Modeling A Scalable High-Speed Interconnect with Stochastic Petri Nets

Gianfranco Ciardo*

Department of Computer Science

College of William and Mary

Williamsburg, VA 23187-8795, USA

ciardo@cs.wm.edu

Ludmila Cherkasova, Vadim Kotov, and Tomas Rokicki

Hewlett-Packard Labs

1501 Page Mill Road

Palo Alto, CA 94303, USA

{cherkasova,kotov,rokicki}@hpl.hp.com

Abstract

This paper presents an approach to using Stochastic Petri nets to model large-scale concurrent systems, in our case, a scalable computer interconnect. We show how Stochastic Petri net models can exploit the symmetry of the system to construct a tractable, but approximate, analytic model, and that they can yield results very close to those of a detailed simulation model, with much less computational effort.

Key Words: Performance analysis, numerical analysis, Markovian models, stochastic Petri nets, approximate models, case study, scalable interconnect.

[*] This research was initiated while G. Ciardo was visiting Hewlett-Packard Labs.

Contents

1 Introduction

In this paper, we present techniques for the modeling and analysis of large-scale concurrent systems using Petri nets. There are two important but conflicting requirements in this sort of analysis. First, the model should be detailed enough to include those system features that have a significant impact on performance. Second, the model should be simple enough to be tractable. Constructing a single model of a complex system does not lead to completely trustworthy results; modeling is subject to the same sorts of errors and inaccuracies as programming in general, yet the results are often not as easy to check. We therefore use different types of models to deal with different aspects and stages of the system analysis. The common feature of these models is that they are based on Petri nets.

We present as a case study the net modeling of a *scalable interconnect* for its performance evaluation and analysis. Our goal was to analyze the performance of the interconnect as a function of various parameters, including network size (which can scale up to hundreds of nodes) and the number of internal buffers on each node.

We have built and analyzed two net models of the interconnect. We used SPNP [8], based on *Stochastic Petri Nets*, to build an approximate model for a quick numerical analysis of performance. We used Design/CPN$^{\text{TM}}$ [9, 14] based on *Hierarchical Colored Petri Nets* to develop a detailed simulation model to help refine the design and identify performance bottlenecks. This model was also used to verify the accuracy and correctness of the approximate SPN model.

The original Petri net formalism is inadequate for the specification of complex large-scale systems, especially systems for which the number of interacting components depend on the input parameters. We instead use two higher-level formalisms based on Petri nets.

High level Petri nets, in particular Colored Petri nets, provide for the specification of large-scale systems with colored tokens, that allow the folding of the system description into a very compact form. The Colored Petri nets used in Design/CPN are a graphical programming language with rich specification and simulation possibilities. The colored tokens and arc expressions allow easy parameterization of the system size. Through the use of hierarchy and refinement, a series of models with different levels of detail can be easily constructed, enabling rapid prototyping and analysis.

The main difficulty when modeling complex systems with high level Petri nets is that the simulation time of these nets does not remain constant with increasing system size. For a very large number of components, simulation time becomes prohibitive.

Stochastic Petri Nets allow the quick construction of a simplified abstract model that is then numerically solved for different model parameters. This analysis is based on the exploration of all reachable states in the model, and is thus even more dependent on the system size. A straightforward SPN model of the interconnect generates exceedingly large stochastic processes for even the smallest network sizes.

The solution to this problem is to construct an approximate model that takes into the consideration some specific features of the modeled system. In our case, we exploit the fact that the interconnect has a very regular structure. We were primarily concerned with two different issues:

1) how performance scales with network size, and

2) how internal design alternatives affect performance.

These tasks are not independent of one another. Nonetheless, it is possible to split the problem into two stages. First, a simplified but scalable model to predict bottom-line performance and identify possible bottlenecks is quickly constructed. Next, a detailed model is constructed to evaluate and analyze particular design alternatives. These two stages may be iterated to verify the results and refine the analysis.

In this paper we show how Stochastic Petri nets can exploit the symmetry of the system to construct a tractable approximate model. We present this model to support our conclusion that Stochastic Petri nets can be successfully used for modeling industrial size systems.

Sections 3 and 4 of this paper present the interconnect structure and its exact SPN model. Section 5 presents our approach to build a tractable approximate SPN model. Section 6 presents the numerical results. Section 7 compares the numerical results of SPN model with simulation results of the interconnect model based on Colored Petri nets. In the conclusion, we discuss a few missing features of SPNP which might further improve the applicability of such a tool.

A brief introduction to the basic terminology is given in the Section 2. For a complete treatment of the class of SPNs used in this paper, see [5].

2 Basic SPN Notions and Denotations

A Petri net (PN) [12, 13] is a directed bipartite graph with two sets of nodes, *places* and *transitions*. *Input arcs* connect a place to a transition, *output arcs* connect a transition to a place. A multiplicity (positive integer) may label an arc. The *input* (*output*) *bag* for a transition is the bag [12] constituted by the input (output) arcs, considered with their multiplicity. Each place may contain any number of *tokens*. A *marking* is a bag representing the configuration of tokens in the places of the PN, it is the "state" of the PN.

A transition is *enabled* if its input bag is a subbag of the (current) marking. When a transition is enabled, it can *fire*, leading the PN into a different marking, obtained by subtracting its input bag from and adding its output bag to the current marking. A marking is *reachable* if it is obtained by a sequence of firings starting in the *initial marking*. The *reachability set (graph)* is the set (graph) of all the reachable markings (connected by arcs representing the transition firings). If *inhibitor arcs* are used, the enabling rules changes. A transition t is disbaled if there is an inhibitor arc with multiplicity k from place p to t, and p contains k or more tokens.

A "generalized stochastic Petri net" (GSPN) [1] is a PN where each transition has an associated *firing time*, which can be zero (*immediate transition*) or exponentially distributed with a parameter dependent on the marking (*timed transition*). If several conflicting immediate transitions are enabled in a marking, a *firing probability* is specified for each of them. A firing probability for timed transitions does not need to be specified, since we assume a *race model*: the first firing time to elapse determines which transition fires first; contemporary elapsing of firing times has probability zero.

If at least one immediate transition is enabled, the marking is said to be *vanishing*, otherwise the marking is said to be *tangible*. Since the firing time of an immediate transition is zero, a GSPN does not remain in a particular vanishing marking for any length of time. In other words, the probability of finding the GSPN in a vanishing marking is zero. A GSPN describes an underlying stochastic process, reducible to a continuous-time Markov chain (CTMC) by *eliminating the vanishing markings* [1, 3, 5].

The GSPN model can be extended by allowing phase-type distribution [10] instead of just simple exponential distributions. The underlying process is still a CTMC, but its states now describe both the marking of the PN and the phase for the firing times of the transitions, hence the size of the state space is larger than that for a similar GSPN. This is a problem, since the size of the state space is already the main obstacle to a numerical analysis. Other distributions can also be used for the firing times (e.g., constants). The term "SPN" is often used as a generic indicator that the timing behavior of the net is probabilistic, hence it is, surprisingly, more general than the term "GSPN". This is due to historical reasons, since the first definition of SPNs allowed only exponentially distributed, but not zero, firing times.

Examples of output measures obtained from the analysis of a SPN are the expected number (or probability distribution) of tokens in a given place or the throughput of a given transition, in steady-state or at a specific point in time. They correspond to reward functions on the markings or on the transitions among markings and they are easily computed after solving for the steady-state or transient probability of each marking [5].

3 System

The interconnect topology is a continuous hexagonal mesh which permits each node to communicate with its six immediate neighbors. We refer to a mesh having n nodes on each edge as an E_n mesh. The resulting total number of nodes is $N = 3n(n-1)+1$. Figure 1 shows the mesh structure for E_2, E_3, and E_4. Physically, nodes on the edge are actually connected to nodes on other edges in a wraparound fashion, so that their virtual connectivity is the same as that of internal nodes.

The distance of a node from a particular node i in an E_n mesh is at most $n-1$ "hops". Furthermore, of the $N-1 = 3n(n-1)$ nodes other than i, 6 are at distance 1, 12 are at distance 2, and so on, up to the $6(n-1)$ at distance $n-1$.

Each node is attached to a processor connected via a local bidirectional port.

In addition, each node i has six ports, numbered 0 through 5, each of them connecting it to a different neighbor, $n(i,0)$ through $n(i,5)$, respectively (see Figure 2). Each of this ports is full-duplex.

The node has a total of N_b buffers to store the packets in transit. The processor injects packets to the interconnect through the local ports when both 1) the local port is available and 2) there is an available buffer to store the packet. Otherwise, the packet waits in a queue until the required resources are freed. In-transit packets follow the same procedure. Packets are always routed through a minimal-length path towards their destination. Each time the packet arrives at a new node, the next node on a minimal path to destination is computed

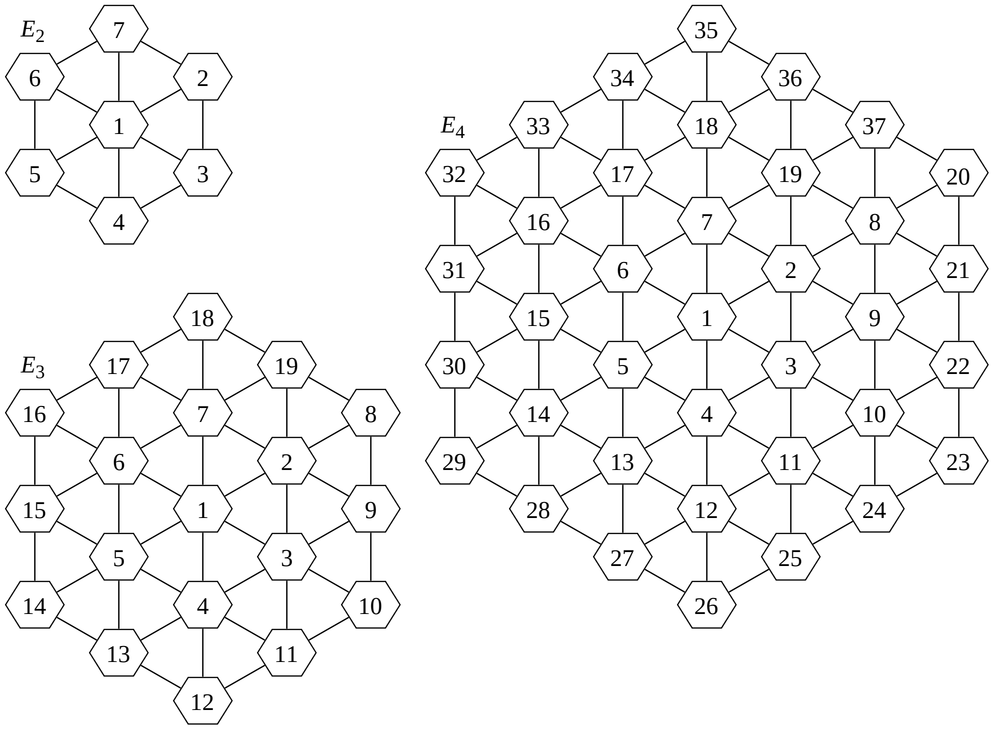


Figure 1: System structure, for $n = 2, 3$, and 4.

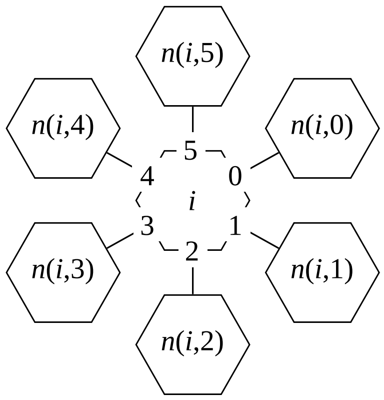


Figure 2: Neighbor nodes.

and the router attempts to forward the packet. If either the port is busy or no buffers in the next node are available, the packet waits. When the packet finally arrives to a destination node, it is ejected from the interconnect through the local processor port.

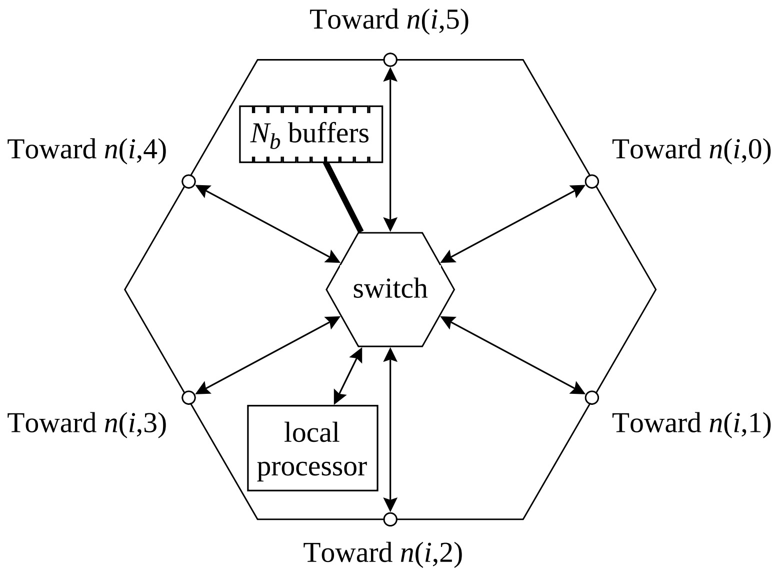


Figure 3: Node structure.

We assume packet transfer, injection, or ejection takes 720 tu (time units). A main parameter of the model is the number of buffers inside each node; for this paper, we use a default of ten.

4 Detailed model

A detailed SPN model for node i is shown in Figure 4. The transition $Generate^i$ generates the packet tokens at a given rate and puts them into the place $Wait^i$. Place EB^i contains tokens corresponding to empty buffers inside node i. The number of tokens N_b initially in this place is the total number of buffers in node i.

Firing of the immediate transition $GetBuf^i$ reserves a buffer inside the node i (if there is a token in the place EB^i) for the new packet and moves a token to a place $Sending^i$. The packet injection to node i is represented by the transition $Send^i$.

The output places $O_0^i, \ldots, O_5^i$ of the node i are the input places for the six neighbor nodes: for example, place O_3^i is the input place for node $n(i,3)$ from the node i.

For simplicity of illustration, a "probabilistic arc" is used from transition $Send^i$ to places

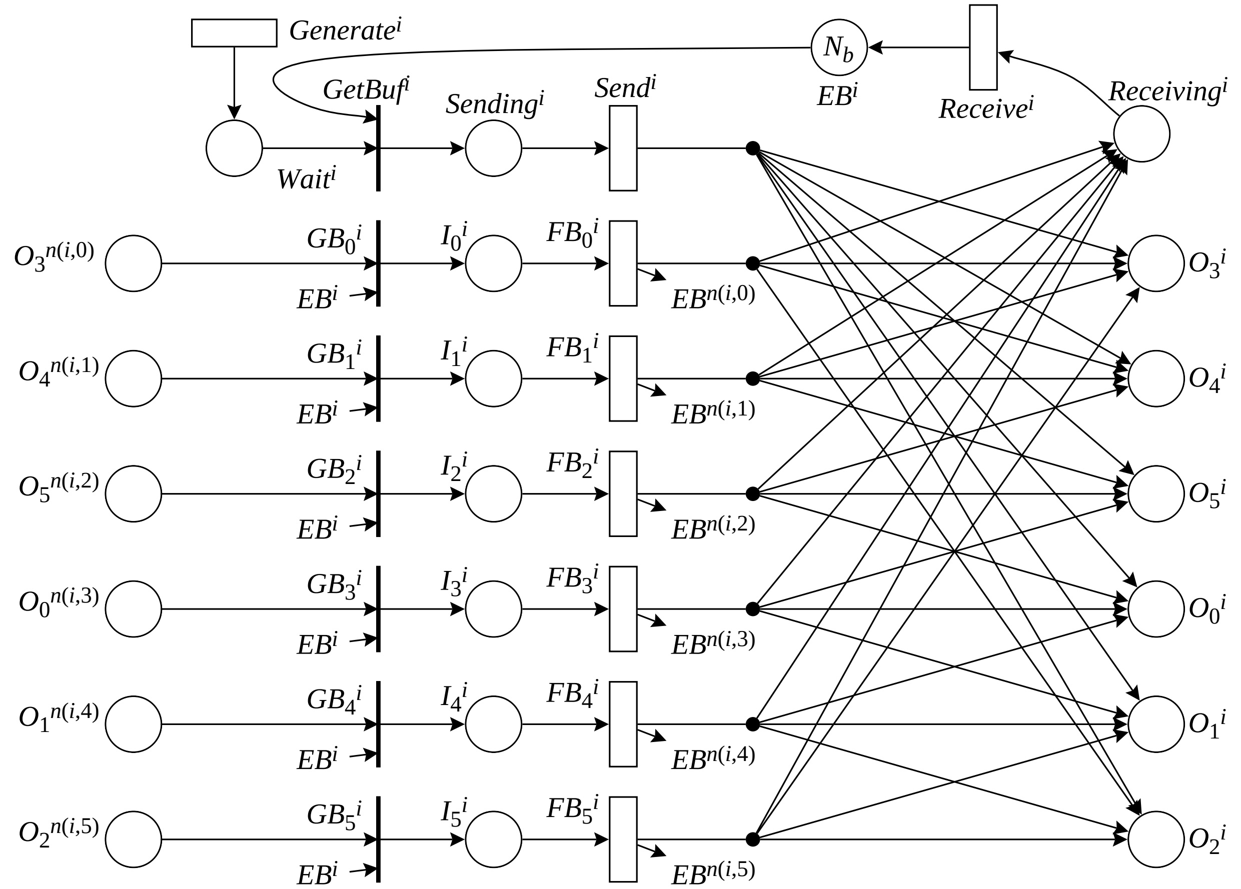


Figure 4: Detailed SPN model.

O_0^i through O_5^i. A probabilistic arc is a shorthand to denote that the token deposited by a transition, say $Send^i$, can end up in exactly one of the destination places. Formally, this behavior is be obtained with one extra place and a set of immediate transitions. For the branches from $Send^i$ to O_0^i through O_5^i, we specify a probability of 1/6, since we assume that the destination node for a packet generated at node i is uniformly chosen among the remaining nodes.

The packets arrive to each node from the local processor (through the local port) and from the six neighbor nodes (through neighbor ports). Places with a superscript $n(i,0)$ through $n(i,5)$ are in the individual SPNs representing the six neighbor nodes. For example, place $O_3^{n(i,0)}$ is the place in the SPN for the neighbor $n(i,0)$ of i.

The packet transfer from the neighbor nodes is the same as from the local processor. First of all, for each packet arriving at node i, a buffer must be reserved. Firing of immediate transition $GB^i_j, (j = 0, \ldots, 5)$ reserves the buffer inside node i if there is one available, represented by a token in place EB_i. The firing of transition FB^i_j corresponds to the packet transfer by the input port j to a node i. When the packet transfer is complete, the buffer occupied by this packet in the previous node $n(i,j)$ is released by returning a token to place $EB^{n(i,j)}$ (we omit these places for clarity).

A'probabilistic arc is used from transitions FB^i_0 through FB^i_5 to places $Receiving^i$ and O^i_0 through O^i_5; these are expanded into extra places and immediate transitions as described before. There are two possibilities:

1. If the packet is destined for the node i then it is put in a place $Receiving^i$, and ejected from the node by the local processor port (represented by the transition $Receive^i$). When the packet ejection is complete, one buffer is released by returning a token to a place EB^i.
2. If the packet is intended for a neighbor node j then it is put into the place O^i_j.

Assuming that packets are always routed only through one of the shortest paths, a packet arriving from port j can only be sent to the current node (place $Receiving^i$) or to another node to be reached through ports $(j+2)$ mod 6, $(j+3)$ mod 6, or $(j+4)$ mod 6. That is, a packet never reverses direction. The routing probabilities depend on both the source and the destination of the packet. For E_2, external packets arriving at node i always have their destination as node i, since the maximum distance between any two nodes is one. When modeling E_n, $n > 2$, we can associate the identity of the source and destination with each token representing a packet, resulting in a colored SPN with a huge state space.

Alternatively, we can remain in the uncolored domain and obtain considerable state space reduction by assigning probabilities to the four possible destinations for a packet arriving through port j:

1. the local node, i, with probability p_{local},
2. an external node reachable through port $(j+2)$ mod 6, with probability p_{side},
3. an external node reachable through port $(j+3)$ mod 6, with probability p_{center},

4. an external node reachable through port $(j+4) \bmod 6$, with probability p_{side} again, since, by symmetry, this case has the same probability as case (2) above.

This probabilistic view involves an approximation, since it is now possible to have packets follow arbitrarily large paths through the mesh. On the other hand, it is nevertheless possible to set the parameter p_{local} so that the average load on the network is correctly matched, and each port on each node has the same load. We define a "hop" to be the movement of a packet from a node to one of its neighbors, and compute the average number of hops required to send a packet from source to destination. For E_n, if node i sends to every other node with equal probability, then $6k$ potential destinations out of $3n(n-1)$ are k hops away, hence the average number of hops is

$$\sum_{k=1}^{n-1} k \cdot \frac{6k}{3n(n-1)} = \frac{2n-1}{3}$$

In particular, the average number of hops is 1 for E_2, as expected. Each incoming packet corresponds to one hop, hence a fraction

$$p_{local} = \left(\frac{2n-1}{3}\right)^{-1} = \frac{3}{2n-1}$$

of the incoming packets is directed to the local node i.

We stress that this choice for p_{local} ensures that the expected number of hops per packet and the average rate of hops over the entire mesh, $N\lambda/p_{local}$, or to an individual node, λ/p_{local}, or even to an individual port, $1/6 \cdot \lambda/p_{local}$, is the same for the exact colored model and for the approximate probabilistic model, where λ is the injection rate from each node. The only approximation lies in the probability mass function (pmf) of the number of hops for a packet. In the exact model,

$$\Pr\{\text{number of hops is } k\} = \frac{6k}{3n(n-1)},$$

while, in the approximate model,

$$\Pr\{\text{number of hops is } k\} = \frac{3}{2n-1}\left(1 - \frac{3}{2n-1}\right)^{k-1},$$

which describes a geometric distribution. The quality of approximation increases with the size n of the mesh, since the number of nodes increases quadratically in n, while the expected number of hops increases linearly in n.

Clearly, $p_{local} + p_{center} + 2p_{side} = 1$, but p_{center} and p_{side} still need to be determined. Observing Fig. 1 again, this corresponds to determining the proportion of hops of type "l", "c", and "s", defined as the first hop (i.e., a packet from the local processor on node 1 exits on the port from 1 to 2), a hop going to the center port (i.e., a packet entered on the port from 1 to 2 exits on the port from 2 to 8), and a hop going to a side port (i.e., a packet entered on the port from 1 to 2 exits on the port from 2 to 9), respectively.

Define H_n to be the expected number of hops of each type for a packet transmitted in E_n, starting from node 1 (because of the symmetric nature of the network, the choice of node 1 is arbitrary). For E_2, each packet takes exactly one hop of type l, hence $H_2 = l$. For E_3, each packet can take the following combinations of hops:

- From 1 to $\{2, 3, \ldots 7\}$: one hop of type l.
- From 1 to $\{8, 10, \ldots 18\}$: one hop of type l and one hop of type c.
- From 1 to $\{9, 11, \ldots 19\}$: one hop of type l and one hop of type s.

Assuming that each node other than 1 is a potential destination with equal probability, $H_3 = l + \frac{1}{3}c + \frac{1}{3}s$.

For larger size networks, the analysis becomes more complex. For example, in E_4, a packet with source node 1 and destination node 21 can choose among three paths: $1 \xrightarrow{l} 2 \xrightarrow{c} 8 \xrightarrow{s} 21$, $1 \xrightarrow{l} 2 \xrightarrow{s} 9 \xrightarrow{s} 21$, and $1 \xrightarrow{l} 3 \xrightarrow{s} 9 \xrightarrow{c} 21$. For the analysis, we assume that, whenever a hop to either one of two neighbors would still achieve the shortest path for the packet, the choice is performed with uniform probability (e.g., $1 \xrightarrow{l} 2$ and $1 \xrightarrow{l} 3$ have probability 1/2, and, given that $1 \xrightarrow{l} 2$ is chosen, $2 \xrightarrow{c} 8$ and $2 \xrightarrow{c} 9$ have probability 1/2). Hence, the probability of the above three paths is 1/4, 1/4, and 1/2, respectively, not simply 1/3 for each of them. By enumerating these paths and computing their probabilities, we can then obtain $H_4 = l + \frac{11}{12}c + \frac{5}{12}s$. For E_5, we obtain $H_5 = l + \frac{29}{20}c + \frac{11}{20}s$. This implies that the value of p_{center} and p_{side} depends on the size of the network. However, their relative value will not affect the average load on the mesh, hence, we could, for simplicity, use the values

$$p_{center} = \frac{1 - p_{local}}{2} = \frac{n-2}{2n-1} \qquad \text{and} \qquad p_{side} = \frac{1 - p_{local}}{4} = \frac{n-2}{2(2n-1)},$$

which coincide with the exact values for $n = 2, 3$.

5 Approximate model

By assuming that the firing times of the timed transitions for the SPN in Fig. 4 are exponentially distributed, the resulting underlying stochastic process is a continuous-time Markov chain (CTMC). In principle, we could study this CTMC to compute the performance measures of interest using software tools such as SPNP [8], but its size is too large for an exact numerical solution, even for the E_2 interconnect having only 7 nodes. In this section, we describe an approximate model based on the idea of SPN decomposition and fixed-point iteration [7]. This approximate model exploits the large amount of symmetry possessed by the interconnect and essentially describes a behavior of one node under a workload that is generated by the whole interconnect fabric. Thus the basic idea is to approximate and generate a proper amount of traffic going through one node in a network of a particular size.

We will construct the approximate SPN model from the following four SPN subnets representing different node activities from the perspective of a single "current" node:

1. Packet injection into the node by the local processor port.
2. Packet transfer from the current node to a neighbor node.
3. Packet transfer from the neighbor node to a current node.
4. Packet ejection from the current node to a processor by the local processor port.

SPN subnet $\mathcal{N}_1$ is shown in Figure 5. The transition A_l generates the packets at a given rate λ and puts them in place WB_l. An inhibitor arc with cardinality K_l from WB_l to A_l is needed to ensure that the population of packets waiting to enter the node from the current node is finite. If K_l is smaller than the actual theoretical maximum number of packets waiting to enter the node from the local node, the inhibitor arc introduces an approximation through truncation of the state space.

Place FB contains tokens corresponding to free buffers inside the current node. The initial number of tokens N_b is the total number of buffers in a node. The immediate transition GB_l reserves a buffer, if one is available (indicated by a token in place FB). An inhibitor arc from the place I_l to a transition GB_l insures that the local processor port is not already busy (indicated by a different waiting token). After IB_l fires, the token is put in the place C_1 which is a place shared between the first and the second SPN subnets. Tokens in place

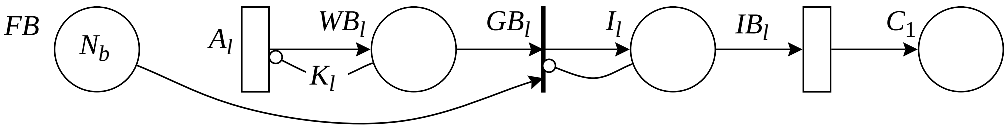


Figure 5: SPN subnet $\mathcal{N}_1$: packet injection into a node by the local processor port.

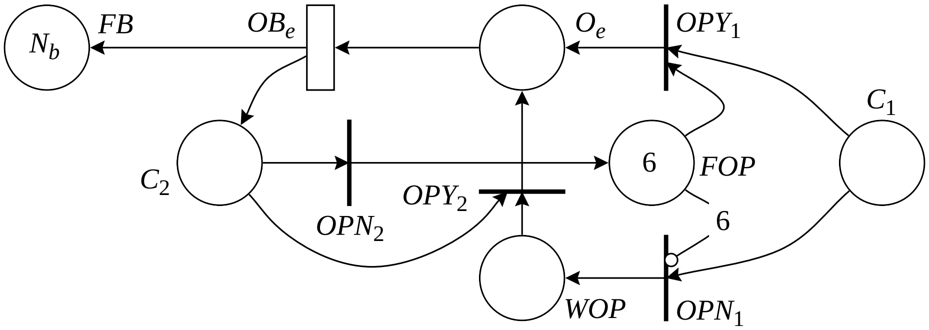


Figure 6: SPN subnet $\mathcal{N}_2$: packet transfer from the current node to its neighbor nodes.

C_1 represent packets stored in the current node buffers and which must be transfered to neighbor nodes.

SPN subnet $\mathcal{N}_2$ is shown in Figure 6. Place FOP contains tokens corresponding to free output ports of the current node to its neighbor nodes. The initial number of tokens in FOP is six, since there are six neighbor nodes. When a token arrives in place C_1, either

- the required output port is available, immediate transition OPY_1 fires, and a token is moved to place O_e, or
- the required ouput port is busy, immediate transition OPN_1 fires, and a token is moved to place WOP. An inhibitor arc with cardinality 6 from FOP to OPN_1 prevents the transition OPN_1 from firing when the place FOP contains all six tokens.

The probability that a particular output port is free is $\#(FOP)/6$, so this probability is assigned to transition OPY_1, and transition OPN_1 is assigned a probability of $1-\#(FOP)/6$.

Tokens in place O_e represent packets being transferred through output ports, and transition OB_e represents the completion of the packet transfer. Its rate is proportional to the number of tokens in place O_e ("infinite server" behavior). When the packet transfer to a neighbor node is completed, a buffer is released by returning a token to a place FB. Place C_2 represents the state when the busy output port has just been released. There are two possibilities at this point. Either

- the place WOP has a packet waiting for this particular output port, in which case transition OPY_2 will fire, removing a token from WOP, or
- there is no waiting packet for this output port, in which case immediate transition OPN_2 will fire, adding a token to FOP.

Let us calculate the probability that there is no waiting packet for this particular output port, and thus the probability that OPN_2 will fire. We know the packets are waiting for ports that are busy, and thus all packets are for either this output port, or one of the other output ports that are free. The probability that a single packet can use this particular output is thus $1/(6 - \#(FOP))$. The probability that that packet cannot use this output port is $(5 - \#(FOP))/(6 - \#(FOP))$. If we have $\#(WOP)$ waiting packets, then the probability that none of them can use the newly freed output port is

$$\left(\frac{5 - \#(FOP)}{6 - \#(FOP)}\right)^{\#(WOP)}$$

This, then, is the probability we assign to OPN_2; we assign the complementary probability to OPY_2.

SPN subnet $\mathcal{N}_3$ is shown in Figure 7. Transition A_e generates the packets ready to be sent by neighbor nodes to the current node. This rate is simply the product of the processor injection rate λ and the average path length, $\frac{2n-1}{3}$, since the packet is injected into a new neighbor node for each hop it takes. The structure and internal arrangement of the third SPN subnet is similar to the second SPN subnet described above. Place FIP (analogous to FOP) contains tokens corresponding to free input ports of the current node (with an initial marking of six tokens). For each waiting packet in place C_3 (analogous to C_1), there are two possibilities.

- If the required input port is available, immediate transition IPY_1 (analogous to OPY_1) fires and moves a token to a place WB_e. Transition GB_e then reserves a buffer in the current node, if there is a buffer available. Transition IB_e completes the packet transfer,

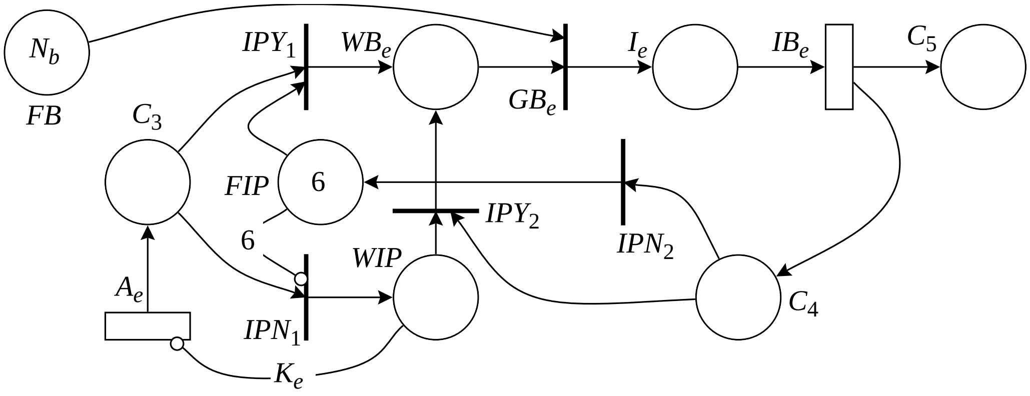


Figure 7: SPN subnet $\mathcal{N}_3$: packet transfer from a neighbor node to the current node.

and the packet ends up in place C_5 which is a place shared between the third and the fourth SPN subnets.

- If the required input port is busy, immediate transition IPN_1 (analogous to OPN_1) fires and moves a token to place WIP (analogous to WOP) representing waiting packets. An inhibitor arc with cardinality six from FIP to IPN_1 prevents transition IPN_1 from firing when place FIP contains all six tokens.

A token in a place C_4 (analogous to C_2) represents the state when a busy input port has just been released. As before, there are two possibilities:

- If there are no waiting requests for this input port, the immediate transition IPN_2 will fire and return a token to a place FIP.
- If place WIP has a packet waiting for this particular input port, the immediate transition IPY_2 will fire and move a token in place WB_e.

The probabilities assigned to these cases are similar to those for $\mathcal{N}_2$, since the situation is analogous. An inhibitor arc with cardinality K_e from WIP to A_e is needed to ensure that the population of packets waiting to enter the current node from the neighbor nodes is finite. This introduces an approximation in our model, by truncating the state space.

SPN subnet $\mathcal{N}_4$ is shown in Figure 8. A token in place C_5 (representing a packet received

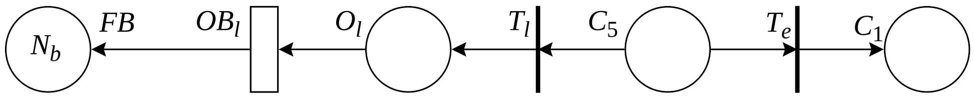


Figure 8: SPN subnet $\mathcal{N}_4$: packet ejection from the current node through the local processor port.

by the current node from its neighbors), is either destined to the current node, or must be transferred further.

- If the packet must be forwarded, then immediate transition T_e moves the token to a place C_1, the input place of the second SPN subnet $\mathcal{N}_2$.
- If the packet must be ejected toward the local node, immediate transition T_l moves the token to a place I_e. Transition OB_l represents the completion of the ejection, after which one buffer in the current node is released by returning a token to place FB. Note that the rate of transition OB_l is constant, not proportional to the number of tokens in place O_l, since the activity modeled corresponds to a "single server".

The composite SPN net $\mathcal{N}$ shown in Figure 9 is obtained as a superposition of $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4$ by merging their shared places, FB, C_1 and C_5. The meaning of the places and transitions in this SPN is summarized in Table 1 and the firing rate and probabilities of the transitions are given in Table 2. Note that places $C_i, i = 1, \ldots 5$ are always empty in a tangible marking. Moreover, transition GB_e has priority over transition GB_l, to ensure that the delivery of packets in transit takes precedence over the injection of new packets in the network. A priority to local packets or an equal priority to local and external packets could also be easily modeled.

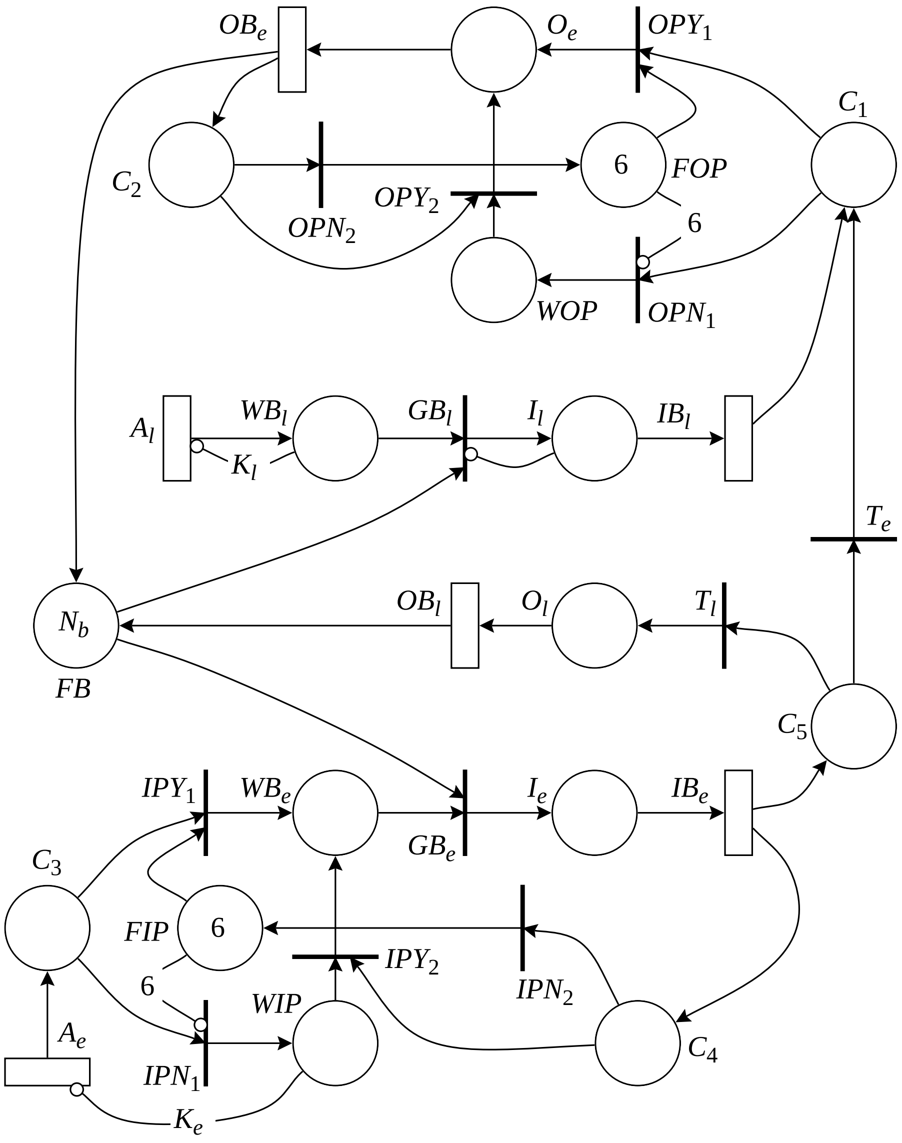
The inhibitor arcs with cardinality K_l and K_e from WB_l and WIP to A_l and A_e, respectively, introduce an approximation corresponding to a truncation of the state space. With exponentially distributed firing times, it is possible to have any number of packets waiting, but the probability of having many packets waiting decreases quickly unless the system is saturated. The introduced approximation does not influence the system behavior in the following two cases:

- the system is lightly loaded. In this case, the probability of having more than a few waiting packets in places WB_l and WIP is close to zero. Hence, the effect of the

Place	**Meaning**
FB	Free buffers.
WB_l	Locally generated packets, waiting for a buffer.
I_l	Locally generated packets, being copied into a buffer.
O_l	Packets to local node, being copied out of a buffer.
WB_e	Externally generated packets, waiting for a buffer.
I_e	Externally generated packets, being copied into a buffer.
O_e	Packets to external node, being copied out of a buffer.
FIP	Free input ports.
FOP	Free output ports.
WIP	Externally generated packets, waiting for an input port.
WOP	Packets to external node, waiting for an output port.
C_1	Choice: is the required output port available for the packet?
C_2	Choice: is there a packet waiting for the output port just released?
C_3	Choice: is the required input port available for an incoming packet?
C_4	Choice: is there a packet waiting for the input port just released?
C_5	Choice: is the local node the final destination for the packet?

Transition	**Meaning**
A_l	Locally generated packet is ready to be transmitted.
GB_l	Locally generated packet gets a buffer.
IB_l	Locally generated packet is put into a buffer.
OB_l	Packet directed to local node is read out of its buffer.
A_e	Externally generated packet is ready to enter the node.
GB_e	Externally generated packet gets a buffer.
IB_e	Externally generated packet is put into a buffer.
OB_e	Packet is transferred to an external node and frees its buffer.
IPY_1	Required input port is available.
IPN_1	Required input port is not available.
IPY_2	Input port just released is required by a packet waiting in WIP.
IPN_2	Input port just released is not required by any packet waiting in WIP.
T_l	Local node is the final destination for the packet.
T_e	Local node is not the final destination for the packet.
OPY_1	Required output port is available.
OPN_1	Required output port is not available.
OPY_1	Output port just released is required by a packet waiting in WOP.
OPN_2	Output port just released is not required by any packet waiting in WOP.

Table 1: Meaning of places and transitions in the SPN $\mathcal{N}$.



IPN_2 is disabled if $\#(FIP) = 5 \wedge \#(WIP) > 0$.
OPN_2 is disabled if $\#(FOP) = 5 \wedge \#(WOP) > 0$.
T_e is present only if $n > 2$.

Figure 9: Approximate SPN model $\mathcal{N}$.

introduced inhibitor arcs becomes negligible.

- the system is saturated even with the inhibitor arcs, which reduce the effective arrival rate. In this case, the probability of WB_l or WIP being nonempty is close to one

Transition	**Firing rate**
A_l	λ (λ is an input parameter to be varied)
IB_l	$1/720\ tu^{-1}$
OB_l	$1/720\ tu^{-1}$
A_e	$(2n-1)/3 \cdot \lambda$
IB_e	$\#(I_e)/720\ tu^{-1}$
OB_e	$\#(O_e)/x$ (x is the unknown iteration parameter)

Transitions		**Priority level**	**Firing probabilities**	
IPY_1	IPN_1	1	$\beta = \frac{\#(FIP)}{6}$	$1-\beta$
IPY_2	IPN_2	2	γ	$1-\gamma = \left(\frac{5-\#(FIP)}{6-\#(FIP)}\right)^{\#(WIP)}$
T_l	T_e	3	$\delta = \frac{3}{2n-1}$	$1-\delta$
OPY_1	OPN_1	4	$\epsilon = \frac{\#(FOP)}{6}$	$1-\epsilon$
OPY_2	OPN_2	5	ϕ	$1-\phi = \left(\frac{5-\#(FOP)}{6-\#(FOP)}\right)^{\#(WOP)}$
GB_l		6	1	
GB_e		7	1	

Table 2: Firing rates and probabilities of the transitions in the SPN $\mathcal{N}$.

and increasing K_l or K_e only increases the state space and the solution cost, without changing in any appreciable way the numerical value computed for the throughput of packets.

The only undefined parameter in Table 2 is x, the average time required by an outgoing packet to obtain and fill a buffer in the next node on its path. Only after this time elapses can the buffer for node i be released (through the arc from OB_e to FB). By symmetry, this time has the same average as the time that a packet in place WB_e must wait before it can obtain a local buffer slot and enter place I_e, plus the time to fill the slot, 720 tu.

Hence, we set up the following fixed-point iteration scheme:

- Choose an initial guess $x^{(0)}$ for x.
- Compute the successive values for x as $x^{(i)} = w + 720\ tu$, where w is the average waiting time and is obtained using Little's law:

$$w = \frac{E[\text{number of packets waiting}]}{E[\text{throughput of packets}]} = \frac{E[\#(WIP) + \#(WB_e)}{E[\text{rate}(A_e)]}$$

- Stop the iterations when $x^{(i)}$ and $x^{(i+1)}$ are sufficiently close.

6 Numerical results

In the numerical experiments, we considered systems of size $E_3, \ldots E_8$, with $N_b = 10$ or 12, interarrival packet time $1/\lambda = 1000\ tu, 1100\ tu, \ldots 2000\ tu$, and truncation parameters $K_l = K_e = 3$. In most cases, the fixed-point scheme converged in just a few iterations. For example, for E_5, $N_b = 12$, $1/\lambda = 1200\ tu$, four iterations are needed, starting from the initial guess $x^{(0)} = 1000\ tu$, to obtain four significant digits: $x^{(1)} = 744.9\ tu$, $x^{(2)} = 729.3\ tu$, and $x^{(3)} = x^{(4)} = 728.8\ tu$.

We discovered that the number of iterations increases as the the system is stressed, that is, as the mesh size increases, the number of buffers decreases, or the interarrival packet time decreases. The maximum number of iterations, thirteen, was observed for E_8, when $N_b = 10$, and $1/\lambda = 1000\ tu$, resulting in $x^{(13)} = 912.7\ tu$. Interestingly, this happens even if our initial guess, $x^{(0)} = 1000\ tu$, turned out to be always an overestimate of the value obtained for x through the iterations, hence it is closest to the value $x^{(13)} = 912.7\ tu$ than to the final value of x obtained for any other combination of input parameters studied.

For our study, we focus on the **average total packet latency time** τ, defined as the average time elapsing from the instant a packet is generated by its source local processor (firing of transition A_l), to the instant it is read by its destination local processor (firing of transition OB_l). In the model of Fig. 9, this is obtained as the sum of three components:

- The "injection time": the average time a packet waits before it is put into a buffer in the source node, computed using Little's law:

$$\frac{E[\#(WB_l) + \#(I_l)]}{E[\text{rate}(IB_l)]}$$

- The "ejection time": the average time a packet waits before it is removed from the buffer in the destination node, computed using Little's law:

$$\frac{E[\#(O_l)]}{E[\mathrm{rate}(OB_l)]}$$

- The "transit time": the average time a packet spends in transit, computed as the product of the time to perform a hop times the expected number of hops:

$$x \cdot \frac{2n-1}{3}$$

Fig. 10 shows the value of τ as a function of the average interarrival time λ^{-1}, for various system sizes (n) and number of buffers (N_b).

7 Comparison with simulation results

In the real system, the time required to perform most activities is far from being exponentially distributed. With simulation, we can accurately portray any distribution, including exponential, uniform, or constant. For example, packet transfers into or out of buffers are constants of 720 tu in the simulation model.

In a Markovian SPN, all time delay distributions are approximated with exponential distributions. By using Erlang(k) distributions with the same mean, we better approximate the constant, or almost constant, nature of the random variables involved. Higher values of k result in better approximation, but they also increase the size of the state space.

With exponential distributions and $N_b = 10$, the underlying CTMC contains 32,797 nodes and 308,267 arcs. Increasing N_b to 12 results in 49,259 nodes and 476,838 arcs. Hence, we limited ourselves to check the effect of using Erlang(2) distributions. When using the Erlang(2) distribution for the $N_b = 10$ case, the underlying CTMC contains 749,795 nodes and 6,829,308 arcs.

The comparison of SPN model results (using either exponentially or Erlang(2) distributed times) against the simulation results is shown in Fig. 11 for the case of networks E_3 and E_6, with $N_b = 10$. The percent workload, defined as $\lambda \cdot 720 \cdot 100\%$ is on horizontal axis. In general, the results are off by only a few percent. As we would expect, the results with Erlang distributions agree with the simulation results much more closely. Interestingly, the results for a larger network show closer agreement than those for a small network. This is probably

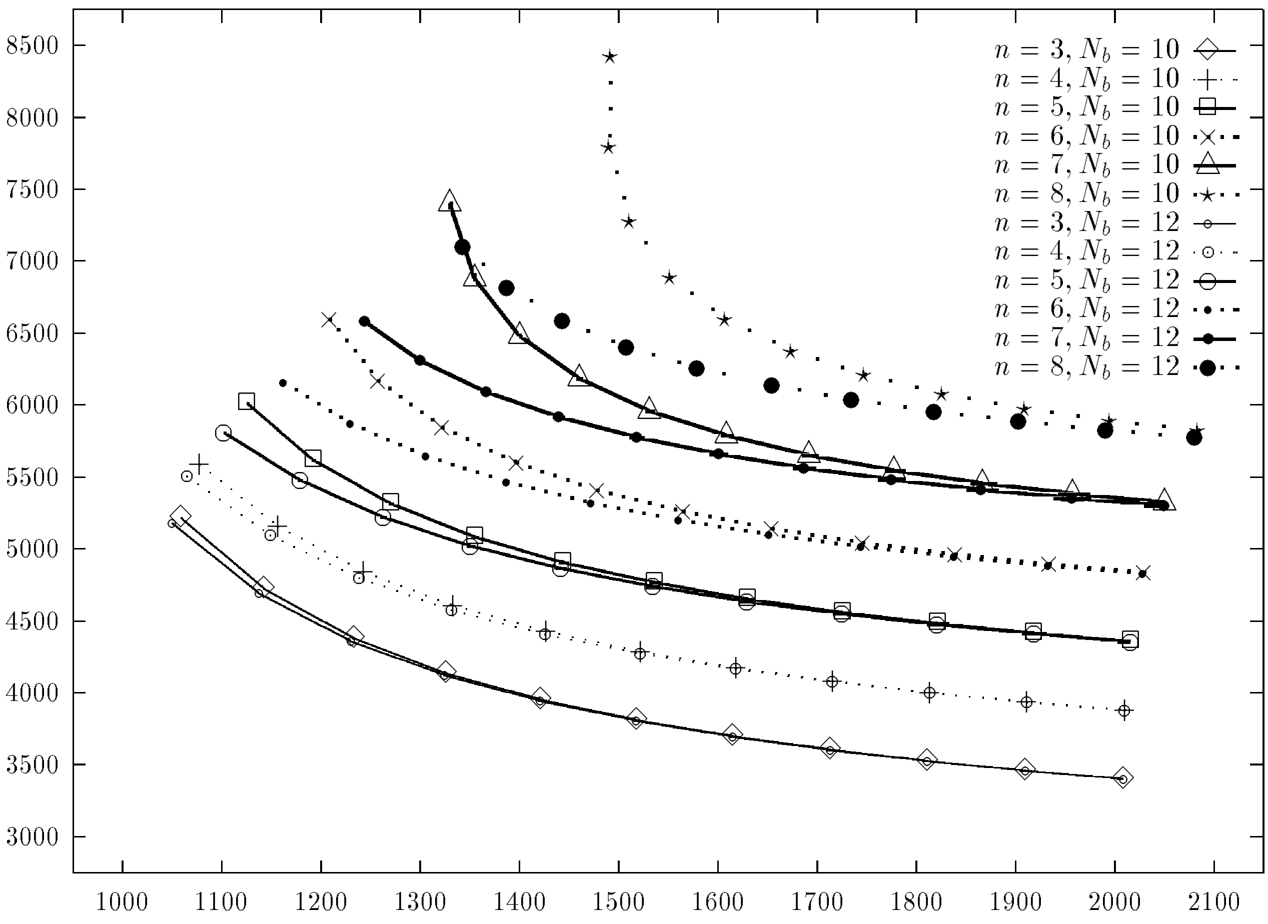


Figure 10: τ, in tu, as a function of λ^{-1} for different values of n and N_b.

due to the fact that, under heavy load, the shape of the distribution of the interarrival times to the transitions is less important, since all transitions are more likely to be busy most of the time anyway.

8 Conclusion

In this paper, we presented our experience in using Stochastic Petri Nets to model an industrial size application. The paper has shown that Stochastic Petri net models can exploit the symmetry of the system to construct a tractable, but approximate, analytic model, and that they can yield results very close to those of a detailed simulation model, with much less computational effort.

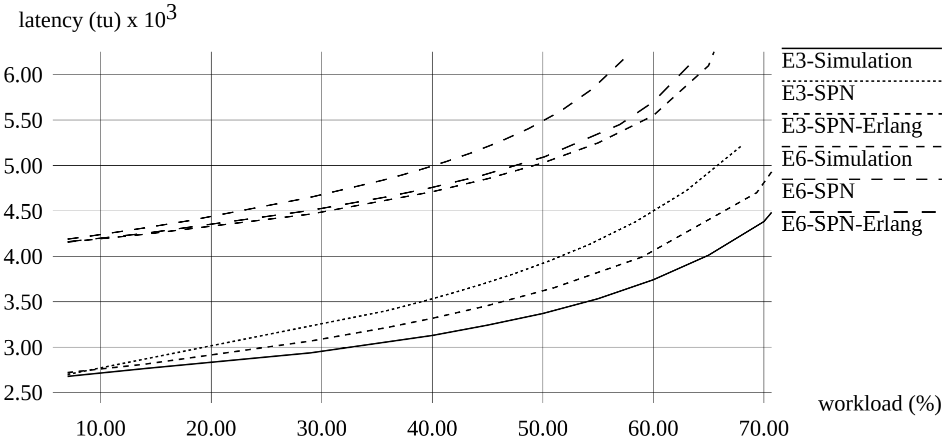


Figure 11: Comparison with simulation results for the case of interconnect fabric E_3 and E_6 with $N_b = 10$.

One of the difficulties in using the SPNs is that all time delays are approximated with exponential distributions, while, in the real system, many time delays are constants. By using Erlang(k) distributions with a given mean in the SPN model, we can better approximate the constant distribution. However, as we have seen, this can lead to a sharp increase in the size of the state space. To solve this problem, we are investigating the use of SPN having underlying discrete-time Markov chains [11, 4]. We are currently working on the design of a software package that will allow us to solve SPNs with continuous-time phase-type distributions, as those used in this paper, their discrete equivalent (where any distribution over the integers can be used, such as constant, discrete uniform, geometric or modified geometric), and even, under certain restrictions, with mixture of the two (see the “deterministic and stochastic Petri nets” [2] and recent extensions [6]).

9 References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of Generalized Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comp. Syst.*, 2(2):93–122, May 1984.

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

[3] G. Chiola. GreatSPN Users' Manual. Technical report, Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy, 1987.

[4] G. Ciardo. Discrete-time Markovian stochastic Petri nets. In W. J. Stewart, editor, *Numerical Solution of Markov Chains '95*, pages 339–358, Raleigh, NC, Jan. 1995.

[5] G. Ciardo, A. Blakemore, P. F. J. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of Markov reward models using Stochastic Reward Nets. In C. Meyer and R. J. Plemmons, editors, *Linear Algebra, Markov Chains, and Queueing Models*, volume 48 of *IMA Volumes in Mathematics and its Applications*, pages 145–191. Springer-Verlag, 1993.

[6] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. *IEEE Trans. Softw. Eng.*, 20(7):506–515, July 1994.

[7] G. Ciardo and K. S. Trivedi. A decomposition approach for stochastic reward net models. *Perf. Eval.*, 18(1):37–59, 1993.

[8] G. Ciardo, K. S. Trivedi, and J. Muppala. SPNP: stochastic Petri net package. In *Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM'89)*, pages 142–151, Kyoto, Japan, Dec. 1989. IEEE Computer Society Press.

[9] K. Jensen. Coloured Petri nets. In *Petri Nets: Central Models and Their Properties* LNCS, vol. 254, pages 248–299. Springer-Verlag, 1987.

[10] L. Kleinrock. *Queueing Systems Volume I: Theory.* Wiley, 1975.

[11] M. K. Molloy. Discrete time stochastic Petri nets. *IEEE Trans. Softw. Eng.*, 11(4):417–423, Apr. 1985.

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

[13] C. Petri. *Kommunikation mit Automaten.* PhD thesis, University of Bonn, Bonn, West Germany, 1962.

[14] R. M. Shapiro. Validation of a VLSI chip using hierarchical colored Petri nets. In *High-Level Petri Nets*, pages 667–690. Springer-Verlag, 1991.