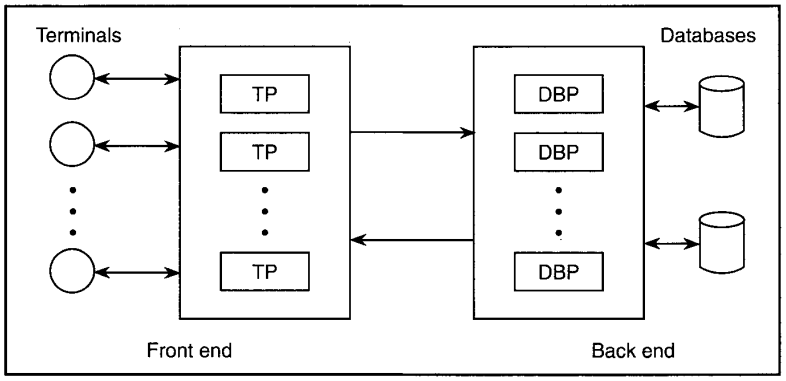


Figure 2. Architecture of an on-line transaction processing system.

tions can also be defined as general functions of the marking of a stochastic Petri net. This would be very useful in modeling a multiple-server queue, for example.

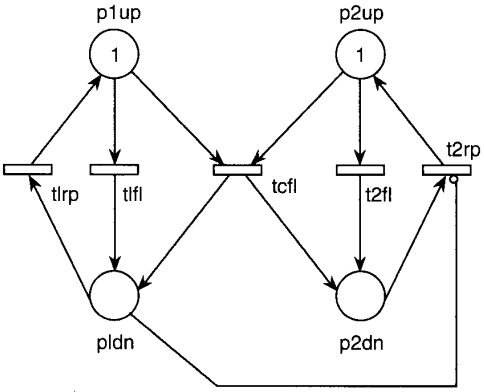
Stochastic extensions. We are interested in automatically generating Markov reward models. For this purpose stochastic Petri nets have been extended by associating reward rates with the markings to obtain stochastic reward nets.[2] The reward rate definitions are specified at the net level as a function of net primitives, such as the number of tokens in a place or the rate of a transition. For each marking of the net, the reward rate function is evaluated and the result is assigned as the reward rate for that marking. The underlying Markov model is then transformed into a Markov reward model, thus permitting evaluation of performance and availability both separately and in combination. We associate a reward rate r_i with every tangible marking i. Since the probability of being in a vanishing marking is zero, there is no need to assign reward rates to vanishing markings. We can then compute $E[X]$, the expected steady-state reward rate; $E[X(t)]$, the expected instantaneous reward rate; $E[Y(t)]$, the expected value of the accumulated reward; $E[Y(\infty)]$, the mean of the accumulated reward until absorption; and $P[Y(\infty) \leq y]$, the distribution of the accumulated reward until absorption.

Note that the definition of reward rates is orthogonal to the analysis type used. Thus, with the same reward definition we can compute the steady-state expected reward rate as well as the instantaneous reward rate at time t, the expected accumulated reward, and the expected time-averaged reward over the interval $[0,t)$.

The figure in this sidebar shows the stochastic reward net model corresponding to the two-processor example considered earlier. In this figure timed transitions t1fl, t2fl, and tcfl represent the failure of processors P1 and P2, and the common-mode failure, respectively. Transitions t1rp and t2rp represent the repair of the two processors. The inhibitor arc from place p1dn to transition t2rp implements the repair priority of P1 over P2. The firing rates of transitions t1fl and t2fl are marking dependent, as shown in the figure. We see that when processor P2 is up, that is, when place p2up contains a token, the firing rate of t1fl is γ_1. However, when P2 has failed, that is, when place p2up is empty, the firing rate is γ_1'. The rate function for transition t2fl is similarly defined.

References

1. M.A. Marsan, G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Trans. Computing Systems*, Vol. 2, No. 2, May 1984, pp. 93-122.
2. G. Ciardo, J. Muppala, and K. Trivedi, "SPNP: Stochastic Petri Net Package," *Proc. Third Int'l Workshop Petri Nets and Performance Models*, CS Press, Los Alamitos, Calif., Order No. 2001, 1989, pp. 142-151.



Transition	Rate Function
t1fl	γ_1 if (#(p2up) = 1) γ_1' otherwise
t2fl	γ_2 if (#(p1up) = 1) γ_2' otherwise

Stochastic reward net model for the two-processor system.

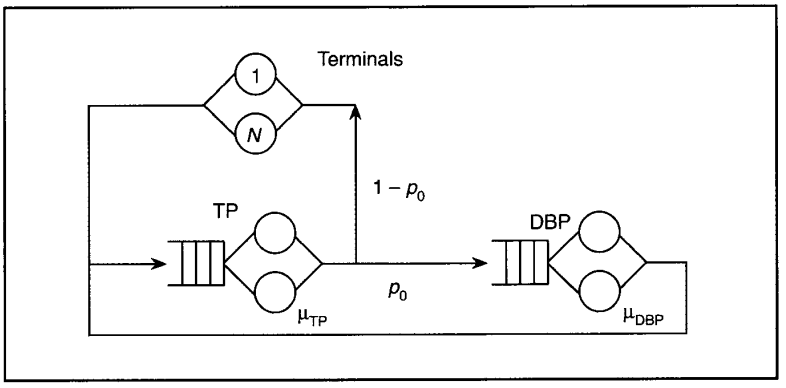
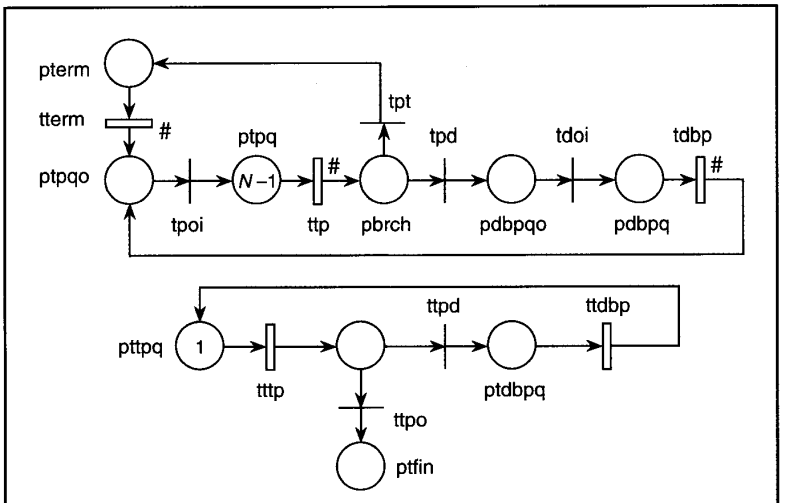


Figure 3. Closed queueing network model of the on-line transaction processing system.



Transition	Enabling Function
tpoi	$(\#(pttpq) = 0 \vee \#(ptpq) < N_{TP}-1) \wedge (\#(ptbr) = 0) \wedge (\#(ptfin) = 0)$
tdoi	$(\#(ptdbpq) = 0 \vee \#(pdbpq) < N_{DBP}-1) \wedge (\#(ptfin) = 0)$
tttp	$(\#(ptpq) < N_{TP}) \wedge (\#(ptfin) = 0)$
ttdbp	$(\#(pdbpq) < N_{DBP}) \wedge (\#(ptfin) = 0)$
Others	$\#(ptfin) = 0$

Transition	Rate Function
tterm	$\#(pterm)\lambda$
ttp	$\min(N_{TP},\#(ptpq))\mu_{TP}$
tdbp	$\min(N_{DBP},\#(pdbpq))\mu_{DBP}$

Arcs	Variable Arc Function
ptpqo → tpoi & tpoi → ptpq	1 if (#(pttpq) > 0) max(1,#(ptpqo)) otherwise
pdbpqo → tdoi & tdoi → pdbpq	1 if (#(ptdbpq) > 0) max(1,#(pdbpqo)) otherwise

Figure 4. Stochastic reward net model for computing the response-time distribution.

transaction processing system is the response time of a transaction.

Next we model various aspects of the on-line transaction processing system, including the response-time distribution for a transaction and the effects of soft-deadline and hard-deadline failures.

Response-time distributions. The performance of the on-line transaction processing system can be studied using the queueing network model shown in Figure 3, since the system involves contention for resources. In this model it is assumed that the transaction processors share a single queue from which transactions are selected for processing using a scheduling discipline that satisfies product-form assumptions.[1]

The transaction processors are modeled using a multiserver queue with the number of servers equal to the number of transaction processors. The database processors are similarly configured. The service times of the transaction processors are exponentially distributed with mean $1/\mu_{TP}$; the service times of the database processors are also exponentially distributed with mean $1/\mu_{DBP}$. The average time between completion of a transaction and submission of the next transaction at a terminal (equivalent to the "think time" at the terminal) is also exponentially distributed with mean $1/\lambda$.

The number of terminals available in the system is assumed to be N. A transaction completing execution at the transaction processor may visit the database processor with probability p_0 or complete execution and return to the terminals with probability $1-p_0$. Since this queueing network obeys product-form assumptions, we could use efficient algorithms such as mean value analysis[1] to compute such steady-state performance measures as average throughput, average queue length at each queue, and average response time for a transaction. Given that there are i transaction processors and j database processors in the system, we can compute the average system throughput T_{ij}. This measure gives the average number of transactions completing in a unit time.

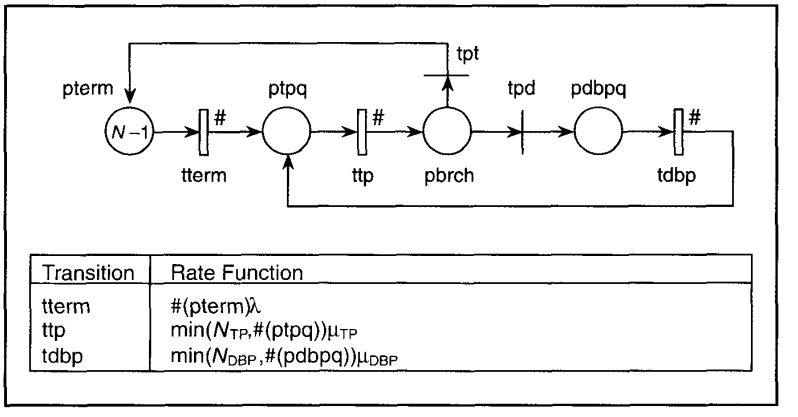
Although computing the averages of various performance measures is easy, this does not present the complete picture, especially with respect to on-line transaction processing systems. It then becomes necessary to evaluate the response-time distributions to obtain the percentiles. Computing response-time distributions for a queueing network with arbitrary structure is very difficult. Closed-form solutions are available for only a small subset of queue-

ing networks.[2] However, it is possible to numerically evaluate the response-time distributions using the *tagged-customer* approach.[7] In this method a target customer is chosen whose movement through the network is observed to compute the response-time distribution. We also adopt the tagged-customer approach to numerically evaluate the response-time distribution for the on-line transaction processing system. We will assume a first come, first served service discipline at the transaction processor queue and database processor queue. The method can be easily extended to other service disciplines.

The stochastic reward net shown in Figure 4 is used to implement the tagged-customer approach. In the figure, #(p) represents the number of tokens in place p, and # associated with a transition means that its service rate is marking dependent.

Let's look at how the position of the tagged customer is tracked through the network. In the figure the places ptpqo, ptpq, and pttpq and the transitions tpoi, ttp, and tttp together implement the multiserver first come, first served queue corresponding to the transaction processors. When the tagged customer is in the queue, all customers ahead of it are represented by the tokens in place ptpq (for convenience we can call this place the inner queue), and customers behind the tagged customer are in place ptpqo (we can refer to this place as the catchment area). The tagged customer will be scheduled for service only when the number of customers ahead becomes less than the number of transaction processors. This procedure is implemented by the enabling function controlling the firing of transition tttp. All customers arriving after the tagged customer are deposited in place ptpqo. Whenever a transaction processor becomes available and the tagged customer is already in service, another customer can be admitted to the inner queue from the catchment area and scheduled for service. This is implemented by allowing the transition tpoi to fire, taking one token out of ptpqo and depositing a token in ptpq. If the tagged customer is not in the queue, an arriving customer can proceed directly from the catchment area to the inner queue. This complex structure is implemented using the variable arc function defined in Figure 4. A similar structure is used for the database processor queue.

Since this is a closed product-form network, by applying the Sevcik-Mitrani (Lavenberg-Reiser) arrival theorem,[1,8] we know that an arriving customer sees the network in equilibrium with one less customer; that is, if we are evaluating the system with N customers, the tagged customer sees the network in equilibrium with $N-1$ customers. When the tagged customer arrives at the transaction processor queue, it sees the network in equilibrium with i customers in the transaction processor queue, j customers in the database processor queue, and the remaining $N-(i+j+1)$ customers at the terminals. Thus, the steady-state probabilities for all the states $(i, j, N-(i+j+1))$, where $0 \leq i, j \leq N-1$, are needed. These probabilities can be calculated by solving the stochastic reward net shown in Figure 5 with $N-1$ customers.



Transition	Rate Function
tterm	#(pterm)λ
ttp	$\min(N_{TP},\#(ptpq))\mu_{TP}$
tdbp	$\min(N_{DBP},\#(pdbpq))\mu_{DBP}$

Figure 5. Stochastic reward net model of the closed queueing network.

Figure 5 represents a stochastic reward net model of the queueing network corresponding to the on-line transaction processing system. In this network, places ptpq and pdbpq represent the queues for the transaction processors and database processors, respectively. Place pterm represents the terminals. Transitions ttp and tdbp represent the service time at the transaction processors and database processors, respectively, and transition tterm corresponds to the think time at the terminals. Routing of the customer to the database processor or the terminal upon completion of service at the transaction processor is implemented by place pbrch and transitions tpd and tpt, respectively.

The response-time distribution is computed as the expected instantaneous reward rate $E[X(t)]$ by associating a reward rate of 1 with those markings where place ptfin is nonempty and a reward rate of 0 for all other markings. The mean time to absorption (token appearing in place ptfin) gives the mean response time for the queueing network. This has been used to validate the stochastic reward net model, since the mean response time can be computed using other product-form solution methods like mean value analysis or convolution.[1]

The response-time distribution $P[R \leq t]$ for various configurations is shown in Figure 6 (the label $[T, D]$ represents the number of transaction processors and the number of database processors, respectively). For this example we set $\lambda = 1.0$ per second, $\mu_{TP} = 50.0$ per second, $\mu_{DBP} = 20.0$ per second, and $p_0 = 0.8$. We assume that the number of transaction processors is equal to the number of database processors. We also assume that with every additional transaction processor, the number of terminals increases by five. Note that the response time decreases with an increase in the number of processors even if the number of terminals is correspondingly increased. Table 1 shows the size of the Markov chains and the number of arcs in the Markov chain for each case. The maximum size of solvable problems is limited by the amount of memory available on the computer and not by the method.

Modeling soft- and hard-deadline failures. Earlier we showed a method for computing the response-time distribution for a transaction. In characterizing real-time systems, we often wish to compute the probability that a transaction will fail to meet a deadline. As stated earlier, a typical requirement for a transaction processing system is for 95 percent of the transactions to complete within a second. This is often characterized by the deadline violation probabilities, which in turn can be easily computed from the response-time distribution. Failure to meet a deadline can be due to several reasons. Resources avail-

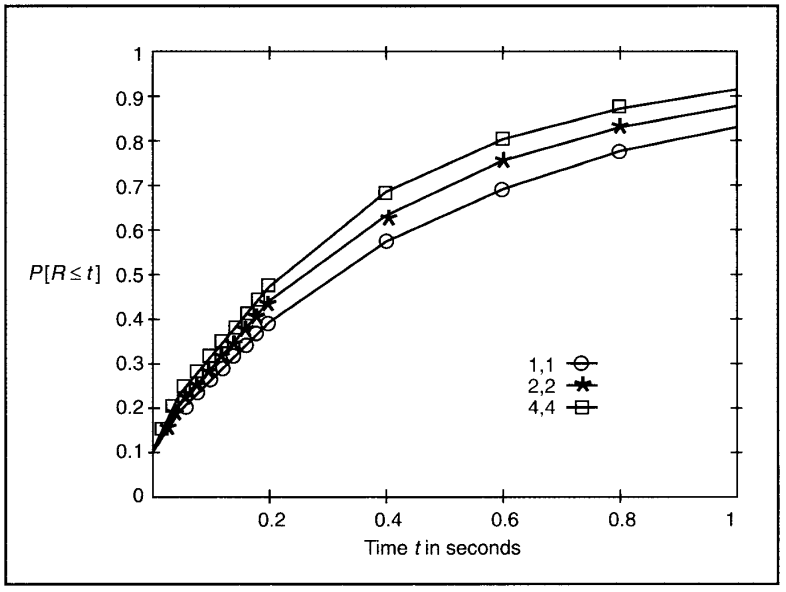
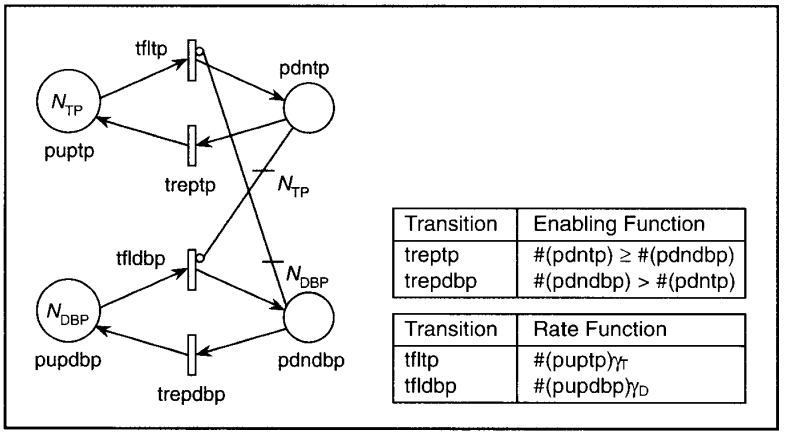


Figure 6. Response-time distribution for the on-line transaction processing system.

Table 1. Sizes of the Markov chains for different configurations.

No. of TPs/DBPs	No. of Terminals	No. of States	No. of Arcs
1	5	85	205
2	10	405	1,305
3	15	1,088	3,722
4	20	2,262	7,998
8	40	14,428	53,932



Transition	Enabling Function
treptp	#(pdntp) ≥ #(pdndbp)
trepdbp	#(pdndbp) > #(pdntp)

Transition	Rate Function
tfltp	#(puptp)γ_T
tfldbp	#(pupdbp)γ_D

Figure 7. Failure-repair behavior of the transaction processors and the database processors.

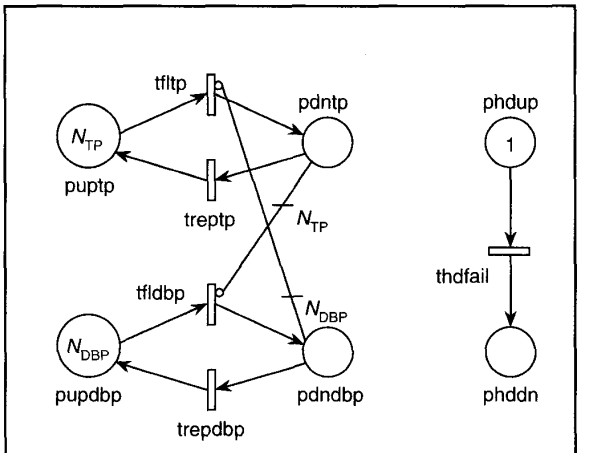
able in a system are subject to failure and repair. Transactions may face inordinate delays, or, in the worst case, be rejected if the system is unavailable when they arrive. Even if a fault-free system provides satisfactory service, a degraded configuration may not. Depending on the nature of the system, soft or hard deadlines may best describe its behavior.

Now let's consider the performance of the on-line transaction processing system in the presence of resource failures and repairs. We will assume that only the processors are subject to faults and the remaining system, including the interconnection networks and the communication medium, is fault free. This is a reasonable assumption, since failure rates for the rest of the components are much lower than those of the processors. We consider both soft- and hard-deadline violations with the implicit assumption that a hard-deadline violation is catastrophic.

The failure-repair behavior of the transaction processors and the database processors is modeled by the stochastic reward net model shown in Figure 7. In this model we assume that the failed transaction processors and database processors share a single repair facility. When both types of processors fail, priority for repair is given to whichever type has the most failures. When an equal number of transaction processors and database processors fail, transaction processors get a higher priority. This is reflected by the enabling functions shown in the figure. The times to failure for the transaction processors and database processors are exponentially distributed with rates γ_T and γ_D, respectively. The corresponding repair times are also exponentially distributed with rates β_T and β_D, respectively.

We assume that transactions in the system must meet a fixed deadline τ. First the soft-deadline case is considered. We assume that violation of the soft deadline results in the transaction's being aborted. The system continues to function, however. We will assign a reward rate of $T_{ij}\,P[R_{ij} \le \tau]$ to the stochastic reward net marking having i tokens in place puptp and j tokens in place pupdbp. Here, T_{ij} is the throughput of the on-line transaction processing system with i transaction processors and j database processors, and $P[R_{ij} \le \tau]$ is the probability that the response time is less than τ. The reward rate assigned corresponds to the average number of transactions completing in a unit time that meet the deadline.

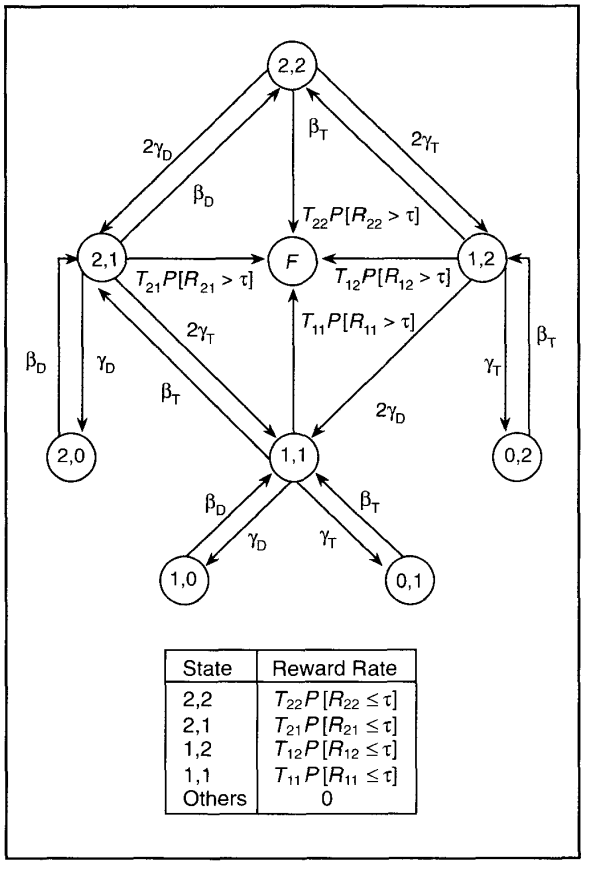
We assume implicitly that whenever a



Transition	Enabling Function
treptp	(#(pdntp) ≥ #(pdndbp)) ∧ (#(phdup) = 1)
trepdbp	(#(pdndbp) > #(pdntp)) ∧ (#(phdup) = 1)
thdfail	(#(puptp) > 0) ∧ (#(pupdbp) > 0)
Others	(#(phdup) = 1)

Transition	Rate Function
tfltp	#(puptp)γ_T
tfldbp	#(pupdbp)γ_D
thdfail	$T_{ij}P[R_{ij} > \tau]$

Figure 8. Failure-repair behavior with hard-deadline failures.



State	Reward Rate
2,2	$T_{22}P[R_{22} \leq \tau]$
2,1	$T_{21}P[R_{21} \leq \tau]$
1,2	$T_{12}P[R_{12} \leq \tau]$
1,1	$T_{11}P[R_{11} \leq \tau]$
Others	0

Figure 9. Markov chain of failure-repair behavior for a 2 × 2 system.

transaction arrives while the system is in a particular configuration with i transaction processors and j database processors, it will complete in the same configuration. This is a reasonable assumption, since failure rates for the processors are very small. Thus, from the viewpoint of a transaction, the system is assumed to be in a quasi-stable state. Shin and Krishna used a similar assumption.[5]

If we consider the deadline to be hard, then we assume that the system will fail if this deadline is violated. In this case the system's failure-repair behavior is modified, as shown in Figure 8. In this figure the failure rate of the transition thdfail is set to $T_{ij}P[R_{ij} > \tau]$ — the rate at which transactions fail to meet the deadline. This is analogous to the failure rate for the hard-deadline violation considered by Shin and Krishna. Note, however, that we have considered a fixed deadline. This procedure can be extended easily to a case in which the deadline itself is a random variable, as with Shin and Krishna. They considered an open queueing system, so the throughput was configuration independent (equal to λ). The response-time distribution was available as a closed-form expression for the simple queue they considered, whereas the response-time distribution had to be computed numerically for our queueing system. In Figure 9 we show the failure-repair Markov chain (structure-state process) corresponding to a system with two transaction processors and two database processors. This Markov chain corresponds to a case in which hard-deadline failures are considered. States (2,0), (1,0), (0,1), and (0,2) are considered system down states. The reward rates assigned to the Markov chain states are also shown explicitly.

We consider various measures to characterize the on-line transaction processing system in the presence of failures and repairs. In our numerical example, we consider a system with four transaction processors and four database processors. The failure rate for the processors is assumed to be 0.01 per hour; the repair rate is 2 per hour. Figure 10 shows a plot of the expected time-averaged system throughput $E[Y(t)/t]$ as a function of time for both the soft- and hard-deadline cases, where we plot the cases when the deadline τ is 1 and 2 seconds. As a comparison, we also plot the expected time-averaged throughput for the system with no deadlines. This represents the upper bound on the overall system throughput that can be achieved. From the figure, we notice that as the deadline is relaxed, the throughputs for the systems with deadlines move closer to this upper bound. As expected, the system with soft deadlines outperforms the system with hard deadlines, since a hard-deadline violation is catastrophic.

Figure 11 plots the probability of system failure, $P_{SF}(t)$, as a function of time for both soft- and hard-deadline cases for a system with four transaction processors and four database processors. In this exam-

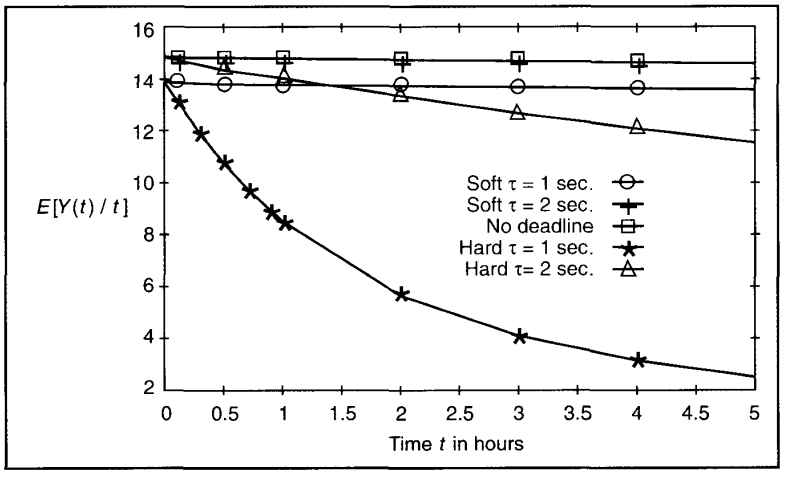


Figure 10. Throughput with deadlines as a function of time.

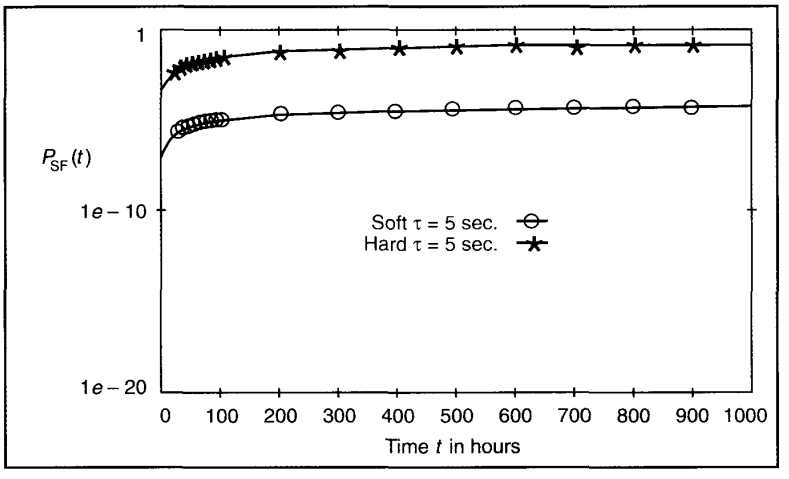


Figure 11. Unreliability as a function of time.

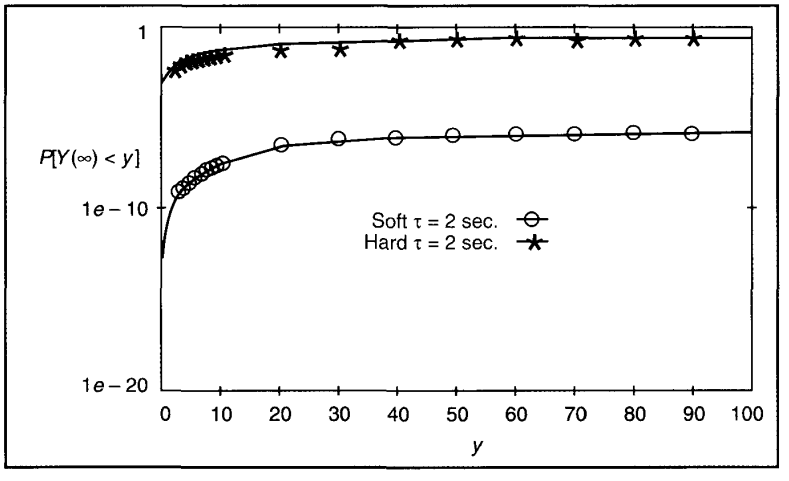


Figure 12. Distribution of number of tasks successfully completed until absorption.

ple we consider as absorbing states those (static failure) states in which all the transaction processors or all the database processors have failed. The figure shows that hard-deadline failure (dynamic failure) is the leading cause of system failure.

In Figure 12 we plot the distribution of number of tasks successfully completed until absorption for both the soft- and hard-deadline cases for a system with four transaction processors and four database processors. The figure shows that the probability of failing to accumulate reward y for the hard-deadline case is much higher than for the soft-deadline case. This is to be expected, since hard-deadline failures are catastrophic.

We have presented some interesting techniques for evaluating real-time systems. These techniques combine the effects of performance, reliability/availability, and deadline violation into a single model. We use a numerical method for computing the response-time distribution for a queueing network. Using throughputs and response-time distributions as reward rates, we have been able to handle systems with multiple resources as well as soft- and hard-deadline constraints. Transient analysis of Markov reward models is the basis of our analysis. Stochastic reward nets are used to generate and solve these models.

Our technique can be extended to cases with multiple types of tasks and mixed hard and soft deadlines. Processing power and cost-based trade-off studies, along with other such design optimizations, can be carried out much as Shin and Krishna demonstrated. Imperfect coverage, failure dependencies, and complex fault-handling behavior can also be taken into account.■

References

1. S.S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, New York, 1983.

2. O.J. Boxma and H. Daduna, "Sojourn Times in Queueing Networks," in *Stochastic Analysis of Computer and Communication Systems*, H. Takagi, ed., Elsevier Science Publishers (North-Holland), Amsterdam, 1990, pp. 401-450.

3. J.F. Meyer, "Closed-Form Solutions of Performability," *IEEE Trans. Computers*, Vol. C-31, No. 7, July 1982, pp. 648-657.

4. R.M. Smith, K.S. Trivedi, and A.V. Ramesh, "Performability Analysis: Measures, an Algorithm, and a Case Study," *IEEE Trans. Computers*, Vol. C-37, No. 4, Apr. 1988, pp. 406-417.

5. K.G. Shin and C.M. Krishna, "New Performance Measures for Design and Evaluation of Real-Time Multiprocessors," *Computer Systems Science and Eng.*, Vol. 1, No. 4, Oct. 1986, pp. 179-192.

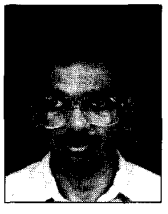
6. G. Ciardo, J. Muppala, and K. Trivedi, "SPNP: Stochastic Petri Net Package," *Proc. Third Int'l Workshop Petri Nets and Performance Models*, CS Press, Los Alamitos, Calif., Order No. 2001, 1989, pp. 142-151.

7. B. Melamed and M. Yadin, "Randomization Procedures in the Computation of Cumulative-Time Distributions over Discrete-State Markov Processes," *Operations Research*, Vol. 32, No. 4, July-Aug. 1984, pp. 926-944.

8. K.C. Sevcik and I. Mitrani, "The Distribution of Queueing Network States at Input and Output Instants," *J. ACM*, Vol. 28, No. 2, Apr. 1981, pp. 358-371.

Steven P. Woolet is working on system performance analysis for IBM at Research Triangle Park in North Carolina. He is also working toward a PhD in electrical engineering at Duke University. His research interests include reliability, and performance and performability modeling of networks and multiprocessor systems. Woolet received bachelors' degrees in mathematics and electrical engineering from Bethel College, Indiana, and the University of Notre Dame, respectively, and an MS in electrical engineering from the University of Minnesota. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Kishor S. Trivedi is a professor of electrical engineering and computer science at Duke University. He has served as a principal investigator on various projects and as a consultant to industry and to research laboratories. He is the author of *Probability and Statistics with Reliability, Queueing, and Computer Science Applications* (Prentice Hall). Both the text and his related research activities focus on computing-system reliability and performance evaluation. Trivedi has a B.Tech degree from the Indian Institute of Technology, Bombay, and MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE Computer Society and a senior member of IEEE.



Jogesh K. Muppala is a student in the Department of Electrical Engineering at Duke University, where he is completing requirements toward a PhD. His research interests include stochastic Petri nets, performance and dependability modeling, and multiprocessor systems. He received a BE in electronics and communication engineering from Osmania University, Hyderabad, India, in 1985 and an MS in computer engineering from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, Louisiana, in 1987. He is a student member of the IEEE, the IEEE Computer Society, and the ACM.

Readers can contact Muppala and Trivedi at Duke University, Dept. of Electrical Engineering, Durham, NC 27706; e-mail jkm@egr.duke.edu or kst@cs.duke.edu. Woolet is with IBM at Research Triangle Park, NC 27709