Performance Analysis of the CORBA Event Service Using Stochastic Reward Nets

Srinivasan Ramani, Kishor S. Trivedi
Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708-0291, USA
{sramani, kst}@ee.duke.edu
Phone: (919) 660-5269
Fax: (919) 660-5293

Balakrishnan Dasarathy
Telcordia Technologies, Inc.
445 South Street
Morristown, NJ 07960, USA
bdasarat@telcordia.com
Phone: (973) 829-5038
Fax: (973) 829-2645

Abstract

The Event service is the earliest CORBA solution to the message queue model of communication in distributed systems. Typical implementations however suffer from the lack of event delivery guarantees. The loss of messages is aggravated in the presence of burstiness in the input to the Event service, and occurrences of isolated bursts of traffic could also have serious effects. In this paper we develop stochastic reward net (SRN) models that can aid in the study and configuration of the Event service to conform to design specifications. To capture burstiness in the input, Markov modulated Poisson process (MMPP) is used as the input source. Erlang distributed event consumption times are used in the models to accommodate more general distributions and a wider range of variances. The models also take into consideration the FIFO discard policy adopted in many Event service implementations. The SRN models are solved using the tool SPNP. The applicability of the models to the CORBA Notification service is also briefly discussed.

1 Introduction

The Common Object Request Broker Architecture (CORBA) [25], [18] spear-headed by the Object Management Group (OMG) [15] is an industry-wide standard that facilitates portable, distributed object-oriented computing. The COS Event service which is one of the Common Object Services (COS) [16] provided in the CORBA specification, is the earliest CORBA solution to a message queue model of communication [11]. The Event service provides decoupled communication between suppliers and consumers. The current CORBA specifications [18] do not address reliability or fault-tolerance issues. This has led to a lot of research on providing high reliability and availability in CORBA using fault tolerance, mainly by means of object replication [9], [14].

Like other aspects of CORBA, the COS Event service specification does not address reliability issues [5], [23]. Specifically, the COS Event service specification does not require vendor implementations to provide any event delivery guarantees. This has led to the development of techniques to recover from lost events, such as re-sending the lost event [4], and resynchronization of state information between suppliers and consumers [22]. More recently, the OMG defined a Notification service [17] that seeks to extend the Event service and also provide configurable reliability and quality of service (QoS) features. But, the Event service is the most prevalent CORBA messaging solution today as it is the most widely available on different platforms with different language bindings. Moreover, some QoS settings in the Notification service permit absence of delivery guarantees, necessitating careful determination of operating ranges with the Notification service as well.

In this paper, we study the performance of the COS Event service using a formalism called stochastic reward nets (SRN) to model its behavior. To keep the discussion simple we consider one supplier and one consumer. The rest of the paper is organized as follows. In Section 2 we provide an overview of the Event service, define the measures of interest, and give a brief introduction to SRN. In Section 3 we develop SRN models for the basic Event service and the Event service that is augmented by a state resyn-
chronization scheme to recover from event drops. Section 4 explains the use of the models in practice by configuring an example system. In a messaging environment, like any queuing system, the input may often be bursty and correlated. The illustrations in Section 4 consider bursty input sources along with Poisson arrival sources. In addition, the effect of an isolated burst of messages is studied. Section 5 discusses the applicability of the models developed in this paper to the COS Notification service, and Section 6 concludes the paper.

2 Background

2.1 CORBA COS Event service

The CORBA Event service is one of the earliest CORBA solutions to messaging. The logical view of the “push” model [16] of the Event service with one supplier and one consumer connected to it is shown in Figure 1. Suppliers of events connect to proxy consumers and consumers of events connect to proxy suppliers, both of which are objects within the Event channel object. The Event channel object itself is part of an Event service object. An Event service could consist of multiple Event channels. The proxies decouple the suppliers and consumers from each other. Events that are destined to a consumer are queued within the Event channel in proxy supplier queues (PSQ), one for each consumer. In many Event service implementations, for example PeerLogic’s LiveContent Broker Event service [10] and Inprise’s Visibroker Event service [26], when the PSQ fills up and a new event bound for its consumer enters the Event channel, the event at the front of the queue (the oldest) is discarded to accommodate the newest event. This is unlike most queuing models in literature in which incoming messages are discarded when the queue is full. Under heavy load conditions, event drops from the PSQ could be quite common. The SRN models presented in this paper take this peculiarity into account.

In addition, some Event service implementations such as PeerLogic’s LiveContent Broker [10] provide a retry mechanism to log discarded events and re-send them later. But this will not preserve event ordering. This might be unacceptable to some applications. By carefully configuring the PSQ and limiting the supplier rate, the possibility of an event drop can be reduced. Additional consideration should be given to accommodate bursty input and also to deal with an occasional burst in input rate. In this paper we provide examples to illustrate how our model can be used for the above purposes.

Configuring the Event service to minimize the possibility of event drop still does not provide absolute guarantees. A resynchronization scheme is described in [22] as a methodology to recover from dropped events if such a state resynchronization between suppliers and consumers is possible and sufficient. In this scheme, when the consumer detects a dropped event (using its sequence number), it requests a resynchronization from the supplier. The supplier obliges by sending the latest status information of the objects it manages. If such a scheme is employed, the rate at which resynchronization can be done is another parameter to be determined. We extend the SRN model of the basic Event service to include the resynchronization scheme and study the effects.

2.2 Measures of interest

While evaluating an Event service configuration the measures of interest include the expected PSQ length, probability that the PSQ is full, probability of a message loss at time $t$, message loss rate, and message retention rate. Let $S$ denote the state space of the queuing system. Let $p_i(t)$ denote the probability of being in state $i$ and $N_i(t)$ denote the number of events in PSQ while in state $i$, both at time $t$. In general, $i$ represents a vector of parameters of the system sufficient enough to represent the system at the desired level of granularity to obtain the required measures. The expected proxy supplier queue length at time $t$ is then given by,

$$PSQlen(t) = \sum_i N_i(t) * p_i(t).$$  

The probability that the PSQ is full at time $t$ is given by,

$$PSQfull(t) = \sum_{i \in S_K} p_i(t),$$

where $S_K$ is the subset of $S$ such that the PSQ length of all states in $S_K$ is $K$, the maximum size of PSQ. The probability that an incoming event is lost at time $t$ is given by,

$$Ploss(t) = P(PSQ\ full\ at\ time\ t \mid\ arrival\ at\ time\ t) = \frac{\sum_{i \in S_K} p_i(t)\lambda_i}{\sum_{j \in S} p_j(t)\lambda_j},$$

where $\lambda_j$ denotes the arrival rate into the queuing system while it is in state $j$. When the inputs to the PSQ constitute Poisson arrivals, i.e., state-independent arrivals, Equation 2 and Equation 3 are equivalent. This is not the case for MMPP arrivals. Message loss rate is the rate of arrivals into the system when the PSQ is full and is defined as,

$$LossRate = \sum_{i \in S_K} p_i(t)\lambda_i. $$

Message retention rate is the rate of arrivals into the system when the PSQ is not full and is defined as,

$$RetentionRate = \sum_{i \notin S_K} p_i(t)\lambda_i.$$
### 2.3 Stochastic reward nets

Stochastic reward nets (SRN) [2] are an extension to Petri nets [21] which were proposed as a formal representation of the control flow in a system. A Petri net is a directed graph containing two types of nodes - places and transitions. A directed arc connecting a place (transition) to a transition (place) is called an input (output) arc. Arcs have a positive integer number called multiplicity associated with them. The default multiplicity of an arc is 1. Places can contain tokens that move from one place to another through transitions. A transition is enabled when each of the places connected to it by its input arcs have at least the number of tokens equal to the multiplicity of those arcs. When an enabled transition fires, a number of tokens equal to the input arc multiplicity is removed from each of the corresponding input places and a number of tokens equal to the output arc multiplicity is deposited in each of the corresponding output places. The state of a Petri net with \( p \) places is represented by a vector \((m_1, m_2, \ldots, m_p)\) called the marking of the Petri net, where \( m_i \) is the number of tokens in place \( i \). When a Petri net is first specified, it can be made to start at a particular marking called the initial marking.

Stochastic Petri nets (SPN) [13] extend Petri nets by allowing “timed” transitions that have exponentially distributed firing times. Generalized stochastic Petri nets (GSPN) [12] also allow immediate transitions. GSPNs include an inhibitor arc which can also have a multiplicity associated with it, that inhibits the transition it is connected to if the place it is connected to at its other end has a number of tokens equal to at least its multiplicity. The default multiplicity is 1. A GSPN marking with at least one immediate transition enabled is called a vanishing marking, while a marking with no immediate transitions enabled is called a tangible marking.

Stochastic reward nets extend GSPNs further by allowing the association of a reward rate to each tangible marking. The tangible markings of an SPN, GSPN, or an SRN and their rates of transition from one marking to another are in fact equivalent to corresponding states and transitions between states of an underlying continuous time Markov chain (CTMC)\(^2\). Hence an SRN can be mapped into an equivalent Markov reward model (MRM). Software tools such as SPNP [3] can automate the translation of an SRN to its equivalent MRM and solve it. The SRN models thus allow the concise specification of various reward functions. To extend the power of specification, SRN includes specification of enabling (or guard) functions for each transition. The transition is enabled only if the enabling function returns “1”. SRN also allows marking dependent arc multiplicities and enabling functions. Another feature of SRN is the provision of priorities and probabilities to determine which of a set of simultaneously enabled transitions will fire first: the transition with the highest priority is fired first, or if the competing transitions have the same priority the one to fire first is chosen probabilistically. Graphically, a place is represented as a circle, \( n \) tokens in a place are represented by \( n \) dots or the number \( n \) within the place, immediate transitions are represented by thin lines, and timed transitions by narrow empty rectangles. An inhibitor arc is represented by a circle instead of an arrow at the terminating end. An arc with multiplicity \( m \) is represented by a “\( m \)” on the arc, and an arc with a marking dependent multiplicity function is indicated by a “N” or an inverted “N” on it. The number of tokens in place \( p \) is indicated as \#\( p \).

### 3 Stochastic reward net models of the Event service

#### 3.1 Queuing model

The basic queuing system that we study is the one shown in Figure 1. The queue in the system is the proxy supplier queue (PSQ). The arrival of events from the supplier can be bursty. To model this burstiness, we use a 2-state (call them states 1 and 2) MMPP process with infinitesimal generator matrix [24] \( Q \) and arrival rates \((\gamma_{s1}, \gamma_{s2})\). The infinitesimal generator matrix is given as,

\[
Q = \begin{bmatrix} -\alpha & \alpha \\ \beta & -\beta \end{bmatrix}
\]  

\[ (6) \]

\(^2\)Thus for an SRN model of a queuing system, each state \( i \) of Section 2.2 corresponds to a tangible marking of the SRN.

![Figure 1. Logical view of the CORBA COS Event service](image-url)
When the MMPP is in state 1 (state 2) the supplier rate is $\gamma_{s1}(\gamma_{s2})$. When $\gamma_{s1} = \gamma_{s2}$, the MMPP models a Poisson source with arrival rate $\gamma_{s1} = (\gamma_{s2})$ corresponding to independent exponentially distributed inter-arrival times. Although we use a 2-stage MMPP process, our models can be easily extended to include MMPP inputs with larger numbers of states. In order to accommodate general consumption time distributions, we let the time taken for an event at the front of the queue to be passed on to the consumer to be Erlang distributed [27] with $r_c$ stages and mean $1/\Upsilon_c$. Thus each stage of the Erlang distribution is independent and exponentially distributed with rate $\tau_c = r_c \cdot \Upsilon_c$. By increasing the number of stages $r_c$, the variance of the consumption time can be decreased while retaining the same mean $1/\Upsilon_c$. A sufficiently high $r_c$ yields a good approximation to a deterministic consumption time, while still retaining the MMPP as a Markovian net making numerical solution tractable.

3.2 Basic Event service

The SRN model of the basic Event service is shown in Figure 2 and is specified in Tables 1, 2, and 3. The MMPP input source is represented by the transitions $t_1$ and $t_2$, and place $P_{\text{mmpp}}$. When place $P_{\text{mmpp}}$ has a token, the rate of transition $t_s$ is set as $\gamma_{s1}$, and when it is empty $t_s$ has a firing rate $\gamma_{s2}$. The inhibitor arc from $P_{\text{mmpp}}$ to $t_2$ prevents $t_2$ from firing when $P_{\text{mmpp}}$ has a token, ensuring that transitions $t_1$ and $t_2$ do not compete with each other. Places $Er_{\text{token}}$ and $Er_{\text{stage}}$ represent an event waiting at the front of the queue to be taken up by the consumer. These places along with the place $P_{\text{psq}}$ represent the proxy supplier queue (PSQ) within the Event channel. In Figure 2 we have modeled the waiting time of an event at the front of the PSQ as $r_c$-stage Erlang distributed, to accommodate more general distributions. For example, if the waiting time is exponentially distributed, $r_c$ will be equal to 1. A deterministic waiting time is approximated well by a sufficiently high number of stages.

The model works as follows. When transition $t_s$ fires, a token is deposited in place $P_{\text{psq}}$. When there is no token waiting to be consumed, places $Er_{\text{token}}$ and $Er_{\text{stage}}$ are empty. This enables the immediate transition $Er_{\text{in}}$. Since the arc from transition $Er_{\text{in}}$ to place $Er_{\text{token}}$ has multiplicity $r_c$, a token is removed from place $P_{\text{psq}}$ and $r_c$ tokens are deposited in place $Er_{\text{token}}$. Each time transition $t_s$ fires, a token from place $Er_{\text{token}}$ is moved to place $Er_{\text{stage}}$. Since the arc from $Er_{\text{stage}}$ to transition $Er_{\text{out}}$ has multiplicity $r_c$, transition $Er_{\text{out}}$ is enabled only when all $r_c$ tokens have been moved to place $Er_{\text{stage}}$. When it is enabled, all the $r_c$ tokens are removed from place $Er_{\text{stage}}$. This represents the $r_c$-stage Erlang distributed time spent by the event at the front of the PSQ before it is consumed.

Note that transition $Er_{\text{in}}$ is disabled when either place $Er_{\text{token}}$ or place $Er_{\text{stage}}$ has a token. This represents the fact that only one event is a candidate for removal at a given time.

A peculiarity of the PSQ in many Event service implementations in comparison with common queueing systems is that when the PSQ is full and a new event arrives, rather than rejecting the new incoming event the event at the front of the queue is discarded to make room for the incoming message. The transitions $Er_{\text{token}}$ empty and $Er_{\text{stage}}$ empty have guard functions that enable the transitions when the number of messages in the PSQ exceeds its size. The arcs that connect places $Er_{\text{token}}$ and $Er_{\text{stage}}$ to the corresponding transitions have variable multiplicity that is set to the number of tokens in the corresponding input places. This arrangement ensures that the places are drained of tokens whenever the PSQ size exceeds its maximum.

3.3 Event service with resynchronization

Figure 3 shows the SRN model of the basic Event service that is augmented with the state resynchronization scheme [22] mentioned in Section 2.1. Tables 4, 5, and 6 give the specifications of the SRN that are different from or in addition to the ones in Tables 1, 2, and 3. The resynchronization consists of a fixed-length sequence of status updates from the supplier that helps the consumer recover from event drops in the Event channel.

In addition to the basic Event service model in Figure 2, the SRN in Figure 3 models the event loss detection and resynchronization. When an event is discarded from the front of the PSQ, a token is deposited in place $P_{\text{loss}}$. The variable multiplicity on the arcs leading to place $P_{\text{loss}}$ ensure that at most one token is present in place $P_{\text{loss}}$ at any given time. This is because when there are multiple event drops, the first drop that is detected sets up a resynchronization and all subsequent messages in the PSQ are ignored by the consumer until the resynchronization is started. Although when an event loss occurs a token is immediately deposited in place $P_{\text{loss}}$, a resynchronization is not initiated until the consumer consumes the next event from the PSQ. This is represented by a token in place $P_{\text{cons}}$. When the consumer consumes an event and a token is placed in $P_{\text{cons}}$, two situations are possible: (1) An event loss has previously occurred, represented by a token in place $P_{\text{loss}}$. In this case transition $t_{\text{loss}}$ is enabled and the tokens in $P_{\text{loss}}$ and $P_{\text{cons}}$ are removed and the number of tokens in place $P_{\text{res}}$ is made equal to the length of the resynchronization sequence, $ResyncSeqLen$. This is achieved by the variable multiplicity arc leading to place $P_{\text{res}}$; (2) No event loss has occurred. In this case the event that was just consumed is in the right order. The transition $t_{\text{add}}$ is enabled in this
Figure 2. Basic Event service

Table 1. SRN specification for the basic Event service - Transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Rate function</th>
<th>Guard function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$s$ (P_{mmpp} \neq 1) (\gamma_{t_1} \cdot \gamma_{t_2})</td>
<td>$#P_{mmpp} + (#\text{Er_token} + #\text{Er_stage})/\gamma_{t_2} &gt; K? 1: 0$</td>
</tr>
<tr>
<td>$\text{Er_token_empty}$</td>
<td>$\gamma_{t_1}$</td>
<td>$#P_{mmpp} + (#\text{Er_token} + #\text{Er_stage})/\gamma_{t_2} &gt; K? 1: 0$</td>
</tr>
<tr>
<td>$\text{Er_stage_empty}$</td>
<td>$\gamma_{t_1}$</td>
<td>$#P_{mmpp} + (#\text{Er_token} + #\text{Er_stage})/\gamma_{t_2} &gt; K? 1: 0$</td>
</tr>
</tbody>
</table>

Table 2. SRN specification for the basic Event service - Arcs

<table>
<thead>
<tr>
<th>Arc</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>return $#\text{Er_token} + #\text{Er_stage}$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>return $#\text{Er_token} + #\text{Er_stage}$</td>
</tr>
</tbody>
</table>

Table 3. SRN specification for the basic Event service - Reward measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Reward rate function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSQLen</td>
<td>$#P_{res} + (#\text{Er_token} + #\text{Er_stage})/\gamma_{t_2}$</td>
</tr>
<tr>
<td>PSQFull</td>
<td>$#P_{res} + (#\text{Er_token} + #\text{Er_stage})/\gamma_{t_2}$</td>
</tr>
<tr>
<td>SupplierTput</td>
<td>return rate($t_1$)</td>
</tr>
<tr>
<td>LossRate</td>
<td>return (PSQLen=K)? SupplierTput : 0.0</td>
</tr>
<tr>
<td>RetentionRate</td>
<td>return (PSQLen=K)? 0.0 : SupplierTput</td>
</tr>
<tr>
<td>Ploss</td>
<td>return LossRate/SupplierTput</td>
</tr>
<tr>
<td>ConsumerTput</td>
<td>return (#\text{Er_token} = 1)? $\gamma_{t_2}$ : 0.0</td>
</tr>
</tbody>
</table>

Figure 3. Resynchronization scheme for the basic Event service

Table 4. SRN specification for the Event service with resynchronization - Transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Rate function</th>
<th>Guard function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Er_token_res_empty}$</td>
<td>$#P_{res} \neq \text{ResyncSeqLen}$</td>
<td>$#P_{res} \neq \text{ResyncSeqLen} \neq 1: 0$</td>
</tr>
<tr>
<td>$\text{Er_stage_res_empty}$</td>
<td>$#P_{res} \neq \text{ResyncSeqLen}$</td>
<td>$#P_{res} \neq \text{ResyncSeqLen} \neq 1: 0$</td>
</tr>
</tbody>
</table>
case since there is no token in place $P_{loss}$ and the token is immediately removed from place $P_{cons}$.

The time between resynchronizations is taken to have an Erlang distribution with $r_{res}$ stages. Like in the case of consumption time, this gives the modeler flexibility in specifying a more general distribution for the time between two successive resynchronization events. The transitions $Er{}^{\text{joker, res empty}}$ and $Er{}^{\text{stage, res empty}}$ have guard functions that enable them when the number of tokens in place $P_{res}$ equals the length of the resynchronization sequence. This is required for the situation in which a resynchronization message reaches the consumer out of sequence, thus setting up a new resynchronization. When resynchronization events are being sent by the supplier, it suspends the normal events modeled by the MMPP source. This is represented by the inhibitor arcs from places $P_{res}$, $Er{}^{\text{joker, res}}$ and $Er{}^{\text{stage, res}}$ to transition $t_s$.

### 4 Application of the models

In this section we illustrate how the SRN models described in the previous section can be applied to a real system. We first show how the basic Event service model in Figure 2 can be used to establish operating ranges. We then show how the model in Figure 3 can be used to determine the maximum permissible resynchronization rate.

#### 4.1 Basic Event service

In [22] a PSQ of size 25 was used and it was determined that for the particular configuration of the basic Event service, the supplier could be allowed to push events at a rate of not more than 49.9 messages per second, if all events are to be received by the consumer without any loss in the PSQ.

### 4.1.1 Exponentially distributed inter-arrival times

In the experiment in [22], the input stream of events had no bursts. This is modeled well by a Poisson input stream whose inter-arrival times are exponentially distributed with rate 49.9 messages per sec. The next step is to estimate the minimum rate at which the consumer must consume messages. Since there is no direct way to determine this “minimum” by experimentation, we solve the model in Figure 2 for various values of the mean consumption time $1/Y_c$. We assume that the consumption time of an event at the front of the PSQ is practically deterministic. This can be modeled by letting $r_c = 25$ (a large number of Erlang stages). A mean consumption time of $1/Y_c$ then translates to $r_c = 25 / Y_c$ in our model.

Figure 4 shows the effect of the mean consumption rate on various measures of interest. Figure 4(a) shows that the steady state PSQ length is high for low mean consumption rates. As the mean consumption rate approaches the supplier rate of 49.9 messages per second, the queue length starts to drop. The corresponding probabilities of message loss are shown in Figure 4(b). We use 70 messages per second as the estimate for the mean consumption rate of the consumer since the corresponding probability of message loss is almost 0.0. Figure 4(c) shows the message loss rate and retention rate. For low mean consumption rates most of the incoming messages are lost. Hence the loss rate is close to the supplier rate of 49.9 messages per second, and the retention rate is close to 0.0. As the mean consumption rate increases, the situation starts to reverse. At the estimated mean consumption rate of 70 messages/second, the loss rate is almost 0.0 and the retention rate is close to the supplier rate of 49.9 messages per second. Finally, Figure 4(d) shows that as the mean consumption rate is increased, the consumer throughput $(\text{ConsumerTput})$ approaches the supplier throughput $(\text{SupplierTput})$.

#### 4.1.2 Bursty arrivals

In actual practice, the Event channel could face an input stream that is bursty, i.e., the input could consist of periods of high and low mean input rates, leading to a high variance in the overall mean supplier throughput. MMPP sources are a very powerful way of modeling bursty and correlated input streams [6], [7]. In order to show the effect of a bursty input, we compare the effect of two sources with the same mean inter-arrival time: (1) a Poisson source whose inter-arrival times are exponentially distributed with rate 49.9 messages/second (as in Section 4.1.1), and (2) a 2-state MMPP source with $\gamma_{s1} = 39.88$ messages per second, $\gamma_{s2} = 100.0$ messages per second, $\alpha = 1/5.0 \text{ sec}^{-1}$. 

---

### Table 5. SRN specification for the Event service with resynchronization - Arcs

<table>
<thead>
<tr>
<th>Arc</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_5$</td>
<td>return $</td>
</tr>
<tr>
<td>$V_4$</td>
<td>return $</td>
</tr>
<tr>
<td>$V_6$</td>
<td>return ResyncSeqLen $\neq P_{res}$</td>
</tr>
<tr>
<td>$V_7$</td>
<td>return $</td>
</tr>
</tbody>
</table>

### Table 6. SRN specification for the Event service with resynchronization - Reward measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Reward rate function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NormalTput</td>
<td>return rate($t_c$)</td>
</tr>
<tr>
<td>SyncTput</td>
<td>return $</td>
</tr>
<tr>
<td>SupplierTput</td>
<td>return $(\text{NormalTput} + \text{ResyncTput})$</td>
</tr>
<tr>
<td>LossRate</td>
<td>return $(\text{PSQlen}=K) # \text{SupplierTput} : 0</td>
</tr>
<tr>
<td>RetentionRate</td>
<td>return LossRate/$\text{SupplierTput}$</td>
</tr>
<tr>
<td>Loss</td>
<td>return $\text{LossRate}/\text{SupplierTput}$</td>
</tr>
</tbody>
</table>
and $\beta = 1.0 \text{ sec}^{-1}$, resulting in a mean arrival rate of 49.9 messages per second.

To study the effect of a bursty input a transient analysis is needed. In Figure 5 we show the estimates in the presence of the Poisson and MMPP sources specified above. For the 2-state MMPP source, we differentiate between the two initial conditions. Initially if the place $P_{\text{mmpp}}$ in Figure 2 contains a token, the supplier starts at rate 39.88 messages per second (non-busy state). If it does not contain a token, it starts at the rate of 100.0 messages per second (busy state). In the plots in Figure 5, these two initial conditions are referred to as MMPP init(1) and MMPP init(0) respectively. Figure 5 shows that when the MMPP source starts in the busy state, there is a big overshoot in the expected PSQ length and loss rate. The expected consumer throughput also shoots up initially and approaches the maximum consumer rate of 70 messages per second. But as the PSQ stabilizes the expected consumer rate falls towards the steady state value. When the MMPP source starts in the non-busy state, there is no overshoot but the expected PSQ length and loss rate are appreciably higher.

When the consumer reaches its maximum consumption rate, the only thing that can be done to reduce the probability of message loss is to increase the maximum PSQ length in order to accommodate the overshoots due to variance in the supplier stream. Figure 6 plots the estimates for the case where the MMPP source starts in the busy state, for different PSQ lengths. It can be seen that for PSQ lengths above 125, the overshoot in queue length and loss rate is very low. Also the consumer throughputs at steady state approaches the mean supplier rate of 49.9 messages per second, indicating that no events are being displaced from the front of the PSQ.

4.1.3 Effect of an isolated burst

Another scenario that is important in real operational systems is the occurrence of an isolated (or rarely occurring) burst from the supplier. For example, consider that the supplier supplies a Poisson stream with a mean arrival rate of 49.9 messages per second. Let us say that there is now a sudden burst of 200 messages that need to be pushed into the Event channel. The problem then is to determine the maximum rate at which the burst can be allowed into the Event channel.

To determine the effect of such an isolated burst, the SRN model is solved in three phases. In the first phase, the SRN is solved as in Section 4.1.1 to obtain the steady state probabilities of all the markings. In the second phase, these steady state probabilities are set as the initial marking probabilities. In our illustration, the rate of transition $t_s$ in the SRN model is changed to reflect four different burst
conditions: (1) burst rate of 66.67 messages per second for 3 seconds, (2) burst rate of 100.0 messages per second for 2 seconds, (3) burst rate of 133.33 messages per second for 1.5 seconds, and (4) burst rate of 200.0 messages per second for 1 second. For each of these burst conditions, a transient analysis is done for the amount of time that the burst lasts. During the last phase of the model solution which is done at the end of the corresponding burst time, the marking probabilities at the end of the second phase are set as the initial marking probabilities and the rate of transition $t_s$ is restored back to 49.9 messages per second. The transient analysis is then continued for a required amount of time.

Figure 7 shows the effect of the isolated burst on the estimates. For all burst conditions except when the expected burst of 200 messages are pushed in over an interval of 3 seconds, the PSQ length shoots towards the maximum of 25. The corresponding loss rate also shoots up during the period of burstiness. Only for the burst case of 66.67 messages per second over a 3 second interval, there is not much deviation from the original conditions. The effect on the expected consumer throughput is interesting. For the burst case of 66.67 messages per second over a 3 second interval, the consumer throughput approaches 66.67 messages per second in order to match the supplier rate. But for the other three scenarios, the consumer throughput initially rapidly approaches its maximum of 70 messages per second. But once the PSQ becomes full, events at the front of the queue start getting thrown out and very few events actually make it to the consumer. Once the burst stops and the supplier returns to the stable rate of 49.9 messages per second, the consumer consumes the messages in the PSQ as fast as it can. Therefore the consumer throughput again shoots up to the maximum value of 70 messages per second. After the excess messages have been consumed, the consumer throughput gradually stabilizes at its steady state value again.

4.2 Event service with the resynchronization scheme

For application domains for which state resynchronization is both possible and sufficient as a means of recovery from event drop occurrences in the PSQ, the rate at which the resynchronization messages can be sent out by the supplier is another parameter to be configured. This rate depends on both the steady state PSQ length, the number of messages in the resynchronization sequence, and the type of input from the supplier under normal conditions (Poisson or MMPP). For our illustration, we shall assume that the events from the supplier under normal conditions constitute a Poisson stream with a rate of 49.9 messages per second, and take the number of messages in the resynchr-
nization sequence to be 10. Figure 8 shows the estimates for this case.

From the plots in Figure 8 it can be seen that appreciable deviations in the estimates can be noticed only after the mean resynchronization rate exceeds 90 messages per second. Mean resynchronization rates higher than the mean consumption rate are possible in this case because the resynchronization sequence length is only 10, whereas the steady state expected PSQ length is less than 2 out of a maximum capacity of 25. Thus the system is able to absorb the rare occurrences of a fast but short resynchronization sequence. This situation will change if the resynchronization sequence is long, resynchronizations occur more frequently (thereby increasing the steady state expected PSQ length for mean resynchronization rates greater than the mean consumption rate), or there is burstiness in the input from the supplier.

5 Applicability of the models to COS Notification service

Recognizing the drawbacks of the Event service, OMG has defined a COS Notification service [17]. Vendor implementations of the service, for example OpenFusion COS Notification service [19] from PrismTech and OrbixNotification [20] from Iona and NEC Systems, are now available. The Notification service extends the Event service by providing: (1) structured event types, (2) selective registration of consumers to specific event types, (3) message content-based filtering, and (4) quality of service (QoS) settings that include persistence and reliability.

The QoS settings of interest in the context of this paper are order policy and discard policy. The Notification service allows event delivery based on FIFO, priority, deadline, or any order. The discard policy QoS can be set to FIFO, LIFO, priority, deadline, or any. The reject new discard policy is also provided in some vendor implementations. The Event service policies discussed in this paper correspond to a Notification service QoS with order policy and discard policy both set to FIFO.

Queuing systems with order policies based on “priority” and “deadline”, and discard policy of “reject new” have been studied in the literature before [1], [8] and can be used to modify the models in this paper to accommodate the different combinations of order and discard policies. Other policies may need new models to be developed depending on the measures required. But the general procedure of configuring the Notification service to minimize the probability of event discard in the presence of bursty input or an isolated burst still holds. The state resynchronization may also be applied to Notification service configurations with QoS settings that do not guarantee against event loss.
6 Concluding remarks

In this paper we studied the most common implementation of the CORBA COS Event service using stochastic reward net (SRN) models. The models can aid in configuring the Event service by determining operating ranges that satisfy design specifications. The models were used to study the effect of Poisson arrivals as well as bursty and correlated arrivals by means of an MMPP event source model. We also demonstrated the use of the models to determine the effect of an isolated burst in input. A model to study the scheme of state resynchronization between suppliers and consumers was also developed.

The Event service specification does not enforce event delivery guarantees and does not specify any QoS settings. The Notification service provides standards based QoS settings to messaging in CORBA applications. Our models can be used to configure a system for minimum event loss as in the case of the Event service, as well as guide the choice of the quality of service as in the case of the Notification service.

References


