Performance Analysis of a Middleware Event Demultiplexing Pattern for Network Services ⋆

Swapna S. Gokhale [a,*]

[a] *Dept. of CSE, Univ. of Connecticut, Storrs, CT 06269.*

Aniruddha S. Gokhale [b]

[b] *Dept. of EECS, Vanderbilt Univ., Nashville, TN 37235.*

Jeffrey G. Gray [c]

[c] *Dept. of CIS, U. of Alabama at Birmingham, Birmingham, AL 35294.*

Abstract

Society is becoming increasingly reliant on the services provided by distributed, performance sensitive software systems. These systems demand multiple simultaneous quality of service (QoS) properties. A key enabler in recent successes in the development of such systems has been middleware, which comprises reusable building blocks. Typically, a large number of configuration options are available for each building block when composing a system end-to-end. The choice of the building blocks and their configuration options have an impact on the performance of the services provided by the systems. Currently, the effect of these choices can be determined only very late in the lifecycle, which can be detrimental to system development costs and schedules. In order to enable the right design choices, a systematic methodology to analyze the performance of these systems at design time is necessary.

Our methodology consists of models of individual building blocks of the middleware and their compositions, which are used to analyze the performance of the composed system. As a first step towards realizing this methodology, this paper introduces a model of the Reactor pattern, which provides important synchronous demultiplexing and dispatching capabilities to network services and applications. The model is based on the Stochastic Reward Net (SRN) modeling paradigm. We illustrate how the model could be used to obtain the performance metrics of a Virtual Private Network (VPN) service provided by a Virtual Router (VR).

Key words: Middleware, Reactor pattern, Performance analysis, Stochastic Reward Nets

1 Introduction

Society is increasingly reliant on the services provided by distributed, performance-sensitive software systems. These systems demand multiple simultaneous quality of service (QoS) properties including predictability, controllability, and adaptability of operating characteristics for applications with respect to such features as time, throughput, accuracy, confidence, security and synchronization. A key enabler in recent successes in the development of such systems is *QoS-enabled middleware* (Schantz and Schmidt (2002)). Middleware comprises software layers that provide platform-independent execution semantics and

⋆ This research is supported in part by US National Science Foundation (NSF) Computer Systems Research (CSR) and Next Generation Software (NGS) programs.

* Corresponding Author.
Email addresses: ssg@engr.uconn.edu (Swapna S. Gokhale), a.gokhale@vanderbilt.edu (Aniruddha S. Gokhale), gray@cis.uab.edu (Jeffrey G. Gray).

reusable services that coordinate how application components are composed and interoperate. The flexibility and configurability offered by middleware is manifested in the large number of reusable software building blocks and their configuration options, which can be used to compose and build large systems end-to-end. These building blocks embody good design practices called patterns (Gamma et al. (1995); Schmidt et al. (2000)). The choice of the patterns and their configuration options is driven by the context of the application. These choices have a profound impact on the overall performance of the provided service.

Current *ad hoc* techniques based on manually choosing the right set of building blocks and their configuration options are error-prone and may adversely impact performance, system costs and schedules, since most errors are caught very late in the lifecycle of the system development. It is desirable to have the ability to analyze the performance of individual building blocks and the composed system much earlier in the system lifecycle, thereby significantly lowering system testing costs as well as improving the correctness of the final developed system.

To address the challenge of system performance evaluation in the design phase, a systematic performance analysis methodology is necessary. This methodology would consist of the performance models of the individual building blocks and their compositions. The performance models may be based upon well-known analytical/numerical modeling paradigms (Puliafito et al. (1997); Choi et al. (1994); Horton et al. (1998)) and simulation techniques (The VINT Project (1996)). As a first step towards the development of such a methodology, this paper presents a model of the *Reactor pattern* (Gamma et al. (1995); Schmidt et al. (2000)), which provides important synchronous demultiplexing

and dispatching capabilities to network services and applications. The model is based on the Stochastic Reward Net (SRN) modeling paradigm (Puliafito et al. (1997)). We illustrate how the model can be used to obtain estimates of the performance metrics of a Virtual Private Network (VPN) service provided by a Virtual Router (VR) (Knight et al. (2004)).

Paper organization: The paper is organized as follows: Section 2 presents the performance model of the reactor pattern. Section 3 illustrates how the SRN model of the reactor pattern is used to obtain the performance metrics of a VPN service provided by a VR. Section 4 offers concluding remarks and directions for future research.

2 Performance Model of the Reactor Pattern

In this section, we first provide an overview of the reactor pattern followed by the SRN model of the reactor pattern. We then describe how the different performance metrics can be obtained from the SRN model.

2.1 Reactor Pattern in Middleware Implementations

Figure 1 depicts a typical event demultiplexing and dispatching mechanism documented in the reactor pattern. The application registers an event handler with the event demultiplexer and delegates to it the responsibility of listening for incoming events. On the occurrence of an event, the demultiplexer dispatches the event by making a callback to its associated application-supplied event handler. This is the idea behind the reactor pattern, which provides synchronous event demultiplexing and dispatching capabilities.

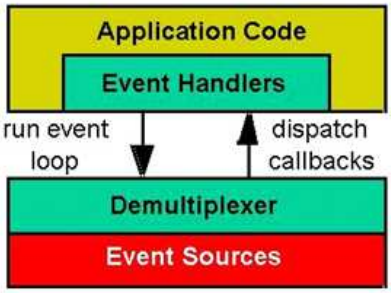

Fig. 1. **Event Demultiplexers in Middleware**

Figure 2 illustrates the reactor pattern dynamics. We categorize these dynamics into two phases described below.

(1) **Registration phase:** In this phase all the event handlers register with the reactor associating themselves with a particular event type they are interested in. Event types usually supported by a reactor are input, output, timeout and exceptions. The reactor will maintain a set of handles corresponding to each handler registered with it.

(2) **Snapshot phase:** Once the event handlers have completed their registration, the main thread of control is passed to the reactor, which in turn listens for events to occur. A snapshot is then an instance in time wherein a reactor determines all the event handles that are enabled at that instant. For all the event handles that are enabled in a given snapshot, the reactor proceeds to service each event by invoking the associated event handler. There could be different strategies to handle these events. For example, a reactor could handle all the enabled events sequentially in a single thread or could hand it over to worker threads in a thread pool. After all the events are processed, the reactor proceeds to take the next snapshot of the system.

The reactor pattern could be implemented in many different ways depending

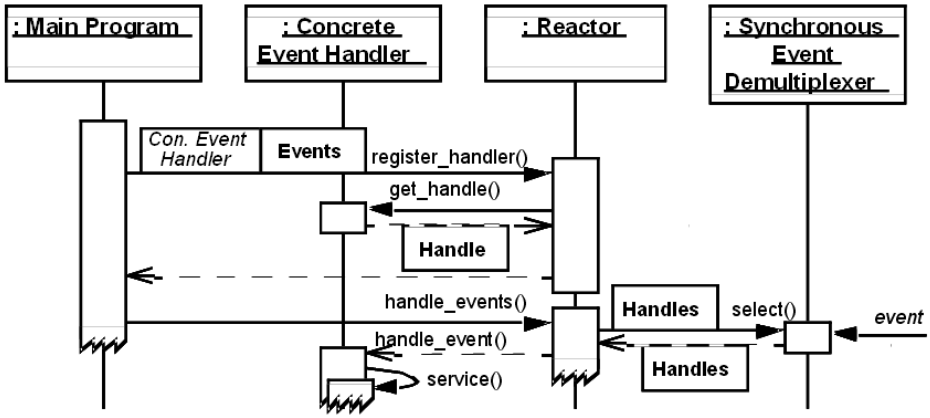

Fig. 2. **Reactor Pattern Dynamics**

on the event demultiplexing capabilities provided by the operating systems and the concurrency requirements of the applications. For example, the demultiplexing capabilities of a reactor could be based on the `select ()` or `poll ()` system calls provided by POSIX-compliant operating systems, or `WaitForMultipleObject ()` as found in the different flavors of Win32 operating systems. Moreover, the handling of the event in the event handler could be managed by the same thread of control that was listening for events leading to a single-threaded reactor implementation. Alternatively, the event could be delegated to a pool of threads to handle the events leading to a thread-pool reactor.

2.2 Characteristics of the Reactor Pattern

To simplify the description of our methodology, we consider a single-threaded, *select*-based implementation of the reactor pattern with the following characteristics:

- The reactor receives two types of input events with one event handler for each type of event registered with the reactor.
- Each event type has a separate queue, which holds the incoming events of that type. The buffer capacity for the queue of type #1 events is denoted N_1 and of type #2 events is denoted N_2.
- Event arrivals for both types of events follow a Poisson distribution with rates λ_1 and λ_2, while the service times of the events are exponentially distributed with rates μ_1 and μ_2.
- In a snapshot, an event of type #1 is serviced with a higher priority over an event of type #2. In other words, when event handles corresponding to both event types are enabled in a snapshot, the event handle corresponding to type #1 is serviced with a priority that is higher than the event handle of type #2.

2.3 Desired Performance Metrics

The following performance metrics are of interest for each one of the event types in the reactor pattern described in Section 2.2:

- **Expected throughput** – which provides an estimate of the number of events that can be processed by the single threaded event demultiplexing framework. These estimates are important for many applications, such as telecommunications call processing.
- **Expected queue length** – which provides an estimate of the queuing for each of the event handler queues. These estimates are important to develop appropriate scheduling policies for applications with real-time requirements.

- **Expected total number of events** – which provides an estimate of the total number of events in a system. These estimates are also tied to scheduling decisions. In addition, these estimates will determine the right levels of resource provisioning required to sustain the system demands.
- **Probability of event loss** – which indicates how many events will have to be discarded due to lack of buffer space. These estimates are important particularly for safety-critical systems, which cannot afford to lose events. These also provide an estimate on the desired levels of resource provisioning.
- **Expected response time** – which indicates the time taken to service an event. These estimates are important for real-time services.

2.4 SRN Model

In this section we present the Stochastic Reward Net (SRN) model of the reactor pattern. A Stochastic Reward Net (SRN) substantially extends the modeling power of Generalized Stochastic Petri Nets (GSPNs) (Puliafito et al. (1997)), which are an extension of Petri nets (Peterson (1981)). A SRN is a modeling technique that is concise in its specification and closer to a designer's intuition about what a model should look like. Stochastic reward nets (SRNs) can be solved using Stochastic Petri Net Package (SPNP) Hirel et al. (2000) and they have been extensively used for performance, reliability and performability analysis of a variety of systems (Ramani et al. (2000); Ibe et al. (1989); Ibe and Trivedi (1990); Sun et al. (1999); Ibe and Trivedi (1991); Muppala et al. (1994)). The work closest to the proposed research is reported by Ramani *et al.* (Ramani et al. (2000)), where SRNs are used for the performance analysis of the CORBA event service. A detailed overview of SRNs can be

obtained from (Puliafito et al. (1997)).

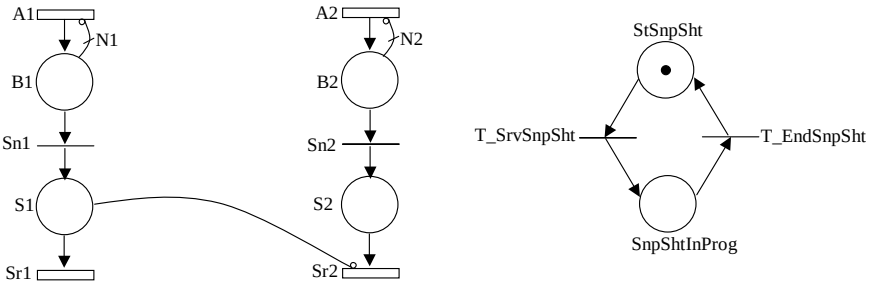

Fig. 3. **SRN model for the reactor pattern**

Description of the net: Figure 3 shows the SRN model for the reactor pattern with the characteristics described in Section 2.2. The net on the left-hand side models the arrival, queuing and service of the two types of events. Transitions $A1$ and $A2$ represent the arrival of the events of type #1 and #2, respectively. Places $B1$ and $B2$ represent the queue for the two types of events. Transitions $Sn1$ and $Sn2$ are immediate transitions that are enabled when a snapshot is taken. Places $S1$ and $S2$ represent the enabled handles of the two types of events, whereas transitions $Sr1$ and $Sr2$ represent the execution of the enabled event handlers of the two types of events. An inhibitor arc from place $B1$ to transition $A1$ with multiplicity $N1$ prevents the firing of transition $A1$ when there are $N1$ tokens in place $B1$. The presence of $N1$ tokens in place $B1$ indicates that the buffer space to hold the incoming input events of the first type is full, and no additional incoming events can be accepted. The inhibitor arc from place $B2$ to transition $A2$ achieves the same purpose for type #2 events. The inhibitor arc from place $S1$ to transition $Sr2$ prevents the firing of transition $Sr2$ when there is a token in place $S1$. This models the prioritized service for an event of type #1 over event of type #2 in a given snapshot.

The net on the right of Figure 3 models the process of taking successive snapshots and prioritized service of the event handle corresponding to type #1 events in each snapshot. Transition $Sn1$ is enabled when there is a token in place $StSnpSht$, at least one token in place $B1$, and no tokens in place $S1$. Similarly, transition $Sn2$ is enabled when there is a token in place $StSnpSht$, at least one token in place $B2$, and no tokens in place $S2$. Transition $T\_SrvSnpSht$ is enabled when there is a token in either one of the places $S1$ and $S2$, and the firing of this transition deposits a token in place $SnpShtInProg$.

The presence of a token in the place $SnpShtInProg$ indicates that the event handles that were enabled in the current snapshot are being serviced. After these event handles complete execution, the current snapshot is complete and it is time to take another snapshot. This is accomplished by enabling the transition $T\_EndSnpSht$. Transition $T\_EndSnpSht$ is enabled when there are no tokens in both places $S1$ and $S2$. Firing of the transition $T\_EndSnpSht$ deposits a token in place $StSnpSht$, indicating that the service of the enabled handles in the present snapshot is complete, which marks the initiation of the next snapshot. Table 1 summarizes the enabling/guard functions for the transitions in the net.

Dynamic evolution of the net: We now describe how the process of taking a single snapshot is modeled by the SRN model presented in Figure 3. We consider a scenario where there is one token in each one of the places $B1$ and $B2$, and there is a token in the place $StSnpSht$. Also, there are no tokens in places $S1$ and $S2$. In this scenario, transitions $Sn1$ and $Sn2$ are enabled. Both of these transitions are assigned the same priority, and any one of these

Transition	Guard function
$Sn1$	$((\#StSnpShot == 1)\&\&(\#B1 >= 1)\&\&(\#S1 == 0))?1 : 0$
$Sn2$	$((\#StSnpShot == 1)\&\&(\#B2 >= 1)\&\&(\#S2 == 0))?1 : 0$
$T\_SrvSnpSht$	$((\#S1 == 1)\|\|(\#S2 == 1))?1 : 0$
$T\_EndSnpSht$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$

[1] Enabling/Guard Functions

transitions can fire first. Also, since these transitions are immediate, their firing occurs instantaneously. Without loss of generality, it can be assumed that transition $Sn1$ fires before $Sn2$, which deposits a token in place $S1$.

When a token is deposited in place $S1$, transition $T\_SrvSnpSht$ is enabled. In addition, transition $Sn2$ is already enabled. If transition $T\_SrvSnpSht$ were to fire before transition $Sn2$, it would disable transition $Sn2$, and prevent the handle corresponding to the second event type from being enabled. In order to prevent transition $T\_SrvSnpSht$ from firing before transition $Sn2$, transition $T\_SrvSnpSht$ is assigned a lower priority than transition $Sn2$. Because transitions $Sn1$ and $Sn2$ have the same priority, this also implies that the transition $T\_SrvSnpSht$ has a lower priority than transition $Sn1$. This ensures that in a given snapshot, event handles corresponding to each event type are enabled when there is at least one event in the queue.

After both event handles are enabled, transition $T\_SrvSnpSht$ fires and deposits a token in place $SnpShtInProg$. The presence of a token in the place $SnpShtInProg$ indicates that the event handles that were enabled in the current snapshot are being serviced. The event handle corresponding to type #1

event is serviced first, which causes transition $Sr1$ to fire and the removal of the token from place $S1$. Subsequently, transition $Sr2$ fires and the event handle corresponding to the event of type #2 is serviced. This causes the removal of the token from place $S2$. After both events are serviced and there are no tokens in places $S1$ and $S2$, transition $T\_EndSnpSht$ fires, which marks the end of the present snapshot and the beginning of the next one.

Assignment of reward rates: The performance measures described in Section 2.3 can be computed by assigning reward rates at the net level as summarized in Table 2. The throughputs of events of type #1 and #2 denoted T_1 and T_2 respectively are given by the rate at which transitions $Sr1$ and $Sr2$ fire. The queue lengths of the events denoted Q_1 and Q_2 are given by the number of tokens in places $B1$ and $B2$ respectively. The total number of events of type #1 denoted E_1 is given by the sum of the number of tokens in places $B1$ and $S1$. Similarly, the total number of events of type #2 denoted E_2 is given by the sum of the number of tokens in places $B2$ and $S2$. The loss probability of type #1 events denoted L_1 is given by the probability of $N1$ tokens in place $B1$. Similarly, the loss probability of type #2 events denoted L_2 is given by the probability of $N2$ tokens in place $B2$.

We obtain the response times of the events denoted R_1 and R_2 using the tagged customer approach (Melamed and Yadin (1984)). In the tagged customer approach, an arriving event is tagged and its trajectory through the system is followed from entry to exit. The response time of the tagged event is then determined conditional to the state in which the system lies when the event arrives. The unconditional response time can be obtained as the weighted sum of the conditional response times, with the weights given by the steady state

Performance metric	Reward rate
T_1	return rate($Sr1$)
T_2	return rate($Sr2$)
Q_1	return ($\#B1$)
Q_2	return ($\#B2$)
L_1	return ($\#B1 == N1?1 : 0$)
L_2	return ($\#B2 == N2?1 : 0$)
E_1	return($\#B1 + \#S1$)
E_2	return($\#B2 + \#S2$)
R_1	return($\#B1 < N1?$ $(1/\mu_1 * (\#S1 + \#B1 + 1) + 1/\mu_2 * (\#S2 + \#B1)) : 0$)
R_2	return($\#B2 < N2?$ $(1/\mu_2 * (\#S2 + \#B2 + 1) + 1/\mu_1 * (\#S1 + \#B2 + 1)) : 0$)

[2] Reward assignments to obtain performance measures

probabilities of being in each one of the states. Typically, the response time of an event consists of two pieces; namely, the time taken to service the event hereafter referred to as the "service time," and the time that the event must wait in the system before its service commences, hereafter referred to as "waiting time". In our case, the average service time of an incoming type #1 and

type #2 event is given by $1/\mu_1$ and $1/\mu_2$, irrespective of the state in which the system lies when the event arrives. The waiting time, however, will depend on the system state. Next, we discuss how the conditional waiting time of each event type is determined.

The conditional waiting time for a tagged event of type #1 will depend on the state of the system, where the state is given by the number of tokens or markings of places $S1$, $S2$, $B1$ and $B2$. Of these four places, the markings of the places $S1$ and $S2$ determine the progress of the current snapshot, whereas, the markings of places $B1$ and $B2$ determine the state of the queue. The time taken to complete the current snapshot is given by the sum of two terms, the first term is the product of the number of tokens in place $S1$ and $1/\mu_1$, and the second term is the product of the number of tokens in place $S2$ and $1/\mu_2$. Even if there are no additional events in the queues, the current snapshot must be completed before the service of an incoming event of type #1 can begin. Hence, the time taken to complete the current snapshot contributes to the waiting time of the incoming or tagged type #1 event. In order to obtain the entire waiting time of a tagged type #1 event, the contribution of the queued events of type #1 and type #2 needs to be determined.

Let n_1 be the number of events of type #1 in the queue, and n_2 be the number of events of type #2 in the queue, when the tagged event of type #1 arrives. This implies that after n_1 snapshots the tagged event will be serviced. The following three possibilities arise between the relative values of n_1 and n_2. If $n_1 < n_2$, then only n_1 of the type #2 events need to be serviced before the service of the tagged type #1 event can commence, and hence the waiting time is given by $n_1(1/\mu_1 + 1/\mu_2)$. If $n_1 = n_2$, then n_1 events of type #1 and type #2 need to be serviced before the service of the incoming type #1 event can

commence, and hence the waiting time is given by $n_1(1/\mu_1+1/\mu_2)$. If $n_1 > n_2$, then in the optimistic case, n_1 events of type #1 and n_2 events of type #2 need to be serviced before the service of the tagged event can commence. The optimistic case assumes that no additional events of type #2 arrive in the first n_1 snapshots. In the pessimistic case, however, $n_1 - n_2$ events of type #2 will arrive while the first n_2 events are being serviced. Thus, in the optimistic case, the waiting time will be $n_1/\mu_1 + n_2/\mu_2$, and in the pessimistic case, the waiting time will be $n_1(1/\mu_1 + 1/\mu_2)$. We consider the pessimistic case since that provides an upper bound on the response time. The pessimistic contribution of the queued events to the waiting time is thus given by the product of the number of tokens in place $B1$ and the sum of the reciprocals of μ_1 and μ_2. Thus, the overall response time of the tagged event will be given by the sum of two terms, the first term is $1/\mu_1$ times the sum of the tokens in places $S1$, $B1$ and 1, and the second term is given by the product of $1/\mu_2$ and the sum of the number of tokens in place $S2$ and $B1$. The contribution of the queued events to the waiting time of the tagged event of type #2 can also be determined using similar reasoning, with an additional consideration given to the prioritized service provided to event of type #1 over an event of type #2 in each snapshot. The reward rates to obtain the performance measures of the events of type #1 and type #2 are summarized in Table 2.

2.5 Model Variations

In the model of the reactor pattern described above, the arrival and the service time distributions are assumed to be exponential for the sake of illustration. For certain types of applications, this assumption may not hold. For example,

for safety-critical applications, events may occur at regular intervals, in which case the arrival process is deterministic. In addition to the deterministic distribution, the arrival and service times may also follow any other non-exponential or general distribution. There are two ways to consider non-exponential distributions in the SRN model. In the first method, a non-exponential distribution can be approximated using a phase-type approximation (Puliafito et al. (1997)), and the resulting SRN model can then be solved using SPNP. In the second method, the model can be simulated using discrete-event simulation incorporated in SPNP (Hirel et al. (2000)).

3 Case Study: VPN Service using Virtual Router

In this section we describe how the SRN model of the reactor pattern presented in Section 2.4 is used to estimate the response time of a Virtual Private Network (VPN) service provided by a Virtual Router (VR).

Figure 4 illustrates the architecture of a provider-provisioned virtual private network (PPVPN) (Nagarajan (2004)) using a VR. A VR is a software/-hardware component that is part of a physical router called the provider edge (PE) router. A VR contains the mechanisms to provide highly scalable, differentiated levels of services in VPN architectures. Multiple VRs can reside on a PE device. VRs can be arranged in a hierarchical fashion within a single PE as shown in Figure 4. Moreover, an entity acting as a service provider for an end customer might itself be a customer of a larger service provider. VRs may also use different backbones to improve reliability or to provide differentiated levels of service to customers.

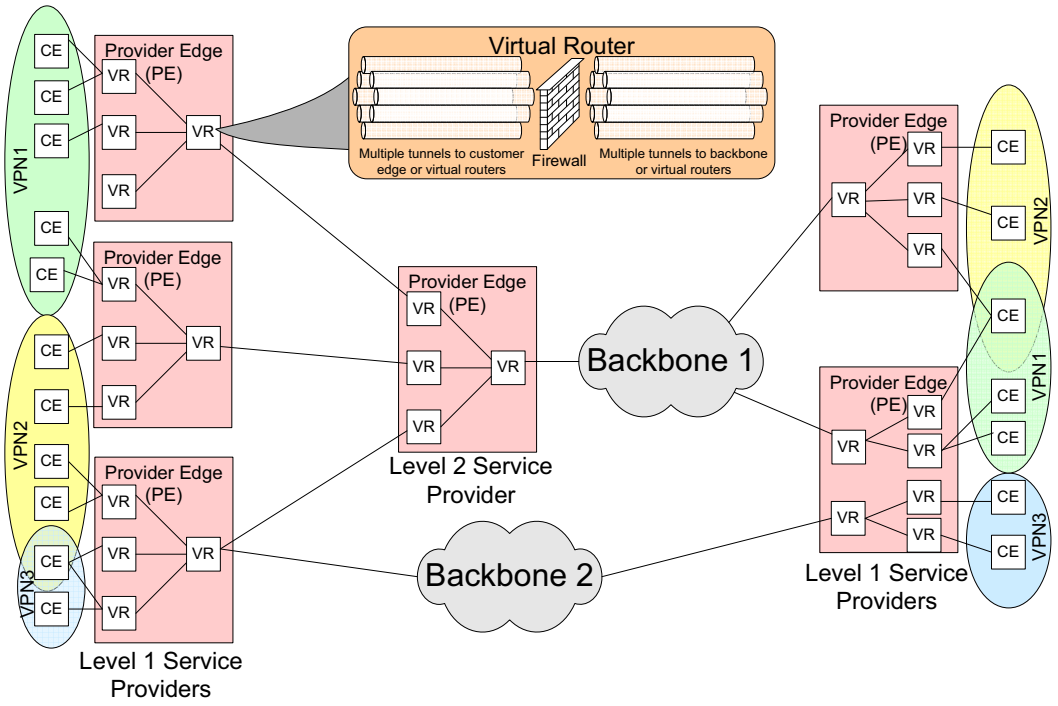

Fig. 4. **VPN Architecture using Virtual Routers**

Customer edge (CE) devices wishing to join a VPN connect to a VR on the PE device. A VR can multiplex several distinct CEs belonging to the same VPN session. A VR may use tunneling mechanisms to use multiple routing protocols and link layer protocols, such as IPSec, GRE, and IP-in-IP, to connect with the CEs. A totally different set of protocols and tunneling mechanisms could be used for inter-VR or VR-backbone communication. These tunneling mechanisms can also be the basis for differentiated levels of service as well as to provide improved reliability. A VR also comprises firewall capabilities.

We consider a scenario where a VR is used to provide VPN services to two organizations, with each organization having a customer edge (CE) router connected to the VR. The employees of each organization issue VPN set up and tear down service requests to the VR via CEs. Also, the VR offers a dif-

ferentiated level of service, with organization #1 receiving prioritized service over organization #2. From the point of view of the employees of the organizations, it is necessary that the service requests be handled in a reasonable amount of time. Also, the probability of denying the service requests should be minimal. From the point of view of the VPN service provider, the rate at which the service requests are processed or the throughput is important. The throughput will determine determine the revenue collected by the provider.

In order to implement the VPN service, a reactor pattern with the characteristics described in Section 2.2 can be used to demultiplex the events. The SRN model of the reactor pattern can thus be used for the performance analysis of the VPN service provided by the VR. In order to use the SRN model, we designate the requests originating from organization #1 as events of type #1 and requests originating from organization #2 as events of type #2.

In the early stages of application development lifecycle, it is necessary to analyze the impact of design choices and configurations on the performance metrics. Also, in these stages, it is rarely the case that the values of the input parameters can be estimated with certainty, which makes it imperative to analyze the sensitivity of the performance metrics to the variations in the input parameters. Sensitivity analysis will enable the provider to determine the regions of operation during which service performance is acceptable. In the subsequent subsections we demonstrate how the SRN model can be used to explore alternative design configurations, as well as to facilitate sensitivity analysis.

3.1 Impact of Buffer Space

The buffer space available to hold the incoming events of each type is a configuration parameter of the reactor pattern and the VPN service that uses the pattern. This choice will have a direct impact on all the performance metrics. Most importantly, from the employees' perspective, the buffer space will influence the probability of denying the service requests.

We analyze the impact of the buffer capacities on the performance measures. The values of the remaining parameters (except for the buffer capacities) are reported in Table 3. We consider two values of buffer capacities N_1 and N_2. In the first case, the buffer capacity is set to 1 for both types of events, whereas in the second case the buffer capacity of both types of events is set to 5. The performance metrics for both these cases are summarized in Table 4. Because the values of the parameters of the service requests from organization #1 (λ_1, μ_1 and N_1) are the same as the values of the parameters for the service requests from organization #2 (λ_2, μ_2, and N_2), the throughputs, queue lengths, and the loss probabilities are the same for both types of service requests for each one of the buffer capacities as indicated in Table 4. It can be observed that the loss probabilities are significantly higher when the buffer capacity is 1 compared to the case when the buffer space is 5. Also, due to the higher loss probability, the throughput is slightly lower when the buffer capacity is 1 than when the buffer capacity is 5.

For both buffer capacities, the total number of service requests from organization #2, denoted E_2, is slightly higher than the total number of requests from organization #1, denoted E_1. Because the requests from organization

#1 are provided prioritized service over the requests from organization #2, on an average it takes longer to service a request from organization #2 than it takes to service a request from organization #1. This results in a higher total number of requests from organization #2 than the total number of requests from organization #1. This also results in a higher response time for service requests from organization #2 as compared with the service requests from organization #1, as shown in Table 4.

Event type	Arrival rate	Service rate
#1	$\lambda_1 = 0.400$/sec.	$\mu_1 = 2.000$/sec.
#2	$\lambda_2 = 0.400$/sec.	$\mu_2 = 2.000$/sec.

[3] Parameter values

3.2 Impact of Service Rates

Another design parameter that will impact the service performance is the service time of the event handlers. Obviously, a faster rate of service will improve performance, but from the provider's perspective, increasing the service rate may require higher levels of resource provisioning thereby increasing the cost. Therefore, it is necessary to determine what service rate would be "good enough" to provide acceptable service performance. This can be accomplished by analyzing the sensitivity of the performance metrics with respect to the service rates, namely, μ_1 and μ_2.

The results reported in Section 3.1 indicate that the loss probability is significantly lower when the buffer space is 5 as compared to the loss probability

Performance measure	Buffer space	
	$N_1 = 1$, $N_2 = 1$	$N_1 = 5$, $N_2 = 5$
T_1	0.37/sec.	0.39/sec.
T_2	0.37/sec.	0.39/sec.
Q_1	0.06	0.12
Q_2	0.06	0.12
E_1	0.25	0.31
E_2	0.27	0.34
L_1	0.06	0.00024
L_2	0.06	0.00024
R_1	0.63 sec.	0.83 sec.
R_2	1.10 sec.	1.33 sec.

[4]Impact of buffer capacity on performance measures

when the buffer space is 1. As a result, for the purpose of sensitivity analysis with respect to μ_1 and μ_2, we set the buffer capacity for both service requests to 5. We vary the service rates μ_1 and μ_2 in the range of 0.4/sec. to 2.0/sec. roughly in steps of 0.025, one at a time, to obtain the expected performance measures by solving the SRN model shown in Figure 3. The remaining parameters are held at the values reported in Table 3.

The plots in the left column of Figure 5 show the variation of the perfor-

mance measures with respect to μ_1, while the plots in the right column of Figure 5 show the variation with respect to μ_2. The topmost plot in the left column shows the variation of throughputs of service requests from organizations #1 and #2 with respect to μ_1. The figure indicates that as μ_1 decreases, the throughputs of requests from both organizations decrease; however, the magnitude of decrease in the throughput of requests from organization #1 is higher than the magnitude of decrease in the throughput of the requests from organization #2. The second and the fourth plots in the same column show the variation of the total number of requests and the queue length for requests from both organizations as a function of μ_1. It can be observed that both the total number of requests and the queue length increase as μ_1 decreases, and the magnitude of increase for service requests from organization #1 is higher than the magnitude of increase for requests from organization #2. Similar trends can be observed in the variation of the loss probabilities and response times in the third and the fifth plots.

It can be noted that as μ_1 decreases, the performance measures degrade (loss probability, queue length, total number of events and response time increases, and throughput decreases). Also, as μ_1 decreases below 0.8/sec., the performance metrics deteriorate rapidly. With decreasing μ_1, the time taken to service a request from organization #1 increases due to the additional time taken to complete each snapshot. As a result, indirectly, this also increases the time taken to service a request from organization #2 in each snapshot. Thus, the performance of requests from both organizations are adversely impacted, even though only the service time for the requests from only one of the organizations increases. The variation of the performance metrics with respect to μ_2 shows the same trend, with the role of event types reversed.

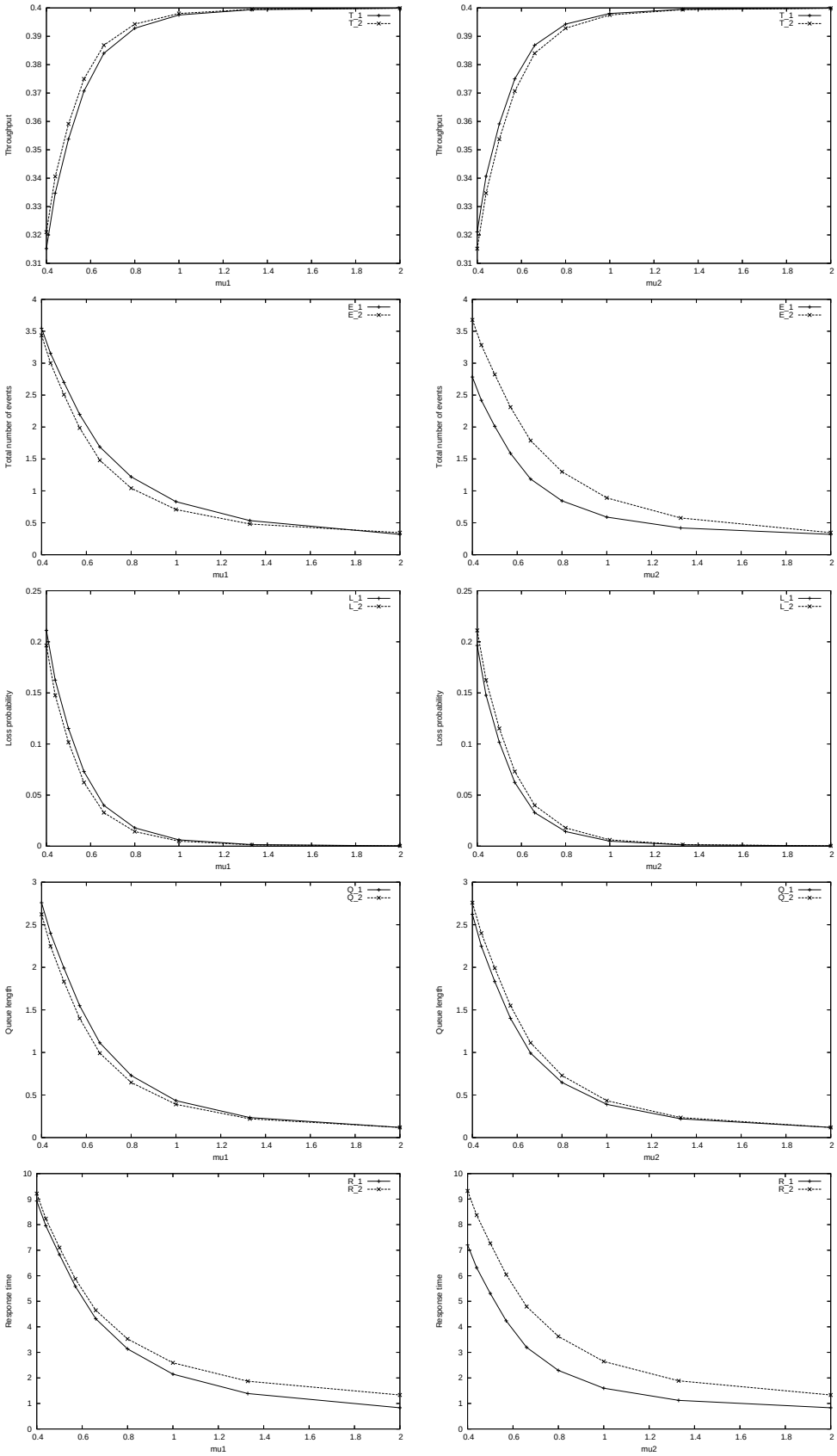

Fig. 5. Sensitivity of performance metrics to service rates μ_1 and μ_2

3.3 Impact of Request Arrival Rates

The arrival rates of service requests, namely, λ_1 and λ_2, constitute the input parameters. In this section we analyze the the sensitivity of the performance metrics to these input parameters.

Based on the results and analysis described in Section 3.1 and Section 3.2, we set N_1 and N_2 to 5, and we set μ_1 and μ_2 to 2.00/sec for sensitivity analysis with respect to λ_1 and λ_2. The parameters λ_1 and λ_2 were varied one at a time in the range of 0.4/sec to 2.0/sec. roughly in steps of 0.025 to obtain the expected performance metrics by solving the SRN model shown in Figure 3.

Figure 6 shows the performance measures as a function of λ_1 and λ_2. The plots in the left column show the variation of the performance metrics with respect to λ_1, whereas the plots in the right column show the variation of the performance metrics with respect to λ_2. Referring to the topmost figure in the left column, it can be observed that initially, the throughput of service requests from organization #1 is nearly the same as the arrival rate of the requests, indicating that the requests are serviced at the same rate at which they arrive. However, as λ_1 increases, the throughput starts lagging the arrival rate, which indicates that the service rate is not sufficiently high to process the requests at the rate at which they arrive. This may cause the queue for requests from organization #1 to operate at full capacity for an extended period of time, which results in a rejection of the incoming requests from organization #1.

It can be observed that a decrease in the rate at which the throughput increases is accompanied by an increase in the loss probability of the requests (as shown in the third plot in the left column) and an increase in the queue length

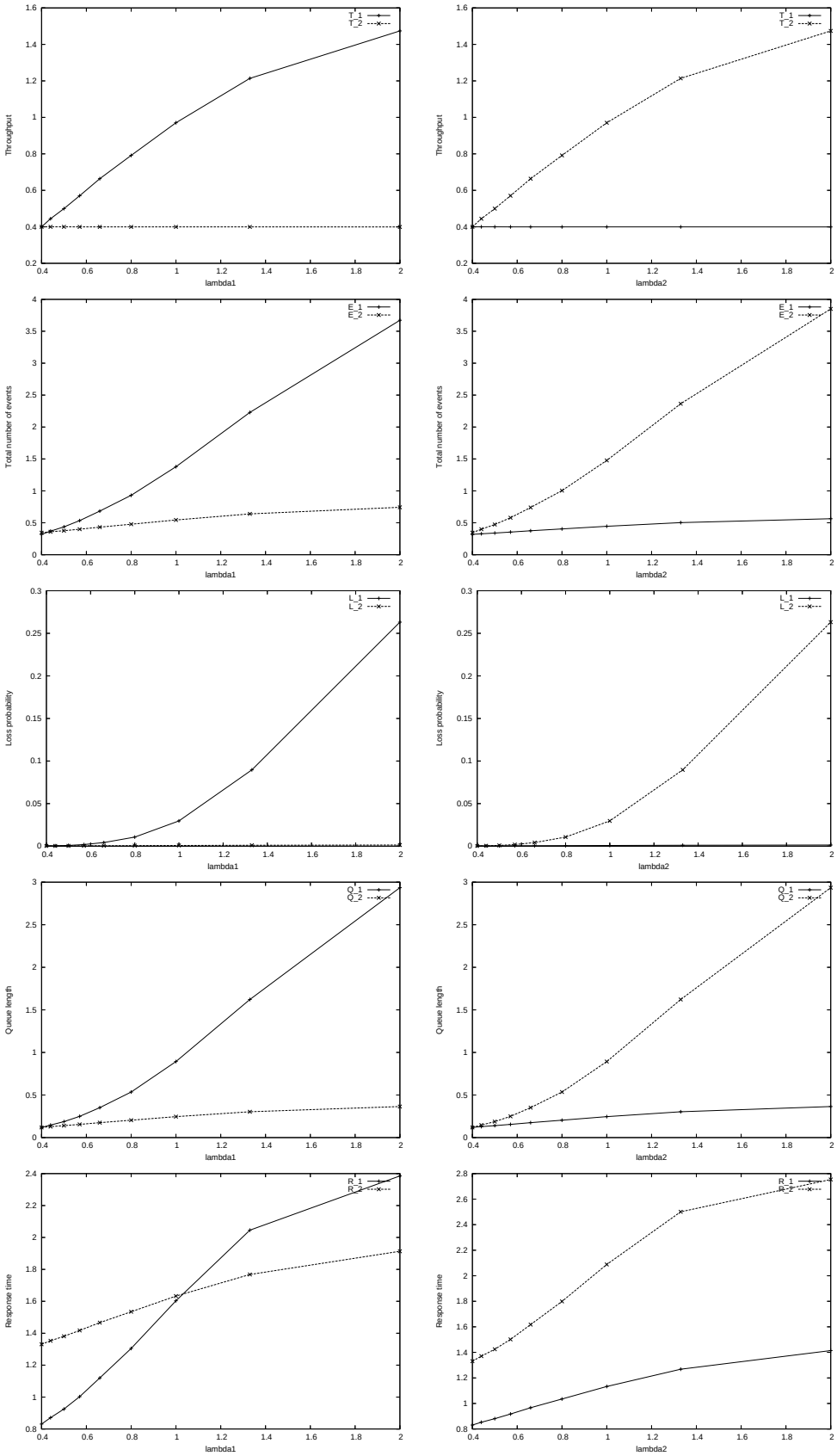

Fig. 6. Sensitivity of performance metrics to arrival rates λ_1 and λ_2

(as shown in the fourth plot in the left column) in Figure 6. The left plot in the second row represents the total number of service requests from each organization as a function of λ_1. The plot indicates that the total number of requests from organization #2 in the system increases as λ_1 increases. When λ_1 increases, the probability of having to service a request from organization #1 prior to servicing a request from organization #2 in a given snapshot increases. Because requests from organization #1 have priority over requests from organization #2, requests from organization #2 tend to reside longer in the system as λ_1 increases. Thus, although the throughput of requests from organization #2 is unchanged with respect to λ_1, the response time of the requests from organization #2 increases as shown in the fifth plot in the figure. At approximately $\lambda_1 = 1.0$, the expected response times for both types of events is close. However, as λ_1 increases beyond 1.0/sec, the expected response time of type #1 events is higher than the expected response time of type #2 events. Thus, effectively, requests from organization #2 are receiving better service than requests from organization #1.

The plots in the right column of Figure 6 indicate that similar trends can be observed for all the performance measures except response time with respect to λ_2, except that the roles of the two types of requests are reversed. The response time, however, follows a different trend. The response time of both the event types increases, however, the response time for the events of type #1 is always lower than the response time for the events of type #2, for the entire range of variation of λ_2.

4 Conclusions and future research

In this paper we presented a performance model of the reactor pattern which offers the important synchronous demultiplexing and dispatching capabilities in middleware. The model was based on the Stochastic Reward Net (SRN) modeling paradigm. We illustrated how the performance model is used to obtain estimates of the performance metrics of a VPN service provided by a Virtual Router (VR). Our future research consists of empirically validating the performance estimates obtained from the performance model. Developing and validating the performance models of other middleware building blocks and the composition of these building blocks is also a topic of future research.

References

Choi, H., Kulkarni, V., Trivedi, K. S., 1994. Markov Regenerative Stochastic Petri Net. Performance Evaluation 20 (1–3), 337–357.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Hirel, C., Tuffin, B., Trivedi, K. S., 2000. SPNP: Stochastic Petri Nets. Version 6.0. Lecture Notes in Computer Science 1786.

Horton, G., Kulkarni, V., Nicol, D., Trivedi, K. S., 1998. Fluid stochastic Petri nets: Theory, application and solution techniques. Journal of Operations Research 405.

Ibe, O., Sathaye, A., Howe, R., Trivedi, K. S., 1989. Stochastic Petri net modeling of VAXCluster availability. In: Proc. of Third International Workshop on Petri Nets and Performance Models. Kyoto, Japan, pp. 142–151.

Ibe, O., Trivedi, K. S., December 1990. Stochastic Petri net models of polling systems. IEEE Journal on Selected Areas in Communications 8 (9), 1649–1657.

Ibe, O., Trivedi, K. S., 1991. Stochastic Petri net analysis of finite–population queueing systems. Queueing Systems: Theory and Applications 8 (2), 111–128.

Knight, P., Ould-Brahim, H., Gleeson, B., Apr. 2004. Network based VPN Architecture using Virtual Routers. IETF Network Working Group Internet Draft, draft-ietf-l3vpn-vpn-vr-02.txt, 1–21.

Melamed, B., Yadin, M., July-August 1984. Randomization procedures in the computation of cumulative-timed distributions over discrete-state markov process. Operations Research 32 (4), 926–944.

Muppala, J., Ciardo, G., Trivedi, K. S., July 1994. Stochastic reward nets for reliability prediction. Communications in Reliability, Maintainability and Serviceability: An International Journal Published by SAE Internationa 1 (2), 9–20.

Nagarajan, A., Jun. 2004. Generic Requirements for Provider Provisioned Virtual Private Network (PPVPN). In: Nagarajan, A. (Ed.), IETF Network Working Group Request for Comments, RFC 3809. IETF, pp. 1–25.

Peterson, J. L., 1981. Petri Net Theory and the Modeling of Systems. Prentice-Hall.

Puliafito, A., Telek, M., Trivedi, K. S., September 1997. The evolution of stochastic Petri nets. In: Proc. of World Congress on Systems Simulation. Singapore, pp. 3–15.

Ramani, S., Trivedi, K. S., Dasarathy, B., October 2000. Performance analysis of the CORBA event service using stochastic reward nets. In: Proc. of the 19th IEEE Symposium on Reliable Distributed Systems. pp. 238–247.

Schantz, R. E., Schmidt, D. C., 2002. Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications. In: Marciniak, J., Telecki, G. (Eds.), Encyclopedia of Software Engineering. Wiley & Sons, New York.

Schmidt, D. C., Stal, M., Rohnert, H., Buschmann, F., 2000. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2. Wiley & Sons, New York.

Sun, H., Zang, X., Trivedi, K. S., June 1999. A stochastic reward net model for performance analysis of prioritized DQDB MAN. Computer Communications, Elsevier Science 22 (9), 858–870.

The VINT Project, 1996. Network Simulator - NS-2. `http://www.isi.edu/nsnam/ns`.