The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software

C.M. Woodside J.E. Neilson D.C. Petriu
S. Majumdar

Telecommunications Research Institute of Ontario
Department of Systems and Computer Engineering, and
School of Computer Science
Carleton University
Ottawa K1S 5B6, Canada

February 18, 1998

Abstract

Distributed or parallel software with synchronous communication via rendezvous is found in client-server systems and in proposed Open Distributed Systems, in implementation environments such as Ada, V, Remote Procedure Call systems, in Transputer systems, and in specification techniques such as CSP, CCS and LOTOS. The delays induced by rendezvous can cause serious performance problems, which are not easy to estimate using conventional models which focus on hardware contention, or on a restricted view of the parallelism which ignores implementation constraints. Stochastic Rendezvous Networks are queueing networks of a new type which have been proposed as a modelling framework for these systems. They incorporate the two key phenomena of *included service* and a *second phase of service*. This paper extends the model to also incorporate different services or *entries* associated with each task. Approximations to arrival-instant probabilities are employed with a Mean-Value Analysis framework, to give approximate performance estimates. The method has been applied to moderately large industrial software systems.

Keywords: client-server, remote procedure call, software performance, distributed software, rendezvous delays, multiple entries.

1 Introduction

A style of synchronous inter-task communication called *rendezvous* has been introduced into software specification techniques such as CSP [2], CCS [24], and LOTOS [25], implementation languages such as Ada [9], distributed operating systems like V [10], and the many distributed systems employing the remote procedure call mechanism. In transputers, rendezvous are implemented in hardware. The wide class of systems using a "client-server" architecture, as in the ISO plan for Open Distributed Processing [1], also communicates by rendezvous. The virtues of the rendezvous are in simplifying program and system behaviour, bounding the message queues, making software easier to verify, and in reducing the copying overheads associated with asynchronous inter-task communications. The synchronism of the rendezvous does, however, pose special problems in performance analysis. This paper describes a modelling framework, the Stochastic Rendezvous Network (or SRN) and gathers together and generalizes some recent work on solution techniques. The SRN includes classical queueing networks as a special case. The SRN models in this paper are for software on closely coupled multiprocessors or on local networks.

Models for this class of systems in the previous papers [6, 3, 4, 18] (to be described in the following section) have dealt with delays to requests when they queue at a server task, and with the effect of internal rendezvous delays on the service times of server tasks. The present paper extends these models to server tasks which offer different services (called here "entries"), with each entry having a distinct execution behaviour, and the entries sharing a common FIFO queue. This is similar to having classes of customers at a FIFO queue, in a queueing network. Other new features account for a client dependency phenomenon called "interlocking", and communications delays. The approach taken here is also new, in that it goes back to the basic concepts of Mean Value Analysis expressed in the Arrival Theorem [31], and constructs (approximate) expressions for states at the instant of an arrival of a request to a server task. The resulting algorithm has been adapted to incorporate previous ideas in contention for processors (from [4]) and priorities (from [18]), so it unifies as well as extends the treatment of those systems. A companion paper [8] has described a second approach to solving for clients with entries, which is more accurate in a certain range of cases but does not account for interlocking or priorities.

2 Stochastic Rendezvous Network (SRN) Models

In an SRN model, entities called "tasks" represent hardware or software objects which may execute concurrently. A single task does not have any internal concurrency. Tasks have random (i.e. data-dependent) execution times and communications patterns, and they communicate by messages in a request-wait-reply sequence which models a rendezvous. Figure 1 describes the sequence. The task which sends the message is the *client*, and is said to request the rendezvous; it blocks in the "rendezvous delay" until it gets the reply. The receiving or *server* task is said to accept the request, and executes two or more phases. The first phase is a *service* phase, and is similar to service given in a queueing network. The second and later

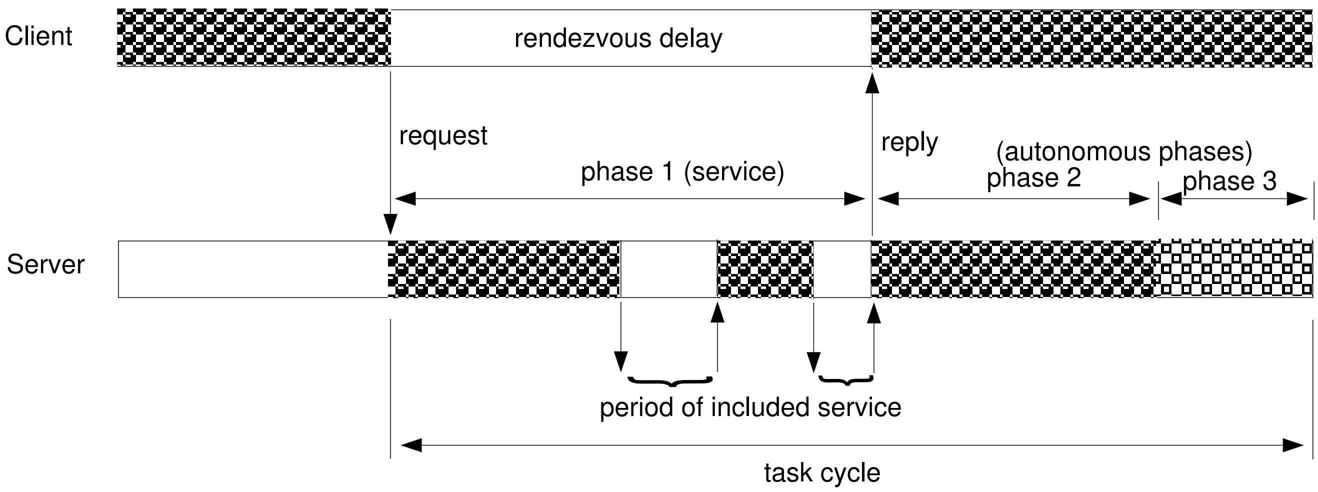


Figure 1: Task Cycles and Phases

phases are *autonomous* phases in which the server task acts completely on its own, after being "launched" by the first phase.

The two features which distinguish an SRN from a queueing network are

1. the second and subsequent phases of autonomous behaviour,

2. the fact that during any phase the server may act as a client requesting service from a third task, a lower level server. The associated delay is termed *included service.*

Some software systems use only a subset of the full rendezvous pattern shown in Figure 1:

- The standard Remote Procedure Call (RPC) has no second phase. However an RPC server may be implemented with a second phase to do house-keeping work such as deleting buffers and closing files after the reply, so as to delay the RPC client as little as possible.

- In the hardware rendezvous implemented in the Transputer and exploited by the Occam language there is no first phase, only an exchange of parameters which is equivalent to a request immediately followed by a reply. The rendezvous in the specification languages CSP, CCS and LOTOS are similar. CSP also permits a more general rendezvous among three or more processes, which is excluded in the SRN.

- In V the rendezvous is implemented by send and reply messages, as shown here, and in Ada there is a procedure-like semantics with the same effect. SRNs are particularly close to the semantics of Ada.

- In any system, a software process requests processor service by joining a ready-to-run queue as a client to the processor, which takes the role of server. In an SRN the processor is represented as a task, nominally without a second phase. However second phase processing could be used to model scheduling overhead, before the next software application task can begin service.

Figure 1 does not show any delays in transferring messages, however the model in Section 3 does include a communications delay.

Entries

A server may provide more than one service, or it may provide service with different performance parameters for different clients. Consider the example shown in Figure 2, which represents data storage/retrieval servers for a real-time system. Six client tasks are shown, in two groups of three. Each group makes retrieval and update requests from its own local data server, LocalServA or LocalServB; if the request cannot be handled locally it is forwarded to the global server GlobalServer. Different services are offered by different *entries*. Retrievals are handled by 'GET' entries in phase 1, with the data returned in the reply, while updates are handled by the UPDATE entries as phase-2 work, after confirming the data handover in a reply. Each entry thus has its own parameters for execution and communications. This system was implemented in the laboratory on a multiprocessor with a real-time kernel. Measurements on it show that in practice, execution for context switching and message handling ensures that the GET entries also have phase 2 execution, and UPDATE entries have significant phase-1 execution.

Figure 3 gives another example of entries, showing a system with two client tasks and two levels of servers. In this case there is no strict layering of services. Application 1 consists of Tasks 1 and 3, where 3 is a server to 1; Application 2 has a single task 2. Both applications use server tasks 4 and 5 (server A and server B), and server A sometimes itself uses server B. Tasks 1 and 2 drive the system by cycling perpetually and never wait for input; this makes Figure 3 a *closed* model (as also is Figure 2). Tasks may also have input from outside the model, giving an *open* or mixed *open-and-closed* model.

The notation in Figures 2 and 3 represents each task by a parallelogram, which may be divided internally into separate entries. Inside each entry in square brackets is a list of average phase execution times, and attached to each arc in round brackets is a list of mean numbers of messages sent, in each phase of the sending task. Processors are not represented explicitly because each task has its own processor; processors are discussed at greater length in Section 3. The explicit model for separate entries within a task is the principal contribution of this paper.

Related Research

Classical queueing models as described in [11, 30] estimate contention by independent tasks for the devices of a system. They have been extended by various authors using approximations such as flow-equivalent aggregation methods and surrogate delays to determine the delays due to passive resources; de Souza e Silva et al [33, 34] give some examples and references. Some attempts have also been made to analyze interdependent tasks, particularly

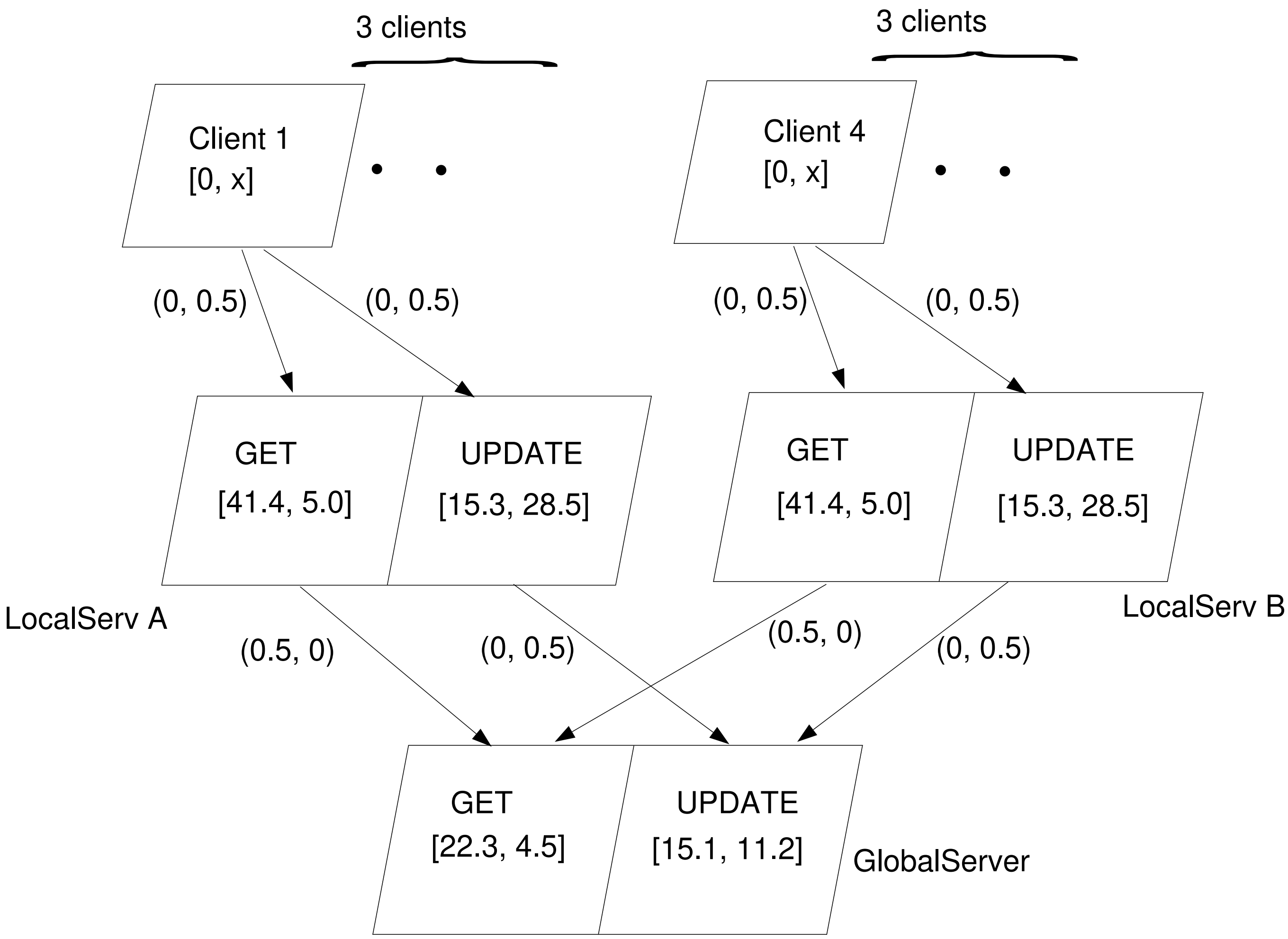


Figure 2: Real-Time Data Server

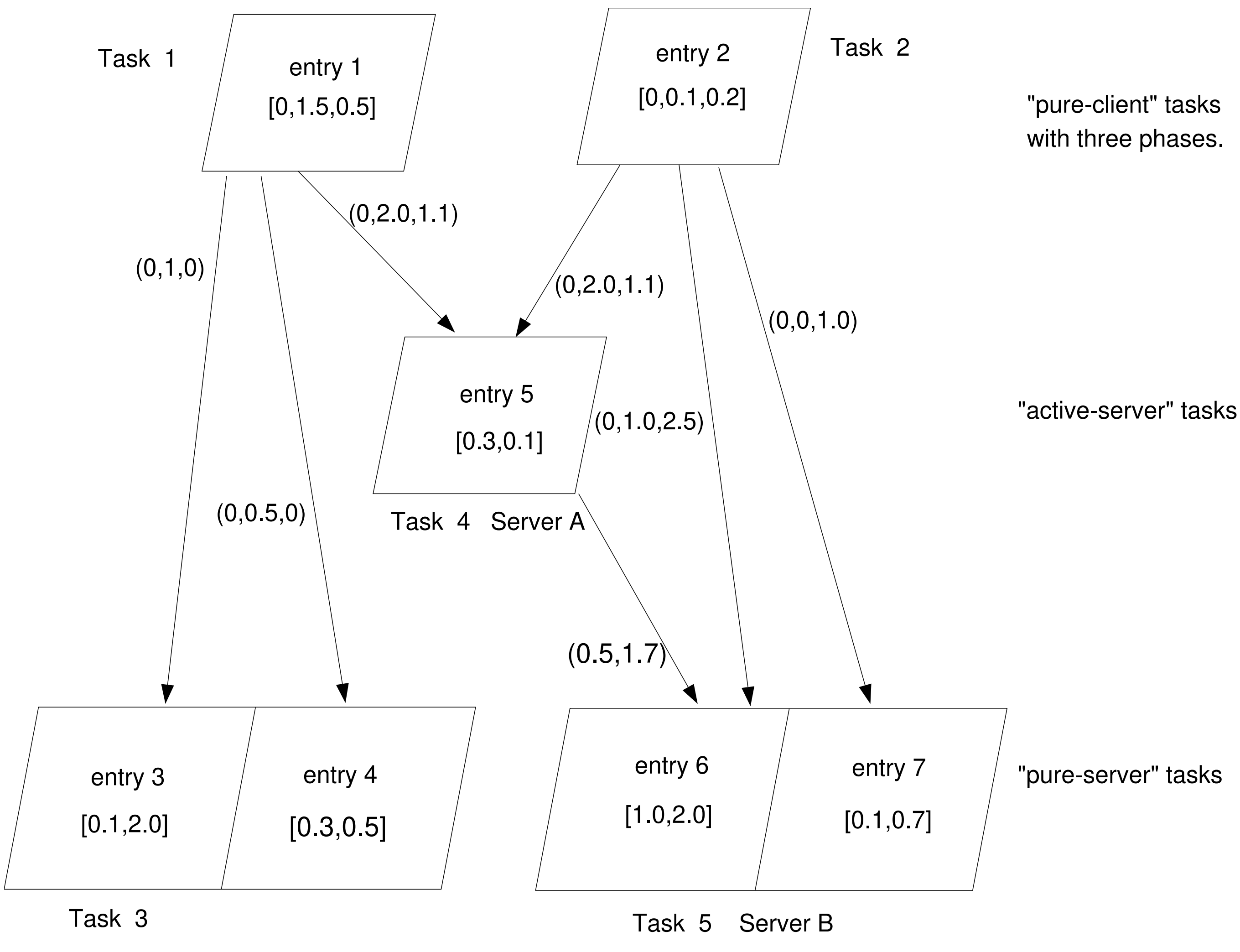


Figure 3: A Stochastic Rendezvous Network with Two Reference Tasks
(the numbers in square brackets are execution times of each entry, by phase, while those in round brackets are the mean numbers of messages from entry to entry, by the phase of the sender.)

in the context of split-join patterns in the execution flow [12, 14, 32]. However (as discussed in [13]) splitting and joining of flows is just one of many forms of interdependence in the execution of related tasks.

A more detailed approach is to model the tasks and their interactions with Petri Nets [17] and to analyze performance via an appropriate form of Timed Petri Net [20, 21]. This involves direct Markov Chain techniques which, as is frequently pointed out, are often impractical because of state explosion. Some work has been done on decomposition to overcome this problem, for example by Ciardo and Trivedi [7].

Compared to Petri Nets, the SRN framework is at a higher level of abstraction. Queueing and synchronization involving intertask messages are implicit, so a given model can be stated much more compactly. The SRN thus applies the concepts of queueing networks, which have been of great power in modelling hardware servers, to software and hardware servers together.

Previous papers on SRNs defined the active server concept [6], and analyzed a "basic" SRN in which every task has its own processor [3]. Processor contention was examined in [19], and priorities for tasks on processors in [18]. Special considerations for pipelines of tasks were given in [5]. These papers all contributed to developing a distinctive approximation loosely based on Mean Value Analysis for queueing networks, adapted to deal with two phase service and included service. Other authors have examined models partly equivalent to SRNs. Agrawal and Buzen's "Aggregate Server Method" [29] models clients with included service but only a single level of servers (plus a third level for processors). Rolia's "Lazy Boss Method" [15] allows any number of levels of servers, cycles in the request graph and priorities. He mapped the SRN onto a queueing network and used MVA directly, with good results. However he did not include the second phase of service or task entries. His "Method of Layers" [16] incorporates many of the features of the present work but requires a strict layering of the servers (servers can only use servers in the next layer down), which we do not. Fontenot also examined a single open queue in which the server obtained included service at a second queue in [22]. There were no second phases of service, and his methods do not generalize to networks. Recently Petriu and Woodside reported an improved waiting time calculation developed specifically for SRNs, from decomposition of a Markovian model for clients at a single server [26],[8]. It is built upon, and makes use of the modelling framework described here. Unfortunately it is not applicable to some important cases such as priority queues or to the "independent phases" defined in Section 3.

The main novelty in the present work is the modelling of entries to tasks in combination with second phases and included service, the uniform treatment of processors as pseudo-tasks, and the uniform modelling of priorities of various kinds. Entries significantly complicate the analysis over that given in [3] because they are analogous to multiple classes of service with different service times and with FIFO queueing.

The model and notation are defined in Section 3, including a transformation to describe processors by equivalent pseudo-tasks. Section 4 describes the throughput approximations. Section 5 states the SRN algorithm for calculating the solution, and discusses its complexity and accuracy. Sections 6 and 7 describe two additions to the algorithm, to deal with very unbalanced entries and with priorities.

3 A Formal Model

The SRN model with entries can be described under four headings: tasks, entries, phases, and throughputs.

1. Tasks: a task is an object which has a single thread of control (no internal concurrency) and which can initiate or accept service requests. There are three types of task as indicated in Figure 3:

   - *pure clients*, numbered in this paper as tasks 1 to R, only initiate requests. They represent independent application software tasks, and are analogous to customers in a queueing network.
   - *active servers*, tasks $R+1$ to K, accept requests and also initiate them.
   - *pure servers*, tasks $K+1, \ldots N_T$, only accept requests and are analogous to servers in a queueing network, except that they may have second phases.

   A task may model a software process or a hardware device; a processor always maps to a pure server. When a software task runs on its own private processor then it and the processor are modelled as a single "task" object.

2. Entries: an entry is a subdivision of a task corresponding to a particular service. Messages are addressed to entries, and each entry has its own execution parameters. An entry corresponds to the Ada notion of entry, or to the notion of a "method" of an object in Smalltalk.

   - a request to entry e goes into a single queue common to all entries of the task.
   - a server task (active server or pure server) executes in a loop as follows:

     (i) examine the message queue and select one message according to the queue discipline,

     (ii) if it is for entry e, execute entry e until completed.
   - a pure client task implicitly has just one entry which it executes in an infinite loop. For simplicity entries 1 to R are taken to be the single entries of the corresponding pure client tasks 1 to R. Entries $R+1, \ldots, N$ are associated with server tasks.
   - $\mathcal{E}(i) =$ the set of entries for task i (e.g. in Figure 3, $\mathcal{E}(5) = \{6, 7\}$)

     $T(e) =$ the task for entry e (e.g. in Figure 3, $T(6) = 5$)

     $\mathcal{S}(e) =$ the set of sibling entries to entry e, including itself. That is $\mathcal{S}(e) = \mathcal{E}(T(e))$; in Figure 3, $\mathcal{S}(6) = \{6, 7\}$.

     $\mathcal{E}^* =$ the set of all entries

     $\mathcal{T}^* =$ the set of all tasks.

     A task in fact is just a collection of entries, with a message queue.

3. Phases: each entry e has P_e phases of execution before it completes. Most models have $P_e = 2$ but three phases were found necessary in [5] for pipeline tasks (phases for input, processing and output).

   - phase 1 is a service phase, and is absent from entries of "pure client" tasks (1 to R).
   - phase 2 is an autonomous phase (and so are later phases, if any).
   - phase (e, p) denotes 'phase p of entry e'.
   - messaging is stochastic; the distribution of the number of messages per phase is geometric.
   - δ_{ed} = mean roundtrip communications delay from $T(e)$ to $T(d)$, beyond that included in the execution time of entries e and d. This is the mean network delay in a distributed system; it is usually almost zero in a bus-connected system, or with tasks on the same processor. It is represented here by a constant mean value, which assumes that the delay is substantially determined by other factors besides the software being modelled, and can be determined or measured in advance.

     s_{ep} = mean total execution time of (e, p),

     y_{edp} = mean number of requests to entry d, during (e, p). Note that $T(d) \neq T(e)$,

     $Y_{ed} = \sum_p y_{edp}$ = mean number of requests to d during all phases of entry e.

     x_{ep} = mean duration of (e, p), also called the *service time* of (e, p):

$$x_{ep} = s_{ep} + \sum_{d \in \mathcal{E}^*} y_{edp}(t_{ed} + \delta_{ed}) \tag{1}$$

     t_{ed} = mean delay to entry e when it sends a message to entry d,

     $X_e = \sum_p x_{ep}$ = mean duration of entry e.

   - execution by entry e is divided up into *slices* by the internal rendezvous in each phase. These slices are assumed to be exponentially distributed with mean ξ_{ep} for each slice of phase (e, p).

$$\xi_{ep} = s_{ep}/(1 + \sum_{d \in \mathcal{E}^*} y_{edp}) \tag{2}$$

4. Throughputs of the entries are the main results (together with the mean queueing delays determined later):

   - λ_e = throughput of entry e in messages/sec.
   - any server entry e may have an *open* or external arrival stream of requests, of rate λ_{0e}.

- rates are algebraically related to each other by traffic equations given by:

$$\lambda_e = \lambda_{0e} + \sum_{d \in \mathcal{E}^*} Y_{de} \lambda_d \qquad e = R+1, \ldots, N \ . \tag{3}$$

  The right side adds up the rates of the various sources of possible requests to entry e. There are N–R equations in N unknowns, so there is no unique solution. However, there is a solution for N–R throughputs in terms of the remaining R throughputs, say in terms of $\lambda_1, \ \ldots \ \lambda_R$, in the following general form:

$$\lambda_e = \sum_{r=1}^{R} \alpha_{er} \lambda_r + \sum_{d=R+1}^{N} \beta_{ed} \lambda_{0d} \ , \quad e = R+1, \ldots, N \ . \tag{4}$$

  The coefficients α and β are determined by applying Gaussian elimination to the set of equations (3) above.

- other rates of message transfer in the model are defined as follows:

  λ_{ed} = rate of messaging from entry e to entry $d = \lambda_e Y_{ed}$.
  $\lambda_{ij}^{T \to T}$ = rate from task i to task $j = \sum_{e \in \mathcal{E}(i), d \in \mathcal{E}(j)} \lambda_{ed}$, for $i \neq j$.
  $\lambda_{id}^{T \to E}$ = rate from task i to entry $d = \sum_{e \in \mathcal{E}(i)} \lambda_{ed}$, for $i \neq T(d)$.
  $\lambda_{ej}^{E \to T}$ = rate from entry e to task $j = \sum_{d \in \mathcal{E}(j)} \lambda_{ed}$, for $T(e) \neq j$.

- because the "pure client" tasks loop forever, their throughputs are given by

$$\lambda_r = 1/X_r \quad r = 1, \ldots, R \tag{5}$$

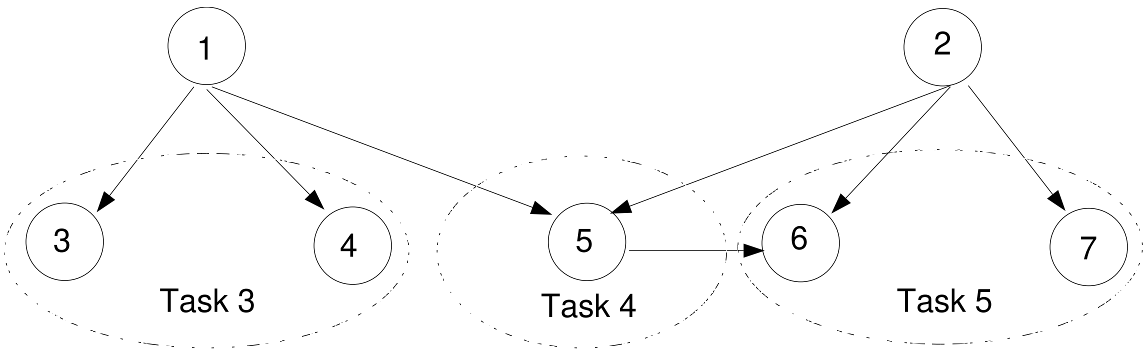
  where r is the label of both the task and of its single entry, as described earlier.

Processor Transformation

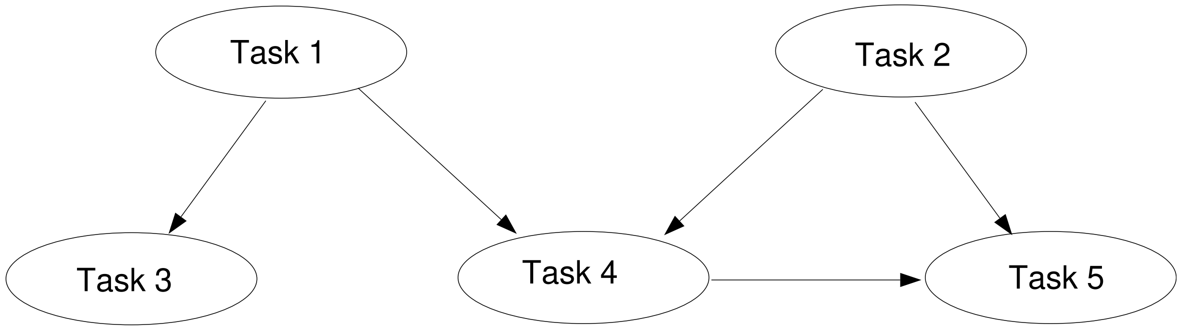
A processor is modelled by a pure server task which accepts requests for execution from the entries of the tasks assigned (statically) to it. To avoid ambiguity this discussion will refer to it as a 'pseudo-task', but in the model it is a task like any other. The entries of the tasks assigned to it request execution one slice at a time, and each request and execution is modelled in the SRN as a rendezvous between the requesting task and the processor pseudo-task. The task sends a ready-to-run 'message' to the processor, and the 'reply' comes when the execution slice is completed. Thus, the execution time of the software is transferred to the processor pseudo-task. Because the $\xi_(e,p)$ may all be different, each one has a separate entry $\epsilon(e,p)$ in the pseudo-task with $s_{\epsilon 1} = \xi_{ep}$.

When a task i is assigned to a processor modelled by pseudo-task j, the following transformation is carried out for each phase (e,p) executed by task i:

i) define entry $\epsilon(e,p)$ in pseudo-task j for each phase (e,p) of task i, and an arc from e to ϵ for its processing requests.

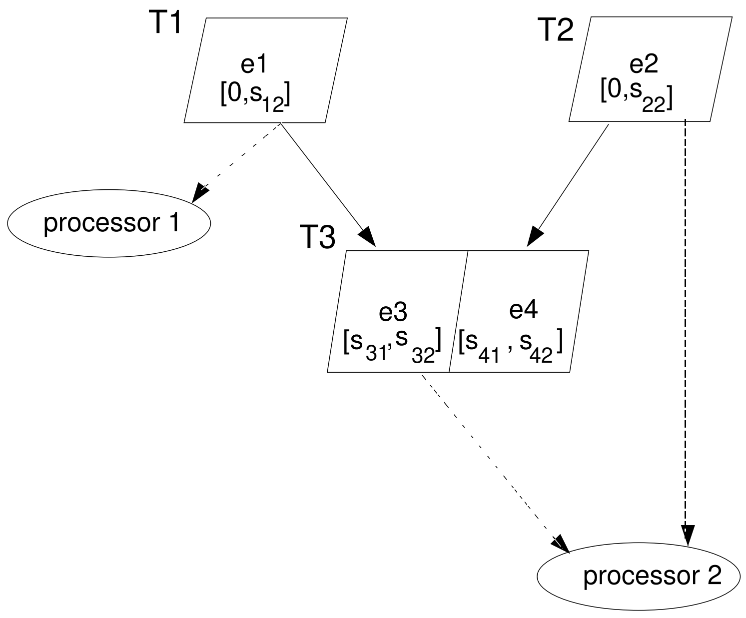


(a) SRN Entry Graph

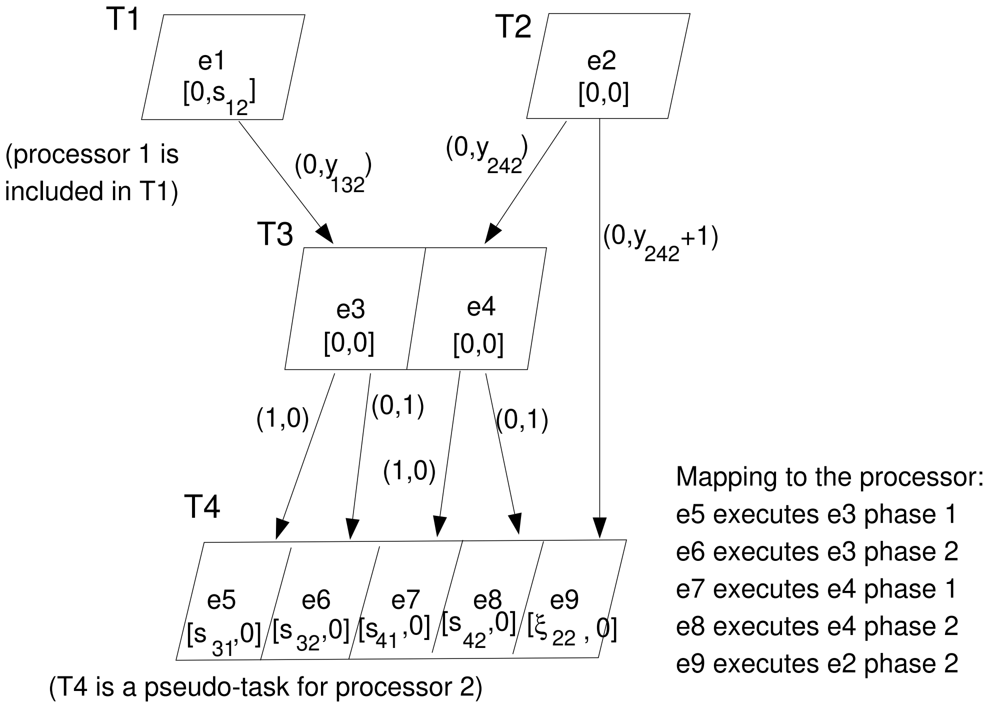


(b) SRN Task Request Graph

Figure 4: Two Graphs for the System of Figure 3



(a) A small example, showing how software tasks (parallelograms) are allocated to processors (ovals)



(b) The same example, with the processors incorporated into the model

Figure 5: Transformation to include processors in the model

ii) label the arc with the request rate $y_{e\epsilon p}$ defined by

$$y_{e\epsilon p} = 1 + \sum_{d \in \mathcal{E}^*} y_{edp} \tag{6}$$

which is the mean number of processing slices in phase (e, p). This assumes that a processor scheduling is always performed between phases.

iii) if s_{ep} is the execution demand by (e, p), compute ξ_{ep} from it using equation (2). Set $s_{ep} = 0$ and $s_{\epsilon 1} = \xi_{ep}$, to associate the execution with the processor rather than the software. Pseudo-task j is a pure server, and entry $\epsilon(e, p)$ has only phase 1.

If a task is assigned alone to its own processor, there is no processor contention to be determined, and therefore there is nothing to be gained by modelling the processor separately. In these cases, the processor and the task are modelled together as a single SRN task, and the transformation is not applied. Whenever task i (in software) is allocated to a processor modelled by pseudo-task j, define

$P(i) =$ the pseudo-task modelling the processor for i; $P(i) = j$.

$\mathcal{T}(j) =$ the set of software tasks allocated to j.

If j does not model a processor then $\mathcal{T}(j)$ is empty, and if a task i has its own processor then $P(i) = i$.

Figure 5(a) shows an example with one task on processor 1, and two tasks which share processor 2. Figure 5(b) shows the result of the processor transformation, with the new task $T4$ being a pseudo-task to represent processor 2. Notice how $T4$ has an entry for each phase of each entry of each task assigned to the processor.

Two Graphs Associated with an SRN

An SRN model (after the Processor Transformation) can be summarized by an *SRN Entry Graph* whose nodes are entries labelled by phases with workload parameters s_{ep}, and with directed arcs labelled by traffic parameters y_{edp}.

From the SRN Entry Graph a second graph can be derived, termed the *Task Request Graph*, with a node for each task and a directed arc from node i to node j if any entry e in task i sends requests to any entry d in task j. This graph is used to determine the order of computation in the algorithm of Section 5.

Figures 4(a) and (b) show the two graphs corresponding to Figure 3. There is an implicit constraint on the Task Request Graph that no arc may go from node i to itself, and we also restrict our attention to acyclic graphs. If the graph is acyclic then the system is free of so-called 'rendezvous deadlocks', characterized by a circular chain of blocked tasks.

4 Contention Delays at Tasks and Processors

The definitions and relationships given above provide a framework for the performance calculation. The key missing element is the queueing delays when messages are sent to tasks, or when tasks are ready to run on processors. The following outline links this missing part together with the parts already described, to give a calculation for throughputs:

1. the parameters λ_{0e}, s_{ep} and y_{edp} are the inputs to the calculation,
2. determine the contention delays (as described in this section),
3. from them, determine the mean request-response delays t_{ed},
4. determine the phase service times x_{ep} from t_{ed} and (1),
5. determine the throughputs of pure client tasks, from X_r and (5),
6. determine other entry throughputs, from (4).

Because each step takes inputs computed at other steps, iteration is applied to steps 2–6.

Before the algorithm is stated the contention calculation in step 2 will be developed, in two stages. The first stage adapts various MVA concepts which are conventional for FIFO queues to SRNs, in terms of certain arrival-instant probabilities. The following stage gives approximations for these probabilities.

Queueing Delay for FIFO Queueing

We will now consider t_{ab} introduced in (1), giving the delay between sending a message from a to b and receiving the response. The task $T(b)$ has a single queueing discipline covering all its entries, but different clients may see different mean delays. Let w_{ab}^{FIFO} be the mean FIFO waiting seen by entry a sending a message to entry b, not including service, while its service time by $T(b)$ is x_{b1}, the duration of phase 1 before the reply is sent to a. These two components add up to t_{ab} defined above, and for FIFO queueing give:

$$t_{ab}^{FIFO} = w_{ab}^{FIFO} + x_{b1} \ . \tag{7}$$

When a task i is blocked while a message waits at server task j, we will refer to task i or its entry e as being either "in the queue" or being "at the server". The waiting part w_{ab}^{FIFO} depends on the state of the task $T(b)$ at the instant the request is made – the "arrival-instant" conditions. The events which may occur at the instant entry a sends a message to entry b are:

- $InService(a, b, c, d, p)$ = event that a request from entry a, arriving at entry b, finds task $T(b)$ executing entry d phase p in response to a message from entry c.

- $InQueue(a,b,c,d)$ = event that a request from entry a arriving at entry b finds an entry c request queued (but not in service) for entry d.

For both $InService$ and $InQueue$, it is implicit that a, b and c belong to separate tasks and b and d are entries of the same task, so $c \notin \mathcal{S}(a)$, $c \notin \mathcal{S}(b)$, and $d \in \mathcal{S}(b)$. Now using the Arrival Theorem [31] the waiting time can be expressed as a component for residual service, plus a component for each message in the queue at $T(d)$:

$$w_{ab}^{FIFO} = \sum_{c\in\mathcal{E}^*} \sum_{d\in\mathcal{S}(b)} [X_d \mathrm{Prob}\{InQueue(a,b,c,d)\} + \sum_{p=1}^{P_d} V(a,b,d,p)\mathrm{Prob}\{InService(a,b,c,d,p)\}] \tag{8}$$

where $V(a,b,d,p)$ is the residual time of the service underway at the time of arrival.

The probabilities of $InService$ and $InQueue$ will be estimated in the next section. The residual service term $V(a,b,d,p)$ may be found as follows. First, it will be assumed that V depends only on the entry and phase which is executing at the arrival instant, and not on the identity of the arriving message or the entry to which it is addressed. Second, the Mean Residual Life, as defined in Renewal Theory, will be used for phase (d,p) – call it $MRL(x_{dp})$. Thus:

$$V(a,b,d,p) \approx MRL(x_{dp}) + \sum_{u=p+1}^{P_d} x_{du} \ . \tag{9}$$

The MRL term depends on the detailed time-structure of a phase, for which there are two important cases:

- *Markov phases* proceed in a sequence of independent decisions made at the end of each execution slice, to end the phase there or to send another message. For this structure, and assuming exponential subintervals for task blocking, it was shown in [3] that:

  $$MRL(x_{dp}) \approx x_{dp} + \sum_e \frac{y_{dep}t_{de}^2}{x_{dp}} \quad \text{(Markov phases)} \ . \tag{10}$$

  Then

  $$V(a,b,d,p) \approx x_{dp} + \sum_e \frac{y_{dep}t_{de}^2}{x_{dp}} + \sum_{u=p+1}^{P_d} x_{du} \quad \text{(Markov phases)} \ . \tag{11}$$

- *independent phases* have independent values for the execution slices and for each message count y_{dep}. In this case it is straightforward to extend the model to allow execution slices with general distributions and coefficient of variation c_{dp} for phase p of entry d, to go with the mean value ξ_{dp}. The variance of the entire phase is $Var(x_{dp})$. Blocking delays are assumed to be exponential and the various components of the phase are independent, from which it is easy to show that

  $$Var(x_{dp}) = c_{dp}^2 s_{dp}^2/(1 + \sum_e y_{dep}) + \sum_e y_{dep}t_{de}^2 \ .$$

Then

$$MRL(x_{dp}) = (x_{dp}^2 + Var(x_{dp}))/2x_{dp} \quad \text{(Independent phases)} \ . \tag{12}$$

$$V(a,b,d,p) = (x_{dp}^2 + Var(x_{dp}))/2x_{dp} + \sum_{u=p+1}^{P_d} x_{du} \quad \text{(Independent phases)} \ . \tag{13}$$

At this point an MVA equation for the contention delay is established in terms of the unknown probabilities of the arrival-instant events $InService$ and $InQueue$ defined above; these will now be considered.

Arrival-Instant Probabilities

The probabilities of the events $InService(a,b,c,d,p)$ and $InQueue(a,b,c,d)$ defined previously will now be approximated in terms of other quantities. The arrival-instant probabilities are affected by the fact that the arriving message comes from a particular task i and entry a; the discussion below also assumes that the message is sent to task j ($j \neq i$), entry b, and that entry c of task k can also send messages to entry d of task j. When task i is blocked while a message waits at server task j, we will refer to task i or its entry e as being "in the queue" or being "at the server".

The calculation uses the following steady state probabilities:

$\lambda_{cd} x_{dp}$ = steady-state probability that task j is serving a message sent from c to d, and is in phase p. In this terminology the second and later phases are described as "serving" the message that started the first phase.

$\lambda_{cd} w_{cd}$ = steady-state probability that a message sent from c to d is in the queue at task j, awaiting service.

U_{ijp} = steady-state probability that task j is serving a message from task i, and is in phase p,
$= \sum_{a \in \mathcal{E}(i), b \in \mathcal{E}(j)} \lambda_{ab} x_{bp}$

U_{ij} = steady-state probability that task j is serving a message from task $i = \sum_p U_{ijp}$

U_j = steady-state probability that task j is busy $= \sum_i U_{ij}$.

Naive estimates of the arrival-instant probabilities could be made by using the steady-state probabilities for the conditions at task $T(b)$, not conditioned on an arrival instant of a message from a to b. As defined above, these are

$$\text{Prob}\{InService(a,b,c,d,p)\} = \lambda_{cd} x_{dp} \quad \text{(naive)} \tag{14}$$

$$\text{Prob}\{InQueue(a,b,c,d)\} = \lambda_{cd} w_{cd} \quad \text{(naive)} \tag{15}$$

As is also done in queueing network Mean Value Analysis, better approximations will be constructed below by modifying these values to account for the sender of the message. However because of blocking, included service and second phases, and the association of entries of the same task, the approximations are different from those in queueing networks.

The approach adopted was to consider how the flow rates λ_{cd}, as they occur in (14) and (15), are modified by the fact that entry a is blocked with a message to entry b. For example, if c is any entry from the same task, or any entry which requires a or one of its sibling entries to be blocked, then it cannot 'arrive' at task j. Some arrival rates become zero, while others are modified, and the phenomenon is termed the 'interlocking' of entry arrival rates. An analysis is given in Appendix A, leading to a modified effective 'arrival-instant' flow rate approximation $\lambda_{cd}^{(ab)}$ which is used in place of λ_{cd}. The definition is:

$\lambda_{cd}^{(ab)}$ = the effective competing rate of requests from entry c to entry d, seen by an arriving request from a to b, when $d \in \mathcal{S}(b)$

Other factors will be introduced in the detailed derivation of the probabilities, case by case. We can distinguish between the 'service state' events $InService$ and the 'queue state' events $InQueue$, and also between events which are 'overtaking' and 'non-overtaking'. The cases and the probabilities are now derived.

(1) Service State Probabilities for 'Overtaking' Events ($c \in \mathcal{S}(a)$, $p > 1$)

A message arrival is an *overtaking event* if it is an arrival to a task which is busy executing a second (or later) phase which was initiated earlier by a message from the same sending task i. This can only occur when there are second phases, and does not occur in queueing networks. Since overtaking analysis only considers the behaviour of the arriving task, interlocking is not a factor. An analysis of the probability of overtaking is given in Appendix B with the result:

$$\mathrm{Prob}\{InService(a, b, c, d, p)\} = \lambda_{cd}\mathrm{Prob}(\alpha_p)/\lambda_{ij}^{T\to T} \tag{16}$$

$\mathrm{Prob}(\alpha_p)$ is described in Appendix B. For $c \in \mathcal{S}(a)$ and $p = 1$, $\mathrm{Prob}\{InService(a, b, c, d, p)\} = 0$. It is also useful to define $P_{OT}^{(a,b)}$ as the probability of all overtaking events together, when a message from a arrives at b:

$$P_{OT}^{(a,b)} = \sum_{c\in\mathcal{S}(a)} \sum_{d\in\mathcal{S}(b)} \sum_{p>1} \mathrm{Prob}\{InService(a, b, c, d, p)\} \, . \tag{17}$$

(2) Service State Probabilities for 'Non-Overtaking' Events ($c \notin \mathcal{S}(a)$)

In non-overtaking cases the $InService$ events will now be considered. The arrivals which are not overtaking have probability $(1 - P_{OT}^{(a,b)})$, and are divided between events which see task j serving a message from some entry $c \notin \mathcal{S}(a)$, and events which find task j idle. Beginning from (14), including interlocking, and conditioning the probability on the fact that these arrivals do not occur while j is serving a message from i, we obtain:

$$P\{InService(a,b,c,d,p)\} = \frac{\lambda_{cd}^{(ab)} x_{dp}(1 - P_{OT}^{(a,b)})}{1 - \text{Prob}\{j \text{ serving a message from } i\}} = \frac{\lambda_{cd}^{(ab)} x_{dp}(1 - P_{OT}^{(a,b)})}{(1 - U_{ij})} . \tag{18}$$

(3) Queue State Probabilities for 'Overtaking' Events ($c \in \mathcal{S}(a)$)

When a arrives at b, it is impossible for any sibling $c \in \mathcal{S}(a)$ to be waiting in the queue, thus

$$c \in \mathcal{S}(a): \ \text{Prob}\{InQueue(a,b,c,d)\} = 0$$

(4) Queue State Probabilities for 'Non-Overtaking' Events ($c \notin \mathcal{S}(a)$)

The queue-state arrival events $InQueue(a,b,c,d)$ show entry c waiting in the queue for entry d when a arrives at b. The naive calculation with interlocking would give $\text{Prob}\{InQueue(a,b,d,c)\} = \lambda_{cd}^{(a,b)} w_{cd}$, but it is affected by two further factors. First, sibling entries $c' \in \mathcal{S}(a)$ cannot, obviously, appear in the queue, so the same factor $1/(1 - U_{ij})$ appears here, as in Case 2. Second, the probability that any other entry c from some other task k is in the queue is reduced by the fact that neither a nor any of its siblings can be in service, at the arrival instant. The queueing probability for task k is therefore reduced in proportion to

$$1 - \frac{\text{Prob}\{j \text{ is serving } i \text{ in phase } 1\}}{\text{Prob}\{j \text{ is busy but not serving } k \text{ in phase } 1\}} = 1 - \frac{U_{ij1}}{(U_j - U_{kj1})} . \tag{19}$$

Taking these factors together we have the approximation for $c \notin \mathcal{S}(a)$:

$$\text{Prob}\{InQueue(a,b,c,d)\} = \lambda_{cd}^{(ab)} w_{cd} \frac{1}{1 - U_{ij}} (1 - \frac{U_{ij1}}{U_j - U_{kj1}}) . \tag{20}$$

This completes the information necessary to determine a solution.

5 The SRN Algorithm

The following algorithm assembles the foregoing relationships into an iterative calculation for the performance of an SRN:

1. Carry out the processor transformation to create tasks to represent processors which have co-allocated tasks.

2. Determine a total ordering of the entries such that, if there is an arc from i to j, then all entries in j precede all entries in i in the ordering. If such an ordering cannot be found, there is a cycle in the Task Request Graph and the system may have a rendezvous deadlock due to a waiting loop of tasks in phase 1. The analysis should not proceed.

3. Initialize: Determine α and β coefficients in (4) by Gaussian elimination applied to the traffic equations (3). Set $w_{ab} = 0$ for all entries a, b with $Y_{ab} > 0$.

4. Service and throughput values:

   (a) for each task i, in the order found at step 1, apply (1) and (7) to each entry a and to each phase, to give $x_{ap} = s_{ap} + \sum_b y_{abp} x_{b1}$,

   (b) $\lambda_r = 1/\sum_p x_{rp}$ for $r = 1$ to R, from (5),

   (c) for entries $a = R+1$ to N, (4) gives $\lambda_a = \sum_{r=1}^{R} \alpha_{ar}\lambda_r + \sum_{e=1}^{N} \beta_{ae}\lambda_{0e}$.

5. Waiting w_{ab}: for all entries a, b with $Y_{ab} > 0$:

   (a) for each $d \in \mathcal{S}(b)$, and each c with $Y_{cd} > 0$, compute:

   - $\lambda_{cd}^{(ab)}$, according to Appendix A,
   - Prob$\{InService(a, b, c, d, p)\}$ for $p = 1$ to P_d, using either equation (16) and Appendix B, or equation (18),
   - Prob$\{InQueue(a, b, c, d)\}$ using equation (20),
   - $V(a, b, d, p)$ using equation (11) for Markov phases or (13) for Independent phases.

   (b) compute w_{ab}, using (8) for FIFO queueing, or (21) or (22) for priority queueing at $T(b)$.

6. Convergence Test: if throughputs λ_r are all sufficiently close to the previous iteration, stop. Otherwise repeat from step 4.

7. Results are throughputs λ_r, mean waiting times w_{ab}, entry service times x_{ep} and X_e, and entry and task utilizations.

In practice, the algorithm was always used with the Fast Coupling heuristic described later; however the description of this heuristic has been postponed to make the algorithm more understandable. The delay at a Poisson arrival point is calculated using an equivalent M/G/1 model for the task which is the arrival point, using the service time and mean residual life found from the SRN solution.

Complexity

The overall complexity of the algorithm presented in this paper is dependent on the computational complexity of each iteration as well as the number of iterations required. Analysis of the mean number of operations required per iteration is presented first. Let

N = Total number of entries in the model,

P = Mean number of phases per entry,

Table 1: Complexity Study: Operations per Iteration, by Equation Number

Parameter (Equation No.)	Dependence on the number of operations per iteration
λ_e (3)	$N(N-R)$
λ_r (5)	RP
x_{ep} (1)	NQP
w_{ed} (8)	NQ^2PZ
$\{U_j\}$	N/Z
$\{U_{ij1}\}$	NQZ
Prob$\{InQueue\}$ (20)	NQ^2Z
Prob$\{InService\}$ (16),(18)	NQ^2PZ

Q = Mean indegree and outdegree of an entry,

Z = Mean number of entries per task.

Each iteration requires the computation of a number of parameters, some of which are then used in more than one equation. The computational complexity of the most significant parameters dominating the mean computation time per iteration is presented in Table 1.

If N is variable and P, Q, Z, are roughly constant then in our experience the complexity per iteration grows only slightly faster than N. Although the computation of the λ_e parameters take of the order of N^2 operations, the constant of proportionality for this term is very small. Thus for moderate N (say, less than about 100), the complexity of each iteration is dominated by terms which are linear in N. For a given N the software architecture of the system as represented by Q and Z can also have significant impact on the complexity of each iteration.

The number of iterations also increases with N, but in our experience the increase is small, much less than linear in N. The actual solution time experienced with the algorithm, for the system in Figure 3 and Table 2, was about one second on a SUN Sparcstation. For a large model with about 80 tasks (each with a single entry), an early version of the algorithm converged in about ten times longer.

Since each computation also requires storage, and all the intermediate computations are stored, the space complexity is of the same order as the time complexity.

Experience and Accuracy

Proof of convergence of this algorithm is an open question. However in practice it has given no difficulty provided an under-relaxation strategy used previously in [3] was applied. Service time updates were computed for each iteration as shown in the SRN algorithm, but only half of the change in each service time was applied, i.e. $x_{\text{new}} = .5\,(x_{\text{old}} + x_{\text{computed}})$. Convergence typically required 10–20 iterations.

Three types of experience will be described. First, a large set of models with R client tasks accessing one server with R entries (each client accessing its own entry) were solved exactly by Markovian analysis, and approximately. The mean absolute value of the percent error, averaged over all client throughputs in 35 examples with widely assorted parameter values, was

- for 3 clients, error = 8.0%
- for 4 clients, error = 9.6%
- for 5 clients, error = 9.3%.

Considering that these experiments have FIFO queueing and have entries with widely different mean service times (up to a ratio of 50:1) as well as second-phase effects, these errors are acceptable, although we could wish they were smaller.

The second set of experiments was run on the system of Figure 3, which has second-level service and 'interlocking' effects. The approximation was compared to simulation results. Table 2 shows 10 cases with four processors and a variety of parameter values, while Table 3 shows the same 10 cases with just two processors. In carrying out the processor transformation described in Section 3, each processor with co-allocated tasks is represented by an extra task, however the parameters shown in Table 3 are the values before the transformation. Error magnitudes are similar to those described earlier.

Results against Lab Measurements

The third experiment is a comparison to measurement data for a multiprocessor implementation of the data-server system shown in Figure 2. The client tasks are intended to represent real-time tasks such as robot controllers or communications processors, accessing and updating parameters stored in the data servers (such as workstep data, or connection and route information). The data servers store all the information in main memory for speed, so there are no disk servers in the model. Data is stored in a binary tree in each server, and this tree is searched for updates and retrievals. The implementation was made with a real-time kernel which carries out rendezvous with messages, on a 12-processor VME bus computer. The execution times shown in Figure 2 are the measured values for s_{ep}, in terms of a clock in the measurement subsystem (one clock unit is 50.5 micro sec.), and the message parameters on the arcs are those that were used in the experiments. Approximately 6000 responses were gathered for each value of x, the client task execution time. The model used the independent phase calculation (12) and (13) for MRL. The performance results are shown in Table 4. The smaller values of x nearly saturate the servers, and at these values the model underestimates the throughputs by nearly 13%; for larger x the accuracy is better.

Results with Communications Delays

In a network there are extra delays associated with transmitting messages between nodes. Part of this extra delay is extra processing at the source or destination, which must be

Table 2: Throughput Results for the Example of Figure 3, by the SRN Algorithm, with Four Processors (tasks 1, 3 and 4 each on its own processor, and task 2 co-allocated with task 5).

Case	Parameters (Differences from the values shown in Figure 3)	Throughput (/sec.) by simulation (with 95% conf. int.)	Throughput (/sec.) approx.	% Error
A1	(Parameters as in Figure 3)	$\lambda_1 = 0.0256 \pm .0003$	0.0255	-0.39
		$\lambda_2 = 0.0146 \pm .0002$	0.0133	-8.90
A2	$s_{52} = 1.$, $s_{61} = 2.$, $s_{62} = 1$	$\lambda_1 = 0.0231 \pm .0002$	0.0225	-2.60
		$\lambda_2 = 0.0151 \pm .0001$	0.0130	-13.91
A3	$s_{51} = 1$, $s_{52} = 1$	$\lambda_1 = 0.0230 \pm .0002$	0.0232	0.87
		$\lambda_2 = 0.0148 \pm .0002$	0.0128	-13.51
A4	$s_{31} = 0.5$, $s_{41} = 0.6$	$\lambda_1 = 0.0254 \pm .0003$	0.0254	0.0
		$\lambda_2 = 0.0149 \pm .0002$	0.0134	-10.07
A5	$y_{251} = 1$, $y_{262} = 1.5$	$\lambda_1 = 0.0262 \pm .0002$	0.0258	-1.53
		$\lambda_2 = 0.0206 \pm .0002$	0.0184	-10.68
A6	$s_{61} = 0.5$, $s_{62} = 1$	$\lambda_1 = 0.0468 \pm .0002$	0.0482	2.99
		$\lambda_2 = 0.0305 \pm .0003$	0.0267	-12.46
A7	arrivals at entry 2	$\lambda_1 = 0.324 \pm .0005$	0.0297	-8.33
	$\lambda_{02} = 0.01$/sec. and as A1	$\lambda_2 = 0.0100 \pm .0002$	0.0100	0.00
A8	arrivals $\lambda_{02} = 0.01$/sec.	$\lambda_1 = 0.0723 \pm .0006$	0.0700	-3.18
	and as A6	$\lambda_2 = 0.0100 \pm .0002$	0.0100	0.00
A9	$s_{31} = 1$, $s_{32} = 2$, $s_{41} = 1$, $s_{42} = 1$	$\lambda_1 = 0.0679 \pm .0004$	0.0724	6.63
	$s_{61} = 0.3$, $s_{62} = 0.5$, $s_{71} = 0.3$, $s_{72} = 0.7$	$\lambda_2 = 0.0614 \pm .0003$	0.0518	-15.64
A10	$s_{61} = s_{71} = 0.5$, $s_{62} = s_{72} = 0.25$	$\lambda_1 = 0.0747 \pm .0004$	0.0779	4.28
		$\lambda_2 = 0.0631 \pm .0003$	0.0529	-16.16

included in the task service times, and part is delay "in the network", represented by the mean delay parameter δ defined in Section 3. This delay is an approximation representing front end network adapter board processing, local network delays for transmission, token circulation or retransmission, and forwarding delays between parts of larger networks. Distributed client-server applications are commonly on local networks where δ may be quite small, significantly less than other delays such as the higher-layer protocol processing.

Various values of δ up to 0.1, or 10% of the baseline service time values, were used with the first example in the "second set" of experiments mentioned above (and described in Tables 2 and 3). An exponential delay of mean δ was introduced into the exact model, which was solved by a Markov analysis. The errors of approximation were virtually unchanged, as shown in Table 5. When δ became very large however, accuracy did deteriorate to error values above 20% when $\delta = 10.0$ sec., which is ten times the baseline service time.

Table 3: Throughput Results for the Example of Figure 3, by the SRN Algorithm, with Two Processors. (Processor 1 has tasks 1, 3, 4 and processor 2 has tasks 2 and 5).

Case	Parameters (Differences from the values shown in Figure 3)	Throughput (/sec.)		% Error
		by simulation (with 95% conf. int.)	approx.	
B1	(Parameters as in Figure 3)	$\lambda_1 = 0.0247 \pm .0003$	0.0246	0.40
		$\lambda_2 = 0.0149 \pm .0002$	0.0132	-11.41
B2	$s_{52} = 1.$, $s_{61} = 2.$, $s_{62} = 1$	$\lambda_1 = 0.022 \pm .0002$	0.0213	-3.18
		$\lambda_2 = 0.0151 \pm .0002$	0.0129	-14.57
B3	$s_{51} = 1$, $s_{52} = 1$	$\lambda_1 = 0.215 \pm .0002$	0.0219	1.86
		$\lambda_2 = 0.0151 \pm .0002$	0.0128	-15.23
B4	$s_{31} = 0.5$, $s_{41} = 0.6$	$\lambda_1 = 0.0245 \pm .0002$	0.0244	-0.41
		$\lambda_2 = 0.0150 \pm .0002$	0.0133	-11.33
B5	$y_{251} = 1$, $y_{262} = 1.5$	$\lambda_1 = 0.0251 \pm .0003$	0.0249	-0.80
		$\lambda_2 = 0.0208 \pm .0002$	0.0184	-11.54
B6	$s_{61} = 0.5$, $s_{62} = 1$	$\lambda_1 = 0.0433 \pm .0003$	0.0412	-4.85
		$\lambda_2 = 0.0297 \pm .0002$	0.0262	-11.78
B7	arrivals at entry 2	$\lambda_1 = 0.0312 \pm .0004$	0.0284	-8.97
	$\lambda_{02} = 0.01$/sec. and as B1	$\lambda_2 = 0.0100 \pm .0001$	0.0100	0.00
B8	arrivals $\lambda_{02} = 0.01$/sec.	$\lambda_1 = 0.0657 \pm .0004$	0.0566	-13.85
	and as B6	$\lambda_2 = 0.0100 \pm .0001$	0.0100	0.00
B9	$s_{31} = 1$, $s_{32} = 2$, $s_{41} = 1$, $s_{42} = 1$	$\lambda_1 = 0.0589 \pm .0004$	0.0521	-11.54
	$s_{61} = 0.3$, $s_{62} = 0.5$, $s_{71} = 0.3$, $s_{72} = 0.7$	$\lambda_2 = 0.0499 \pm .0001$	0.0480	-3.81
B10	$s_{61} = s_{71} = 0.5$, $s_{62} = s_{72} = 0.25$	$\lambda_1 = 0.0648 \pm .0004$	0.0536	-17.28
		$\lambda_2 = 0.0542 \pm .0004$	0.0472	-12.92

Table 4: Comparison of model calculations to measurements on a testbed implementation of the data server example (Figure 2) (Parameters are as in the Figure, plus coefficients of variation for ServA and ServB of [0.52, 0.14] for GET, [0.04,0.65] for UPDATE.) (One tick = 50.5 microsec.)

Client task execution time x (ticks)	Throughput (each client) (measured) (ticks^{-1})	(model) (ticks^{-1})	% Error
59.5	0.0072	0.0063	12.5
100.6	0.0062	0.0054	12.9
142.6	0.0051	0.0047	7.8
183.6	0.0043	0.0041	4.6
266.7	0.0032	0.0031	3.1

Table 5: Throughput Errors with Communications Delays (for the example of Figure 3, with parameters corresponding to case A1 of Table 2 and case B1 of Table 3).

Case	Communications Delay (sec.)	Throughput λ_1 of Task 1 ($\sec^{-1}$) Exact	Approx	% Error	Throughput λ_2 of Task 2 ($\sec^{-1}$) Exact	Approx.	% Error
A1	0.00	0.025413	0.025512	0.39	0.014851	0.01329	-10.51
	0.01	0.025347	0.025452	0.41	0.01489	0.013287	-10.77
	0.02	0.025298	0.025392	0.37	0.014916	0.013284	-10.94
	0.05	0.025184	0.02521	0.10	0.014967	0.013273	-11.32
	0.10	0.025026	0.024903	-0.49	0.015022	0.013253	-11.78
B1	0.00	0.024493	0.024589	0.39	0.014931	0.013235	-11.36
	0.01	0.024454	0.02454	0.35	0.014954	0.013226	-11.56
	0.02	0.024426	0.02449	0.26	0.014964	0.013216	-11.68
	0.05	0.024372	0.024343	-0.12	0.014972	0.013187	-11.92
	0.10	0.024312	0.024095	-0.89	0.014956	0.013137	-12.16

6 Priority Queueing

Priorities have been introduced into the SRN model as priorities between the entries of a task. This is flexible and powerful. For example if priorities apply between tasks which share a processor, the priorities are modelled at the corresponding entries of the processor pseudo-task. A task's priority at the processor can also be made to depend on the entry, which can be used to model dynamic priorities and priority inheritance. Finally a priority discipline can be imposed between entries of an ordinary software task, giving priority to the messages coming to certain entries. The notation $e \succ d$ is used for 'entry e is of higher priority than entry d', or $e \succeq d$ for 'higher or equal priority'.

For non pre-emptive priority queueing (denoted NP) the waiting for tasks present at the arrival-instant will be denoted by w^*_{ab}. It is defined by the same sum as (8) for w^{FIFO}_{ab}, except the sum of the Prob$\{B\}$ terms is restricted to $d \succeq b$. There is also waiting w^+_{ab} for tasks that arrive later but are served first because $d \succ b$. Define:

$\lambda^{(ab)}_{cd}$ = the mean arrival rate of messages from entry c to entry d while entry a is blocked at b, as determined in Appendix A.

$U^{(ab)} = \sum_{c\in\mathcal{E}^*}\sum_{d\in\mathcal{S}(b),d\succ b}\lambda^{(ab)}_{cd}X_d$
= delay introduced by new arrivals to $d \succ b$, for each unit of time during which a is blocked at b.

We will adopt the arguments used for priorities in [18] (adapted in turn from the MVA Priority Approximation [27]), to obtain the queueing time to be

$$w^{NP}_{ab} = w^*_{ab} + w^+_{ab} = w^*_{ab} + w^{NP}_{ab}U^{(ab)}$$

$$= w^*_{ab}/(1 - U^{(ab)}) \qquad (21)$$

and the total delay per message is:

$$t_{ab}^{NP} = x_{b1} + w_{ab}^{NP} \ .$$

For pre-emptive priority (denoted PP) the waiting for tasks present at the arrival-instant is denoted by w_{ab}^{**}. w_{ab}^{**} is found using (8) but with the sums for both $InService$ and $InQueue$ restricted to $d \succ b$. Then the MVA Priority Approximation gives:

$$\begin{aligned} t_{ab}^{PP} &= w_{ab}^{**} + x_{b1} + t_{ab}^{PP} U^{(ab)} \\ &= (w_{ab}^{**} + x_{b1})/(1 - U^{(ab)}) \ . \end{aligned} \tag{22}$$

Priorities: Results and Accuracy

The priority algorithm was evaluated against simulations on a set of 41 cases with pre-emptive priority at processors, which seems to be the case of greatest practical interest. The results were mixed, with substantial errors in some cases, which has also been the experience of other MVA calculations for prioritiy queues. 16 cases had mean throughput errors which were all less than 10%, while there were 17 cases with some errors between 10 and 20%, and 8 cases with still larger errors. Some cases had low-priority tasks with small throughputs which were very substantially underestimated. Better results were obtained with the more elaborate approximation described in ([18], Eq. (18)), which however was not included here partly for reasons of space and partly for uniformity of treatment.

7 Fast Coupling Heuristic

There exist extreme cases combining very unbalanced entry service time and also unbalanced client delays, in which the algorithm as described above is deficient. A classic example has two client tasks with entries 1 and 2 and a single server task with entries 3 and 4. Client 1 with service time s_{12} sends requests only to entry 3, with service time s_{31} and no second phase; client 2 sends only to entry 4 with corresponding times s_{22} and s_{41}, both much longer than for client 1. Here client 1 is said to have 'fast coupling' with the server. Values $s_{12} = s_{31} = 1$ and $s_{22} = s_{41} = 100$ will illustrate the problem. The exact solution by Markov Chain methods gives $\lambda_1 = 0.253$, $\lambda_2 = 0.0049$.

This problem also arises in MVA for queueing networks, in particular in Reiser's approximation for FIFO queues with interclass service time variability in [28]. Because there are no second phases or included service intervals in this model, Reiser's algorithm itself can be applied directly. It uses three steps. The first two calculate the server utilization with only task 1, which gives 0.5, and with only task 2, which again gives 0.5. The third step computes the waiting times for the full system. Using the notation of this paper within Reiser's algorithm, they are found to be:

$$w_{13} = V(1,3,2,4)(0.5) = 0.5 s_{41} = 50$$

$w_{24} = V(2,4,1,3)(0.5) = 0.5 s_{31} = 0.5$

which makes $\lambda_1 = (s_{12} + s_{31} + w_{13})^{-1} = 0.0192$ (compared to an exact value of 0.253) and $\lambda_2 = (s_{22} + w_{24} + s_{41})^{-1} = 0.00499$ (which is almost the exact value). The error for task 1, the 'quick' task, is enormous.

Reflection shows that Reiser's analysis ignores the fact that task 1 with fast coupling returns to the server many times during a single absence of task 2. Because task 2 is away from the server for so long, the interference effect on task 1 is greatly reduced.

The following simple and novel heuristic correction to Reiser's algorithm was found to be quite effective on these cases. It is based on the fact that interference in a FIFO queue is limited by relative arrival rates, by the following bound:

$$\text{Prob}\{\text{entry } a \text{ finds task } k \text{ at task } j\} \leq \lambda_{kj}^{T\to T,(ab)} / \lambda_{aj}^{E\to T,(ab)}$$

where $\lambda_{kj}^{T\to T,(ab)} = \sum_{e\in\mathcal{E}(k),d\in\mathcal{E}(j)} \lambda_{ed}^{(ab)}$ is the task to task messaging rate from task k to task j seen by a arriving at b, and $\lambda_{aj}^{E\to T,(ab)} = \sum_{d\in\mathcal{E}(j)} \lambda_{ad}^{(ab)}$ is an entry-to-task rate. Essentially this bound says, "if k goes to d less often, then a must find it there less often, either in service or in the queue". This bound has also been used in [23] to find throughput bounds for rendezvous networks. If all service times are phase 1 and equal, as in FIFO queues in product-form queueing networks, this inequality is automatically satisfied. In our case we can express it (with a little re-arrangement) as

$$\tau_{abk} = [\sum_{c\in\mathcal{E}(k),d\in\mathcal{E}(j)} \text{Prob}\{InQueue(a,b,c,d)\} + \sum_{p} \text{Prob}\{InService(a,b,c,d,p)\}] \frac{\lambda_{aj}^{E\to T,(ab)}}{\lambda_{kj}^{T\to T,(ab)}} \leq 1. \quad (23)$$

To force $\tau_{abk} \leq 1$, an adjustment was inserted after all the Prob$\{InService\}$ and Prob$\{InQueue\}$ values were calculated as described in step 4(a) of the algorithm. Just before computing V at the end of step 4(a), the adjustment was made by

- computing τ_{abk} according to (23).
- if $\tau_{abk} > 1$, divide each Prob$\{InService\}$ and Prob$\{InQueue\}$ by τ_{abk}, to force τ_{abk} (after the modification) to be unity.

On the two-client example which began this section, this approach gives the values $\lambda_1 = 0.251$ and $\lambda_2 = 0.00499$, which are almost exact. For practical calculation, to avoid the sharp nonlinearity at $\tau = 1$ which sometimes made the iteration unstable, the probabilities were multiplied by a smooth function $\text{sat}(\tau) = 1/(1+\tau^n)^{1/n}$ instead of dividing by τ, with a large value of n.

8 Conclusions

The Stochastic Rendezvous Network model incorporates the multiphase behavior of software, with synchronous (in-rendezvous) and asynchronous (post-rendezvous) phases. The model has been extended here to handle tasks with distinct entries. A systematic study of arrival-instant cases has been described to give contention probabilities for a Mean Value Analysis algorithm. Earlier models involving processor contention and priorities have been unified with the treatment of entries.

The model is solved more quickly (often hundreds of times more quickly) than a Timed Petri Net or a simulation, at the cost of an approximation error which (at least for non-priority systems) is usually less than 15%. Results consist of mean throughputs λ_e for entries, λ_i for tasks, utilizations U_i for tasks, and response delays w_{ed} between entries. For moderate sized systems the space complexity and the time complexity per iteration are predominantly linear in the number of entries for small and middle-sized systems, although there is a quadratic term with a small coefficient that will eventually become important in large models.

The algorithm tends to underestimate the throughput, although some throughputs are overestimated. The tendency may be due to assumptions that some distributions (e.g. waiting times) are exponential, or that arrivals are independent.

The advantages of the SRN model (compared to Petri Nets or Markov Chains) are: its high level of abstraction, its explicit treatment of rendezvous delays, the uniform treatment of hardware and software contention, and a fast iterative approximate solution technique. The present version of the model assumes exponential random execution times, random messaging, independent rendezvous, tasks which always accept messages on any entry, and FIFO, non-preemptive or preemptive-resume priority queueing.

It is assumed here that every task i is ready to accept requests on all entries at all times. If this is not the case, if the entry may be 'closed' on a guard condition, then throughput can be affected. In practice there are many ways that guards can be controlled. It is easy to model specific examples, such as guards to control input and output from a bounded buffer, but we know no model for the more general situation.

One kind of guard that can be modelled here is a guard which is used to provide effective priorities between entries of a task. Here the priority is identified directly in the model as a non-preemptive priority discipline for the entries of the task.

Computational Tools

The implementation of the algorithm is available by itself, and also within a prototype software tool called TimeBench, and in a spreadsheet-style tool for PCs, which displays tasks in a table by processor allocation and priority.

Acknowledgements

This research was supported by grants from Bell Canada and from the Natural Sciences and Engineering Research Council of Canada, under their operating grants program, as well as by the Telecommunications Research Institute of Ontario through its project on Telecom Software Methods.

We gratefully acknowledge the help of the anonymous referees. Greg Franks programmed the algorithms and made many helpful comments. Jerzy Miernyk contributed to earlier versions of this work and his influence is still present.

References

[1] D. Bowen, "Open Distributed Processing," *Computer Networks and ISDN Systems*, Vol. 23, pp. 195–201, 1991.

[2] C.A.R. Hoare, *Communicating Sequential Processes.* London, England:Prentice-Hall International, UK, Ltd., 1985.

[3] C.M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks," *Performance Evaluation*, Vol. 9, pp. 143–160, 1989.

[4] C.M. Woodside and G.M. Yee, "Traffic Relationships in Networks of Tasks," In *Proc. IEEE Infocom '89*, pp. 1040–1049, April 1989.

[5] C.M. Woodside, J.E. Neilson, J.W. Miernik, D.C. Petriu, and R. Constantin, "Performance of Concurrent Rendezvous Systems with Complex Pipeline Structures," In *Proc. 4th International Conference in Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 361–378, September 1988.

[6] C.M. Woodside, E. Neron, E.D.S. Ho, and B. Mondoux, "An "Active-Server" Model for the Performance of Parallel Programs Written Using Rendezvous," *J. Systems and Software*, 1986.

[7] G. Ciardo and K.S. Trivedi, "Solution of Large GSPN Models," In *Proc. First Intl. Conf. on the Numerical Solution of Markov Chains*, pp. 603–626, North Carolina State University, Raleigh, North Carolina, January 1990.

[8] D.C. Petriu and C.M. Woodside, "Approximate MVA from Markov Model of Software Client/Server Systems," In *Third IEEE Symposium on Parallel and Distributed Processing*, pp. 322–329, Dallas, Texas, December 1991.

[9] DOD, *Reference Manual for the Ada Programming Language ANSI/MIL-STD 1815.* United States Department of Defense, January 1983.

[10] D.R. Cheriton, "The V kernel: A software base for distributed systems," *IEEE Software*, Vol. 1, No. 2, pp. 19–42, April 1984.

[11] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.G. Sevcik, *Quantitative System Performance.* Englewood Cliffs, N.J. 07632:Prentice-Hall, Inc., 1984.

[12] L. Flatto and S. Hahn, "Two Parallel Queues Created by Arrivals with Two Demands," *SIAM on Appl. Math.*, Vol. 44, No. 5, October 1984.

[13] P. Goyer, P. Momtahan, and B. Selic "A Synchronization Service for Locally Distributed Applications," In *Distributed Processing*, Elsevier Science Publishers B.V., North-Holland, 1988.

[14] P. Heidelberger and K.S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency," *IEEE Transactions on Computers*, Vol. C-32, No. 1, pp. 73–82, January 1983.

[15] J.A. Rolia, "Performance Estimates For Systems With Software Servers: The Lazy Boss Method," *Proceedings VIII SCCC International Conference on Computer Science*, July 1988.

[16] J.A. Rolia *Predicting the Performance of Software Systems* PhD thesis, University of Toronto, January 1992.

[17] J.L. Peterson, *Petri Net Theory in the Modeling of Systems.* Englewood Cliffs, N.J.:Prentice-Hall, 1981.

[18] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Performance of Stochastic Rendezvous Networks with Priority Tasks," In *Proc. Int. Seminar on Performance Distributed and Parallel Systems*, pp. 501–515, December 1988.

[19] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Throughput of Stochastic Rendezvous Networks with Caller-Specific Service and Processor Contention," In *Proc. IEEE Infocom '88*, pp. 1040–1049, March 1988.

[20] M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 12, pp. 1297–1310, December 1987.

[21] M.A. Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri Nets for the performance evaluation of multiprocessor systems," *ACM Trans. on Computer Systems*, Vol. 2, No. 2, May 1984.

[22] M.J. Fontenot, "Software Congestion, Mobile Servers, and the Hyperbolic Model," *IEEE Trans. Software Eng.*, Vol. 15, pp. 947–962, 1989.

[23] S. Majumdar, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Performance Bounds for Concurrent Software with Rendezvous," *Performance Evaluation*, Vol. 13, pp. 207–236, 1991.

[24] R. Milner "A Calculus of Communicating Systems," Lecture Notes in Computer Science 92, 1980.

[25] O.S.I., "LOTOS: A Formal Description Technique Based on Temporal Ordering of Observational Behavior," *International Organization for Standardization, Information Processing Systems, Open Systems Interconnection*, Vol. ISO8807, 1988.

[26] D.C. Petriu, "Approximate Mean Value Analysis of Client-Server Systems with Multi-class Requests," In *Proc. of ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 77–86, Nashville, Tennessee, May 1994.

[27] R.M. Bryant, A.E. Krzesinski, M.S. Lakshmi, and K.M. Chandy, "The MVA Priority Approximation," *ACM Transactions on Computer Systems*, Vol. 2, No. 4, November 1984.

[28] M. Reiser, "A Queueing Analysis of Computer Communication Networks with Window Flow Control," *IEEE Trans. on Communications*, Vol. COM-27, pp. 1199–1209, August 1979.

[29] S.C. Agrawal and J.P. Buzen, "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems," *ACM Trans. on Computer Systems*, Vol. 1, No. 2, pp. 116–143, May 1983.

[30] S.S. Lavenberg, *Computer Performance Modeling Handbook*. Academic Press, 1980.

[31] S.S. Lavenberg and M. Reiser, "Stationary state probabilities at arrival instants for closed queueing networks with multiple types of customers," *Journal of Appl.Prob.*, Vol. 17, pp. 1048–1061, 1980.

[32] J. Zahorjan, E.D. Lazowska, and D.L. Eager, "Spinning versus Blocking in Parallel Systems with Uncertainty," In *International Seminar on Performance of Distributed and Parallel Systems*, December 1988.

[33] E. de Souza e Silva and R.R. Muntz, "Approximate Solutions for a Class of Non-Product Form Queueing Network Models," *Performance Evaluation*, Vol. 7, pp. 221–242, 1987.

[34] E. de Souza e Silva, S.S. Lavenberg, and R.R. Muntz "A Perspective on Iterative Methods for the Approximate Analysis of Closed Queueing Networks," In G. Iazeolla, P.J. Courtois, and A Hordijk, editors, *Mathematical Computer Performance and Reliability*, Elsevier Science Publishers B.V., North-Holland, 1984.

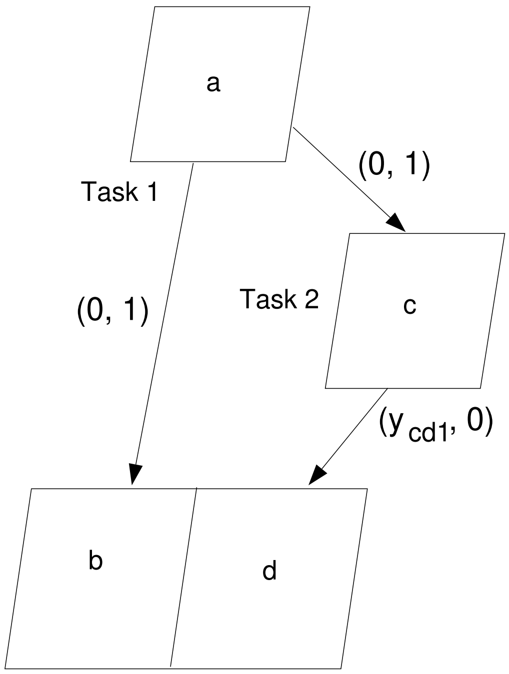


Figure 6: Task Interlocking Example

Appendix A. Interlocked Flows

Contention at arrival instants or while a task is blocked is sometimes reduced by dependencies among the message flows which can be traced back to a common source. When the common source stimulates one flow by sending a message, it becomes blocked in a rendezvous and is thus unable to stimulate other flow components. In such a case an arrival which is part of one flow will see the other flow components as being reduced, or cut off. The flow components which are cut off are "locked out" by the blocking of the common source, and the two flows are described as interlocked.

Interlocking is accounted for by defining a modified effective flow rate for competing messages:

$\lambda_{cd}^{(ab)}$ = the effective competing flow, (instead of λ_{cd}), seen by an arrival from a to b, when $d \in \mathcal{S}(b)$.

For example, if $c \in \mathcal{S}(a)$, then automatically $\lambda_{cd}^{(ab)} = 0$, since $T(a)$ can only send one message at a time.

Two-layer interlocking is illustrated in Figure 6. Task a is the upper layer, and task c is the lower layer. When task 1 sends to entry 3, task 2 is either idle, or executing on its own, but it never has a message for entry d (because $y_{cd2} = 0$). Task 2 does send messages to entry d but only when task 1 is blocked at task 2 waiting for a reply, so in these cases task 1 never competes. Thus although there are two flows of messages to task 3 no message ever finds task 3 busy in phase 1 (however, they may find it busy in phase 2).

A partial interlocking in which only a fraction of the competing flow is locked out, is a more general situation. For example if $y_{cd2} = y_{cd1}$ then the messages from task 1 would see only half the flow from task 2 locked out.

Two-layer interlocking effects take two forms. From the point of view of entry a sending to b the flow from c to d is *send interlocked* if a or some sibling of a sends to c. This is the situation in Figure 6. Define the *interlocked* rate of c, relative to a, as

$$\lambda^{IL}_{c;a} = \sum_{a' \in \mathcal{S}(a)} \lambda_{a'c} \ . \tag{24}$$

This component of λ_c is automatically excluded when entry a is sending a message anywhere else but c. Thus for send interlocking,

$$\lambda^{(ab)}_{cd} = \lambda_{cd} - \lambda^{IL}_{c;a} y_{cd1} \ . \tag{25}$$

On the other hand, from the point of view of a sending to b, the flow from c to d is *receive interlocked* if c or some sibling of c sends to a. (This would be the situation in Figure 6 if a and c were interchanged). The fraction of messages sent from a in phase 1 to b while a is responding to a message from $T(c)$, is $\lambda^{IL}_{a;c}\ y_{ab1}/\lambda_{ab}$, and this fraction is assumed to see no competition from c, while the remainder sees independent behaviour. Thus for receive interlocking,

$$\lambda^{(ab)}_{cd} = \lambda_{cd}(1 - \lambda^{IL}_{a;c} y_{ab1}/\lambda_{ab}) \ . \tag{26}$$

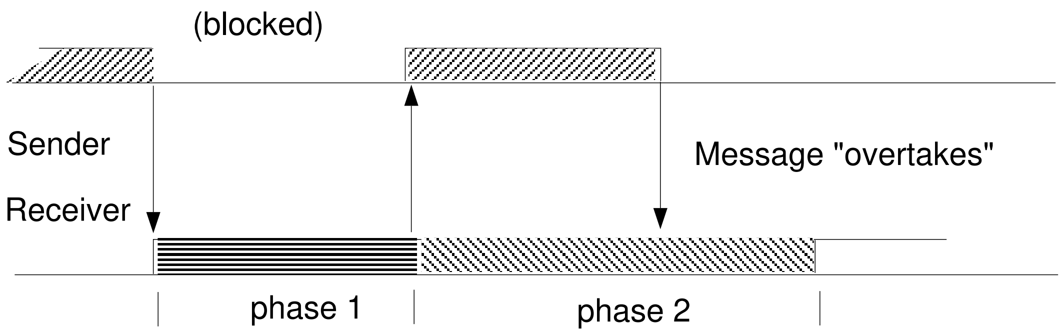
All flow rates λ_{cd} used to compute Prob$\{InService(a, b, c, d, p)\}$ with $p = 1$ were replaced by $\lambda^{(ab)}_{cd}$, to incorporate two-layer interlocking effects. Send and receive interlock cannot occur simultaneously as it would imply a phase 1 cycle in the task communications graph.

The calculation follows these steps:

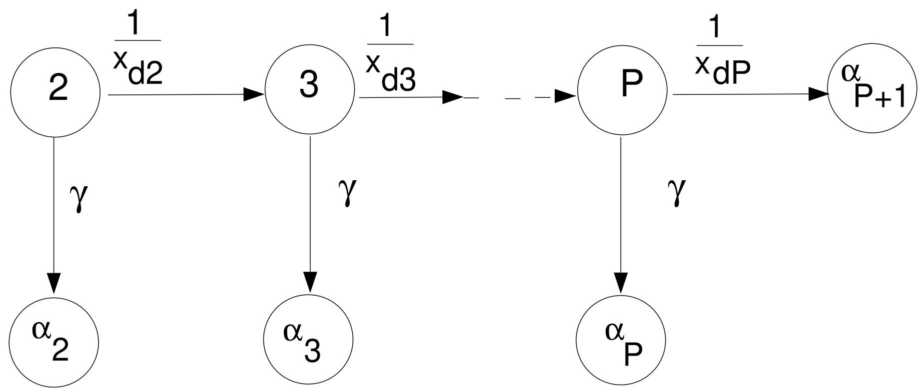
- if c is send interlocked with a, $\lambda^{(ab)}_{cd}$ is found from (24), (25).
- if c is receive interlocked with a, $\lambda^{(ab)}_{cd}$ is found from (24), (26).
- else, $\lambda^{(ab)}_{cd} = \lambda_{cd}$.

The effects included here are only the two-level effects, which appear to be the most common. More complex dependencies exist and a more complete approach would have to include them.

The same interlocking constraints continue to operate on the flows which may occur after arrival while entry a is blocked in the queue at b. Therefore these expressions for the competing flows are used in the priority queueing calculations, as well as in the arrival-instant analysis.



(a) An Overtaking Event



(b) States of a Transient Markov Chain for Overtaking

Figure 7: Overtaking

Appendix B. Overtaking

An analysis of the probability of a message from task i "overtaking" at task j begins at the moment of the previous reply from task j to task i. Task i then goes through a series of actions taking an average time γ^{-1} sec, up to its next message to task j, called its *return time* while task j completes phase 2 of entry d, and possibly later phases. Considering only phase 2, the two delays are assumed for simplicity to be exponentially distributed so that the probability that the new message comes before the end of phase 2 is

$$\text{Prob (overtaking)} = \gamma/(\gamma + x_{d2}^{-1})$$

where the return time γ^{-1} is given by the average period of messages from i to j, minus the average waiting for one message, minus the phase-1 service that started the present cycle:

$$\gamma^{-1} = (\lambda_{ij}^{T\to T})^{-1} - w_{ij}^{T\to T} - x_{d1}$$

$$w_{ij}^{T\to T} = \sum_{e\in\mathcal{E}(i),d\in\mathcal{E}(j)} \lambda_{ed} w_{ed}/\lambda_{ij}^{T\to T} \ .$$

If there are P_d phases instead of 2 we can consider the transient Markov chain shown in Figure 7, in which the transitions of rate γ lead to "overtaking in phase p", labelled as event α_p, and the other transitions show phase completions before the return of task i. The chain ends up in state α_{P_d+1}, meaning no overtaking, or in state α_p for $2 \le p \le P_d$, meaning overtaking in phase p, with

$$\text{Prob}(\alpha_p) = \frac{\gamma}{\gamma + 1/x_{dp}} \Pi_{p'=2}^{p-1} \frac{1/x_{dp'}}{\gamma + 1/x_{dp'}} \qquad p = 2, \ldots, P_d + 1 \ . \tag{27}$$

To break overtaking down according to which entries c and d are involved, their relative throughputs give the probability as $\lambda_{cd}/\lambda_{ij}^{T\to T}$. Thus:

$$\text{Prob}\{InService(a,b,c,d,p)\} = \frac{\lambda_{cd}}{\lambda_{ij}^{T\to T}} \text{Prob}(\alpha_p)$$

for cases with $c \in \mathcal{S}(a)$, $d \in \mathcal{S}(b)$, $p > 1$, and using $\text{Prob}(\alpha_p)$ given by (27).