Modeling A Fibre Channel Switch with Stochastic Petri Nets

Gianfranco Ciardo*

Department of Computer Science

College of William and Mary

Williamsburg, VA 23187-8795, USA

Ludmila Cherkasova, Vadim Kotov, and Tomas Rokicki

Hewlett-Packard Labs

1501 Page Mill Road

Palo Alto, CA 94303, USA

Abstract. Discrete-event simulation is the most common method of performance analysis because it allows the modeling of arbitrarily complex systems with a minimum of assumptions. Unfortunately, programming errors, improper statistical methods, and runs that are too short can yield inaccurate results. The numerical analysis of Markovian models can supplement discrete-event simulation, but only if the system can be simplified to reduce its state space to a manageable level. The approximations introduced through such simplification can also cause a significant amount of error. Recent advances in stochastic Petri nets and Markov chain analysis, and the availability of fast workstations with large amounts of memory, allow the numerical solution of much larger and more complex Markov models. In this report we discuss the analysis of a complex packet switch through both discrete-event simulation and fixed-point approximate numerical solution of Markovian stochastic Petri net models. We compare the results obtained from them and identify the error in the various approximations. We conclude that stochastic Petri net analysis can be a useful technique even for complex systems.

Key Words. Performance analysis, numerical analysis, Markovian models, stochastic Petri nets, approximate models, discrete-event simulation, Fibre Channel model.

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

Contents

1 Introduction

As computing systems become increasingly complex and concurrent, performance analysis, especially in the architectural design phase, becomes more important. Many different techniques are used in performance analysis. Back-of-the-envelope calculations and simple numerical approximations can often generate bounds on performance. Queueing theory and Markovian analysis can provide insight into the steady-state performance of the system. Discrete-event simulation can provide a view of the system behavior at any level of detail, provided enough modeling manpower is available.

Each of these techniques involves different types of approximations, and building a model for each of these techniques requires different types of skills. Back-of-the-envelope calculations require the intuitive identification of the major bottlenecks — usually this technique provides only an upper bound on the possible performance. Queueing theory and Markovian analysis require extensive knowledge of probability theory and skill at approximating discrete-state systems with simplified mathematical models. Such approximations are error-prone and difficult to check. Usually the approximations must be significant in order to make the state-space manageable; for example, they may require to ignore specific types of interactions, which can significantly compromise the results. Discrete-event simulation requires programming skills and significant amounts of computer power. Because programming is still mostly an empirical art, subtle implementation errors in the simulation can yield incorrect performance results. Such problems might go undetected because the simulation 'works', just like an incorrect implementation in a sort routine can cause it to correctly sort the data, just not as fast as it should. In addition, using discrete-event simulations, especially for reactive systems, requires waiting for steady state and then collecting enough sample points to obtain statistically significant results.

Because the set of skills for each technique is so different, many practitioners limit themselves to only one technique. Yet, when carrying on the performance evaluation of a system in an industrial setting, different phases of the design require different types of analysis. Initially, back-of-the-envelope calculations can give some initial guidance in early design decisions. When a basic overall architecture is selected, analytic or Markovian models can provide more accurate results. As the design proceeds and more system parameters become known, more elaborate and accurate simulation models are needed to evaluate further alternatives.

As a result, the authors had to use all of the above performance analysis techniques in their modeling efforts. This paper shows how, with stochastic Petri nets (SPNs), the symmetry of the system and its behavior can be exploited to construct a tractable approximate numerical model. In the paper, we point out the approximations used in the SPN model and we discuss how much each approximation affected our final results.

To validate the correctness of the approach we used, and in order to show that by folding the detailed Fibre Channel SPN model into an approximate SPN model, the essential system behaviour remained the same, the SPN model results were compared against the simulation model results.

The remainder of the paper presents our results in more detail. Section 3 describes the structure and basic features of Fibre Channel switch. Section4 provides an overview of our simulation model. Sections 5 and 6 present an approximate SPN model and the results obtained from its study, respectively. Finally, Section 7 contain some possible directions for future work in this area.

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 [7].

2 Basic SPN Notions and Denotations

A Petri net (PN) [15, 16] 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 [15] 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, 4, 7].

The GSPN model can be extended by allowing phase-type distribution [13] 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.

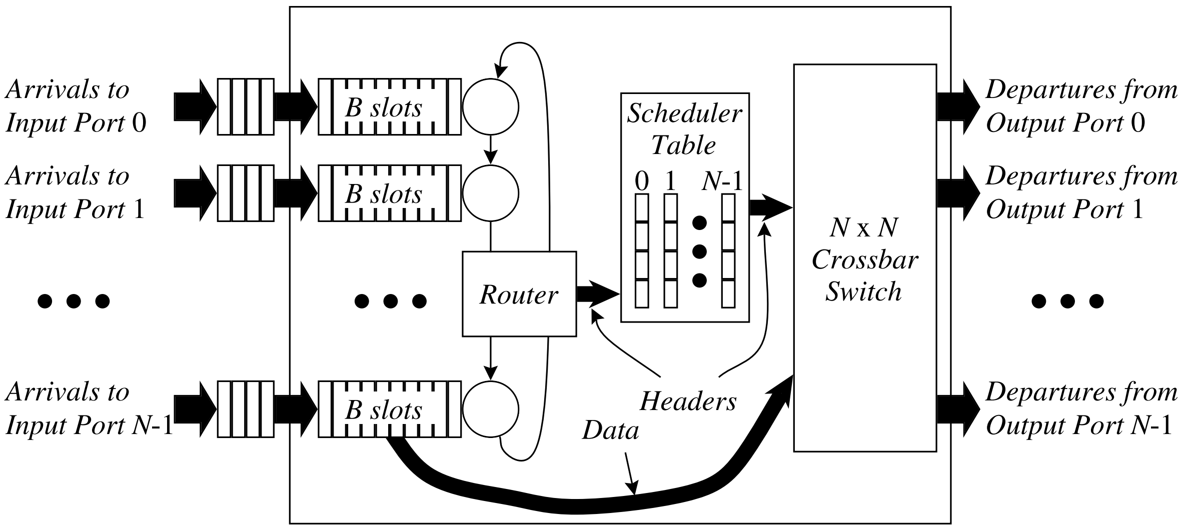

Figure 1: System structure.

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 [7].

3 System

We consider a switch with N ports, numbered from 0 to $N-1$ (see Figure 1). Each port has an associated pool of B buffers to store incoming messages. When a message arrives at a port, it starts being read into a buffer as soon as one is available. A switch router iteratively polls each port for new messages in a cyclical fashion. The router polls port i to check whether it contains messages to be routed. If so, the router reads the header information for the first of them and inserts it into a scheduler table, organized as N FIFO queues, one for each possible message destination (we assume that a message arriving from port p chooses any of the other $N-1$ ports with uniform probability). Then, it polls port $(i+1)$ mod N, and so on. If there is no message to be routed, the amount of time spent by the router at a port is negligible. Note that the router can start routing a message even if the message has not been completely read into the buffer. A switch scheduler uses the scheduler table to control the use of a $N \times N$ crossbar switch. The constraints that the scheduler must observe are:

- The crossbar switch can transfer at most one message from each source port at any given time.
- The crossbar switch can transfer at most one message to each destination port at any given time.

	Destination			
	0	**1**	**2**	**3**
First	1	2	1	0
Second	2		3	2
Third	1			1
...				

Table 1: A possible scheduler table configuration.

- A message can be transferred to its destination port only when it is at the top of the queue for that destination.

For example, if $N = 4$ and the content of the scheduler table is as in Table 1, the crossbar could be transferring the messages $\{1 \mapsto 0, 2 \mapsto 1, 0 \mapsto 3\}$ or the messages $\{2 \mapsto 1, 1 \mapsto 2, 0 \mapsto 3\}$. Note that, in the first case, the transfer $3 \mapsto 2$ is not possible, even if the source 3 and the destination 2 are idle, because the message from source 3 is not at the top of the queue for destination 2. In other words, source 1 blocks source 3.

As a case study, we considered a Fibre Channel switch with 16 ports and port bandwidth of 26.6 Mbytes/sec. Each port has fifteen buffers, $B = 15$. The value of the bandwidth affects the rate at which data moves through each of the components of the switch. Hence, the time to transfer a message of length L_{Msg} into a buffer slot or through the crossbar is $T_{Msg} = L_{Msg}/26.6\mu$sec. The amount of time required by the router to route a message is T_{Rtr}, while the amount of time between the instant a message arrives in the queue and the router can begin routing it is T_{Hdr}.

4 Simulation model

We constructed a simulation model capturing the essential architectural features of the switch. This simulation model was built with the CSIM C++ class library and consisted of approximately 3,000 lines of code.

All simulation runs involved over 1,000,000 messages, and the 95% confidence intervals using the batch means approach were tighter than $\pm 1\%$. For low traffic, the confidence intervals were significantly better than this.

We used a variety of message length distributions, with the expected varying results. One workload we considered was a bimodal distribution, with mostly short messages, but most of the bandwidth taken by fixed-size long messages; we considered this to be a typical workload. Another workload we considered was with a geometric message length distribution. The difference between these two workloads was significant, especially with

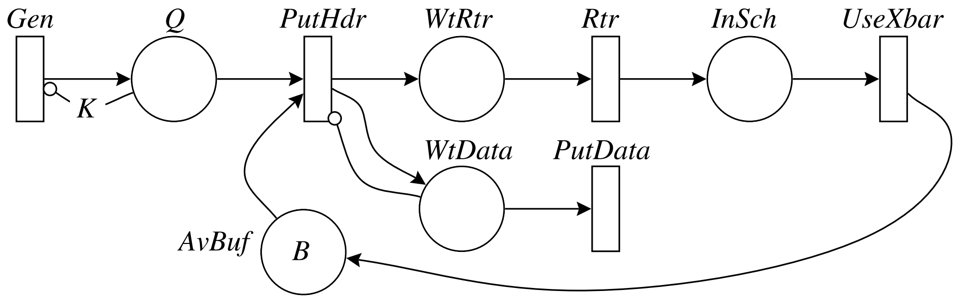

Figure 2: Approximate SPN model of the switch.

Transition	**Firing rate**
Gen	LF/T_{Msg}
$PutHdr$	$1/T_{Hdr}$
$PutData$	$1/(T_{Msg} - T_{Hdr})$
Rtr	$1/(T_{Rtr}(1 + (N-1)\alpha))$
$UseXbar$	$\gamma(\#(InSch))/T_{Msg}$

Table 2: Firing rates for the transitions of the approximate SPN model.

respect to the maximum attainable aggregate bandwidth. Under the bimodal message length distribution, the switch was capable of about 60% aggregate utilization, while under the exponential distribution, the switch was only capable of about 52% aggregate utilization. This difference is explained by some classic papers on crossbar contention [3, 17] which we will consider more carefully in the next section.

The results presented in this paper consider only the geometric message length distribution because this is the one most easily modeled by SPNs with exponentially distributed firing times.

5 Approximate model

In the following, we assume that all activities have exponentially distributed durations. This is realistic if the arrival streams are roughly Poisson, and if the message length has a geometric distribution. The duration of the router activities is constant, so this implies an approximation. However, we will see that the router is not the bottleneck, so the impact of using a diffentent distribution is, in this case, minor.

An exact SPN model for the switch can be built, but its state space is exceedingly large for practical values of N and B, rendering unfeasible a numerical solution approach based on the enumeration of the state space. Because of the extensive symmetries in the system, and especially because of the FIFO policy used in the scheduler table, a colored SPN [12, 5] greatly simplifies the description of the model, at the cost of having to read net inscriptions. However, discrete-event simulation must be used to study this model.

Hence, we explore the use of an approximate model based on the idea of SPN decomposition and fixed-point iteration [9]. It considers the switch from the point of view of a particular source port, say source 0 (Fig 2).

A message arriving at source 0 (firing of transition Gen) must wait (in place Q) for a buffer. The inhibitor arc from Q to Gen with multiplicity K enforces a truncation of the state space. It is needed to ensure that the token population in Q is kept between 0 and K. When a buffer is available (place $AvBuf$ is not empty), the message header starts being copied into it (transition $PutHdr$). After this, the rest of the message can be copied (transition $PutData$), but, concurrently, the header (in place $WtRtr$) can start requesting service from the router (transition Rtr), which will put the message header information in the appropriate queue the scheduler table (place $InSch$). Given our assumptions, the message chooses a particular queue with uniform probability over $\{1, 2, \ldots, N-1\}$, but the model does not represent this choice explicitly (the tokens in $InSch$ represent the total number of messages from source 0 queued for any destination). When the message reaches the top of its queue, it starts using the crossbar switch (transition $UseXbar$) and, when done, it leaves the switch and releases the buffer (arc from $UseXbar$ to $AvBuf$).

The model just described observes the system at a rough level of detail:

- Transition $PutData$ represents the copying of the remaining part of the message into a buffer, after an initial portion has been copied. This is done because the router, if available, can start routing the header information (transition Rtr) in parallel with this activity. However, this approach is correct only for deterministic firing times. Given our assumption of exponentially distributed firing times, it is possible for a "data" token to be still in place $WtData$ after its corresponding "header" token has been routed and the message to which they both refer has actually left the system using the crossbar switch. This sequence of events cannot happen in reality.

- In the real system, the router follows a cyclical polling pattern, checking whether there is a message to be routed at port 0,1, ..., $N-1$, 0, and so on. In the approximate model, messages to be routed simply wait for the firing of transition Rtr.

- In the real system, a header from source 0 waits in the queue for destination i, competing with other headers from any source port other than i (including 0 itself) and destination port i. More specifically, its relative position in the queue is relevant to the determination of the time it must wait before using the crossbar switch. In the approximate model, message headers simply wait for the firing of transition $UseXbar$.

The impact of the modeling error described in the first point could be reduced by using distributions with smaller variance, such as the Erlang distribution [11], for transitions

PutHdr and *PutData*, since these transitions are the only ones corresponding to events for which the system actually spends exactly the same amount of time, independent of the state of the router, scheduler table, or crossbar switch.

5.1 Heuristic approximations

The last two observations, however, deserve particular attention. Indeed, the state of the approximate model does not contain enough information to determine the timing of transitions *Rtr* and *UseXbar*, so we must resort to a heuristic argument.

The heuristic for the rate of *Rtr* is fairly simple: if we knew the set of probabilities $\{\alpha_n : n = 1, 2, \ldots, N-1\}$ that the router will route a message from each of the other $N-1$ source ports during the current polling cycle, we could estimate the average time between the firing of one of *Rtr* as

$$T_{Rtr} \cdot \left(1 + \sum_{n=1}^{N-1} \alpha_n\right).$$

In other words, we could say that a full polling cycle of the router, when there is message header from port 0 to be routed, requires one T_{Rtr} with probability one, for source 0, plus one T_{Rtr} from each port n, with probability α_n, on average. This assumes that the router is always in the worst position from the point of view of a header from 0: the router has just finished polling port 0. This is justified under heavy load, that is, if it is highly likely that a second header requires to be routed from port 0 while the previous one has not be completely routed yet. Since the router is not the bottleneck for this system, this will rarely be the case, but we nevertheless use this assumption, with the knowledge that it is somewhat pessimistic.

The heuristic for the rate of *UseXbar* is more complex. The crossbar switch is used by exactly one message from source 0 whenever at least one of the queues for destinations 1, ..., $N-1$ has a message header from source 0 in the top position. Hence, the problem is to determine what is the probability $\gamma(i)$ of this event, when the number of tokens in place *InSch*, indicated by $\#(InSch)$, is i. The marking-dependent rate of *UseXbar* can then be simply set to $\gamma(\#(InSch))/T_{Msg}$.

We now focus on the computation of γ. Define s^n_k to be the number of message headers from source n in the queue for destination k in the scheduler table. If we knew the steady-state probabilities

$$\beta^n(i_0, i_1, \ldots, i_{n-1}, i_{n+1} \ldots, i_{N-1}) = \Pr\{s^n_0 = i_0, s^n_1 = i_1, \ldots, s^n_{n-1} = i_{n-1}, s^n_{n+1} = i_{n+1}, \ldots, s^n_{N-1} = i_{N-1}\}$$

for $n \in \{1, \ldots, N-1\}$ and

$$\beta^0(i_1, \ldots, i_{N-1}|i) = \Pr\{s^0_1 = i_1, \ldots, s^0_{N-1} = i_{N-1} \mid i_1 + \cdots + i_{N-1} = i\},$$

we could, in principle, compute the conditional probability of each possible configuration of headers from all sources in each destination queue, given that $\#(InSch) = i$, and then obtain the conditional probability $\gamma(i)$ that source 0 is using the crossbar switch as the sum of the probabilities of the configurations having a header from source 0 at the top of at least one queue. Unfortunately, there are $(B+1)^N$ possible header mixes in the scheduler (from 0 to B headers from each port $i \in \{0, \ldots, N-1\}$). Each mix corresponds to many configurations, since the headers from port i can be distributed over $N-1$ possible ports and the relative position of the headers in each queue should be considered as well. The resulting number of configurations for practical values of B and N is unacceptably large, even taking into account symmetries to reduce the number of configurations that need to be considered.

Hence, we must resort to a rougher approximation, based on the knowledge of just the average number σ_k^n of headers from source n in the queue for destination k, rather than its distribution. We can define the average number of headers from sources $1, \ldots, N-1$ in competition with headers from source 0 for access to the output port of destination k as

$$\sigma_k = \sum_{n=1}^{N-1} \sigma_k^n.$$

Given the values σ_k, we then compute an approximate value for $\gamma(i)$ using a recursive approach. Define $\Gamma(i,j)$ to the be the probability that there is a header from source 0 at the top of at least one of the queue for destinations $\{1, \ldots, j\}$, given that there is a total of i headers from source 0 waiting in these j queues. Clearly, $\gamma(i) = \Gamma(i, N-1)$. The value of Γ can be defined recursively as:

- $\forall j \in \{1, \ldots, N-1\}, \Gamma(0,j) = 0.$
- $\forall i \in \{1, \ldots, B\}, \Gamma(i,1) = \dfrac{i}{i+\sigma_1}.$
- $\forall i \in \{1, \ldots, B\}, \forall j \in \{2, \ldots, N-1\}, \Gamma(i,j) = \displaystyle\sum_{k=0}^{i} c(i,k,j) \left(\frac{k}{k+\sigma_j} + \frac{\sigma_j}{k+\sigma_j} \Gamma(i-k, j-1) \right).$

where $c(i,k,j)$ is the probability that, when distributing i headers over j queues, queue j contains exactly k headers, and is given by the binomial distribution:

$$c(i,k,j) = \begin{pmatrix} i \\ j \end{pmatrix} \left(\frac{1}{j}\right)^k \left(\frac{j-1}{j}\right)^{i-k}.$$

The computation of Γ can then be performed using a convolution-like tableau initialized as in Table 3. In practice, columns $\{1, \ldots, j-2\}$ are not needed to compute columns $\{j, \ldots, N-1\}$, hence only two adjacent columns need to be stored at any time. At the end, the last column contains the values of $\gamma(i)$, for all possible values of i. This computation, which has complexity $O(B^2N)$, can be performed before starting the analytical solution of the SPN. Table 4 gives an example of the values of γ.

Unfortunately, this heuristic is overly optimistic under high traffic. Define the set S_j, $j = 0, \ldots, N-1$ to be the set of destination queues having a message from source j at the top. When the system is saturated, no destination queue is empty, hence

$$\sum_{j=0}^{N-1} |S_j| = N.$$

	1	2	$\cdots$	$N-2$	$N-1$
0	0	0	$\cdots$	0	0
1	$\frac{1}{1+\sigma_1}$		$\cdots$		
2	$\frac{2}{2+\sigma_1}$		$\cdots$		
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$B-1$	$\frac{B-1}{B-1+\sigma_1}$		$\cdots$		
B	$\frac{B}{B+\sigma_1}$		$\cdots$		

Table 3: The tableau for the computation of Γ.

Given the FIFO service restriction, only messages corresponding to these N elements can use the crossbar switch. However, only one message from each source j can be transferred at any time, hence, exactly N transfers can occur only when S_j is a singleton for each j. Whenever $|S_j| = k > 1$, $k-1$ source ports are being blocked by source j, and the number of concurrent transfers is decreased by $k-1$.

If we ignore that a source port cannot send to itself (for $N = 16$, we found that the values of γ computed using our heuristic with and without this restriction vary by less than 1%) the problem is simply the classic crossbar contention analysis. Our heuristic approach for estimating γ assumes a uniform random selection of the possible sources, given information about the number of packets from source 0 and the average queue length. This would make evenly distributed sets $|S_j|$ more likely than largely skewed ones. For instance, with $N = 4$, a total of four messages in the scheduler table, and a uniform random assignment of sources, there is only one case out of 256 where $|S_0| = 4$. Yet, there are $4! = 24$ possible cases where each packet comes from a distinct source port, and thus the assignment $|S_0| = |S_1| = |S_2| = |S_3| = 1$ is 24 times more likely than $|S_0| = 4$.

This uniform random selection assumption fails as traffic increases, as shown by Bhandarkar and Fuller [3], Rau [17], and others. Their results show that, for a saturated system with N processors, N memories, and exponential service times, all assignments of processors queued for the memories are equally likely. That is, with $N = 4$ and $|S_0| + |S_1| + |S_2| + |S_3| = 4$, the probability that $|S_0| = 4$ is identical to the probability

i	$\gamma(i), \sigma = 0.5$	$\gamma(i), \sigma = 1$	$\gamma(i), \sigma = 2$	$\gamma(i), \sigma = 4$	$\gamma(i), \sigma = 8$	$\gamma(i), \sigma = 15$
0	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1	0.6666667	0.5000000	0.3333333	0.2000000	0.1111111	0.0625000
2	0.8829630	0.7444444	0.5518519	0.3582222	0.2092181	0.1208640
3	0.9578748	0.8679506	0.6974705	0.4844110	0.2961889	0.1754802
4	0.9844816	0.9310583	0.7949471	0.5852167	0.3733311	0.2266003
5	0.9941552	0.9636429	0.8604649	0.6658582	0.4417846	0.2744544
6	0.9977517	0.9806392	0.9046797	0.7304590	0.5025543	0.3192574
7	0.9991176	0.9895923	0.9346361	0.7822814	0.5565257	0.3612096
8	0.9996470	0.9943538	0.9550110	0.8239096	0.6044798	0.4004977
9	0.9998561	0.9969096	0.9689216	0.8573941	0.6471053	0.4372958
10	0.9999403	0.9982939	0.9784541	0.8843637	0.6850104	0.4717666
11	0.9999748	0.9990502	0.9850103	0.9061145	0.7187317	0.5040614
12	0.9999892	0.9994670	0.9895357	0.9236791	0.7487434	0.5343217
13	0.9999953	0.9996985	0.9926702	0.9378814	0.7754645	0.5626794
14	0.9999979	0.9998281	0.9948488	0.9493794	0.7992655	0.5892578
15	0.9999991	0.9999013	0.9963681	0.9586997	0.8204740	0.6141717

Table 4: Example values of γ for various values of σ ($B = 15$, $N = 16$).

that $|S_0| = |S_1| = |S_2| = |S_3| = 1$. Informally, this effect is due to the variance in service times, which cause messages from a given source to build up for the various destination queues.

It is important to note that the dynamic effects become more pronounced with a higher variance in service time. Indeed, the expected bandwidth for a $N \times N$ system processor-memory crossbar with expected service time normalized to one is [17]:

- $N^2/(2N-1)$ if the memory service time is exponentially distributed.
- $2N - 1/2 - \sqrt{(2N-1/2)^2 - 2N^2}$, asymptotically, if the memory service time is constant.

These impose a constraint on the value of γ, since, if we focus on just source 0, the corresponding throughput should be a fraction $1/N$ of the above bandwidths, and

$$\text{“throughput for source 0”} = \sum_{i=1}^{B} \Pr\{\#(InSch) = i\} \cdot \gamma(i) \cdot \frac{1}{T_{Msg}}.$$

For the exponential case, then,

$$\sum_{i=1}^{B} \Pr\{\#(InSch) = i\} \cdot \gamma(i) \leq N/(2N-1).$$

For $N = 16$, $N/(2N-1) \approx 51.6\%$. If the service time is constant, the analogous upper bound is somewhat higher, $(2N - 1/2 - \sqrt{(2N-1/2)^2 - 2N^2})/N \approx 59.9\%$.

In the fixed-point iteration, we use two heuristics to enforce the bound $N/(2N-1)$, we will call the corresponding models “SPN-1” and “SPN-2”. Since only the value of γ is affected, both models are still described by the SPN of Figure 2 with the rates of Table 2.

In the first heuristic, SPN-1, we simply define $\gamma(i)$ to be a weighted average of the value $\Gamma(i, N-1)$, obtained from our initial approximation using the tableau, and the limiting value $N/(2N-1)$, which we know is the steady-state probability that the switch is being used by port 0 when the system is saturated:

$$\gamma(i) = \frac{\Gamma(i, N-1) + N/(2N-1) \cdot \sigma}{1 + \sigma}.$$

The weights corresponding to the two values are one and σ, respectively. The reason for using a weight increasing with σ for the second value is obvious: the larger σ is, the closer the system is to being saturated. The actual choice of σ as opposed to, say, $\sqrt{\sigma}$, 2σ, or σ^2, however, is simply the most natural, but it is otherwise arbitrary.

In the second heuristic, SPN-2, we want instead to ensure that the value of $\gamma(i)$ when i is “close” to σ, the average number of messages in every queue, is not greater than $N/(2N-1)$. To this end, we define the “typical value of γ” as

$$\gamma_{typ} = \Gamma(\lfloor\sigma\rfloor, N-1) \cdot (\lfloor\sigma\rfloor + 1 - \sigma) + \Gamma(\lfloor\sigma\rfloor + 1, N-1) \cdot (\sigma - \lfloor\sigma\rfloor).$$

That is, γ_{typ} is a weighted average of the two values $\Gamma(i, N-1)$ and $\Gamma(i+1, N-1)$ corresponding to the unique interval $[i, i+1)$ which contains σ. The weights used are one minus the distance from σ to i and $i+1$, respectively (see Figure 3). Then, if $\gamma_{typ} > N/(2N-1)$, we scale all the values of γ by a factor $N/(2N-1)/\gamma_{typ}$:

$$\gamma(i) = \begin{cases} \Gamma(i, N-1) \cdot \dfrac{N/(2N-1)}{\gamma_{typ}} & \text{if } \gamma_{typ} > N/(2N-1) \\ \Gamma(i, N-1) & \text{otherwise} \end{cases}.$$

We observe that the simpler choices $\gamma_{typ} = \Gamma(\lfloor\sigma\rfloor, N-1)$ or $\gamma_{typ} = \Gamma(\lceil\sigma\rceil, N-1)$ are prone to problems in the fixed-point iterations. If, in two subsequent iterations, the

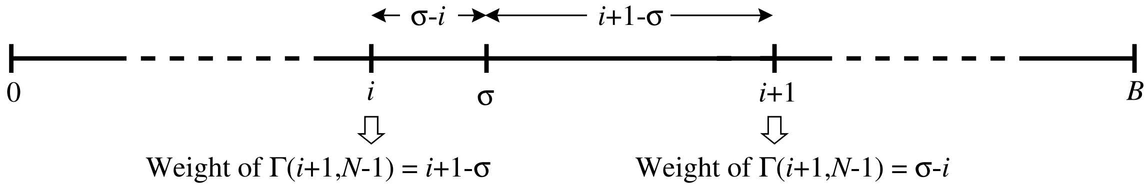

Figure 3: The computation of the weights for SPN-2.

value of σ crosses an integer value, the value of γ_{typ} will experience a jump, due to the discretization imposed by the floor or ceiling operators. Conceivably, this phenomenon could even lead to an oscillating behavior, although we did not encountered this problem in our experiments. A more mundane problem due to these choices is that the plots for the various measures considered in Section 6 would be "jagged", due to the discrete jumps, so we do not consider them further.

Equally poor would be a simple truncation of the values of γ. The choice

$$\gamma(i) = \min\{\Gamma(i, N-1), N/(2N-1)\}$$

results in highly pessimistic results for light loads. This is immediately apparent by observing the values of γ for $\sigma = 0.5$ in Table 4.

Table 2 summarizes the firing rates for the timed transitions in the model of Figure 2. LF is a value between 0% and 100%, representing the percentage load factor on the system. Given that both the insertion of a message into a buffer slot and the use of the crossbar switch require one T_{Msg}, the average time between generation of messages must be greater than this time, $LF < 100\%$, if the system is to be stable. Indeed, given the previous bandwidth analysis for a $N \times N$ switch, it is clear that the system is saturated as soon as LF approaches the value $100 \cdot N/(2N-1)\%$. To model a fully saturated system, we can simply remove transition Gen and place Q from Figure 2, that is, we allow transition $PutHdr$ to fire whenever there are available buffers and place $WtData$ is empty. The truncation parameter K is not required in this case.

5.2 Fixed-point iterations

The value of α represents the probability that a generic port i has at least one header waiting to be routed (we assume uniform message distributions). Its estimation is performed iteratively, starting with a guess and refining it using the assignment

$$\alpha \leftarrow \Pr\{\#(WtRtr) > 0\}.$$

The values of γ are also computed iteratively, starting with a guess for $\mathrm{E}[\#(InSch)]$. Then, again because of symmetry, we set

$$\forall k \in \{1, \ldots, N-1\}, \sigma_k = \sigma \leftarrow \frac{N-2}{N-1} \cdot \mathrm{E}[\#(InSch)]$$

and iterate. That is, the average number of headers from source i ($i \neq 0$) waiting to go to destination j ($j \neq i, j \neq 0$) can be equated, because of symmetry, to the number $\mathrm{E}[\#(InSch)]/(N-1)$ of headers going from 0 to j. The factor $N-2$ is the the number of possible sources, other than 0, that can send headers to destination j (source j does not send to itself).

We observe that our iterative approach would still work even if the arrival rates to the N ports were different or if the choice of the destination port were not uniform (or both). However, instead of solving a single model for a generic port, we must now solve N models differing only in their parameters, one for each port. Hence, we would have to start with $2N$ guesses, one for each of the N values for α_k and σ_k. The tableau computation would still have complexity $O(B^2N)$, although the values $c(i,k,j)$ would be different for each model, to reflect the nonuniform destination distributions.

5.3 Modeling acknowledgments

Certain types of messages might trigger the transmission of an acknowledgment back to the source, upon reception. An acknowledgment behaves just like any other message, with two differences:

- An acknowledgment carries only header information and no data, hence it is much shorter than a message.
- If both messages and acknowledgments are waiting at a source port, acknowledgments are given preference. However, an acknowledgment cannot preempt a message transfer which has already initiated and, inside the switch, it is not granted any special treatment.

The length of an acknowledgment is L_{Ack}. Hence, the time to transfer it into a buffer slot or through the crossbar switch is $T_{Ack} = L_{Ack}/26.6\mu$sec. The amount of time required by the router to route an acknowledgment is still T_{Rtr}, as for messages. Figure 4 and Table 5 describe the corresponding SPN and the firing time distributions for its timed transitions (transitions $GetBuf_m$ and $GetBuf_a$ are immediate, they fire in zero time). We indicate this model with "ACK".

The path for acknowledgements (from transition Gen_a to transition $UseXbar_a$) is exactly analogous to that for messages, with the exception that acknowledgements carry no data (place $WtData_m$ and transition $PutData_m$ have no corresponding place and transition in the acknowledgement path).

The model also contains several inhibitor arcs. The inhibitor arcs from Q_m and Q_a to Gen_m and Gen_a with multiplicity K enforce the required truncation of the state space. We observe that the "cross arcs" from Q_m to Gen_a and from Q_a to Gen_m achieve a different purpose. Given that the firing rate of Gen_m and Gen_a is the same, these inhibitor arcs ensure that the two transitions are either both enabled or both disabled, thus they ensure that the truncation in effect results in the same throughput for the generation of messages and acknowledgements.

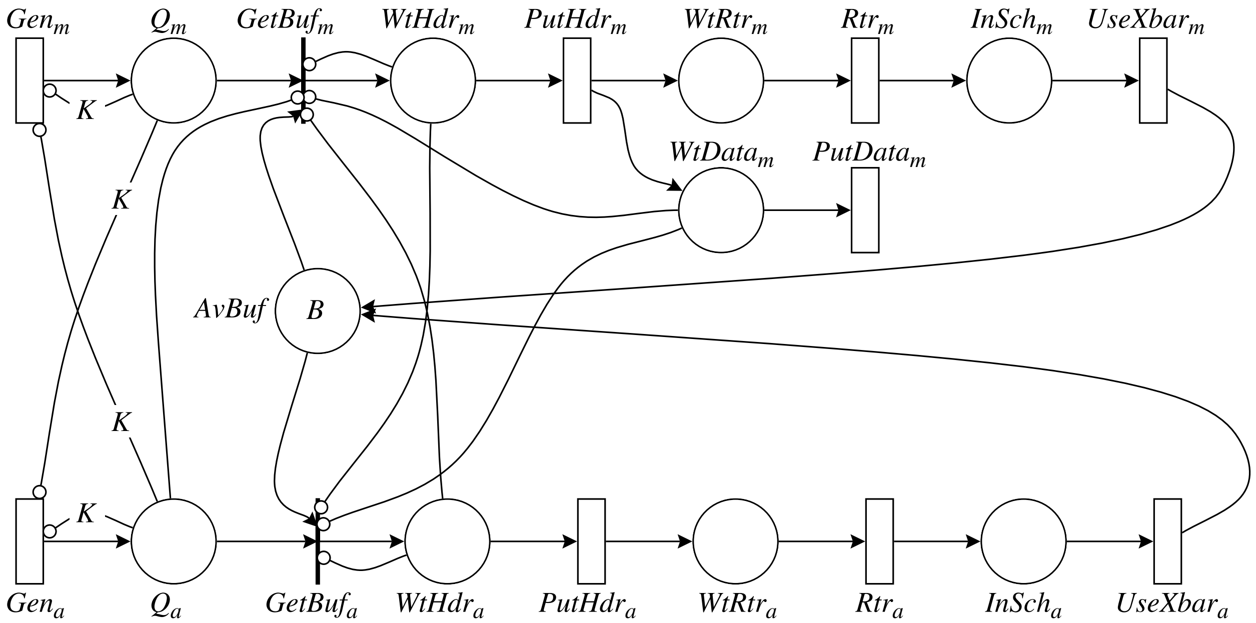

Figure 4: Approximate SPN model with acknowledgments.

The inhibitor arcs from $WtHdr_m$ and $WtHdr_a$ to $GetBuf_m$ and $GetBuf_a$ ensure that at most one token can be in $WtHdr_m$ and $WtHdr_a$ overall at any time:

$$\#(WtHdr_m) + \#(WtHdr_a) \leq 1.$$

Transition	**Firing rate**
Gen_m	$LF/(T_{Msg} + T_{Ack})$
Gen_a	$LF/(T_{Msg} + T_{Ack})$
$PutHdr_m$	$1/T_{Hdr}$
$PutHdr_a$	$1/T_{Hdr}$
$PutData_m$	$1/(T_{Msg} - T_{Hdr})$
Rtr_m	$1/(T_{Rtr}(1 + (N-1)\alpha)) \cdot \#(WtRtr_m)/(\#(WtRtr_m) + \#(WtRtr_a))$
Rtr_a	$1/(T_{Rtr}(1 + (N-1)\alpha)) \cdot \#(WtRtr_a)/(\#(WtRtr_m) + \#(WtRtr_a))$
$UseXbar_m$	$\gamma(\#(InSch_m) + \#(InSch_a))/T_{Msg} \cdot \#(InSch_m)/(\#(InSch_m) + \#(InSch_a))$
$UseXbar_a$	$\gamma(\#(InSch_m) + \#(InSch_a))/T_{Ack} \cdot \#(InSch_a)/(\#(InSch_m) + \#(InSch_a))$

Table 5: Firing rates for the timed transitions in the approximate SPN model with acknowledgments.

Model	Parameters	Distributions	States	Nonzeros
SPN-1/2	$K = 15,\ LF < 100$	Exponential	4,352	15,736
SPN-1/2	$K = 15,\ LF = 100$	Exponential	272	736
ACK	$K = 5$	Exponential	380,920	2,093,690
SPN-1/2	$K = 15,\ LF < 100$	Erlang(10)	41,216	151,960
SPN-1/2	$K = 15,\ LF = 100$	Erlang(10)	3,856	10,620
ACK	$K = 5$	Erlang(2)	730,220	4,035,740

Table 6: Size of the state space for various models and parameter choices.

This, plus the inhibitor arcs from Q_a to $GetBuf_m$, ensures that acknowledgements have nonpreemptive priority over messages when requesting buffers.

5.4 Phase-type distributions and state-space size

If the size of the messages being sent were constant, a more accurate model of the actual system would be obtained by using Erlang distributions, instead of exponential ones, for selected transitions. In the SPN-1 and SPN-2 models, transitions $PutHdr$ and $PutData$ are excellent candidates for this, while transitions Rtr and $UseXbar$ correspond to a number of concurrent activities (now with constant duration), so it might still make sense to use exponential distributions for their firing times. Analogous considerations apply to the ACK model.

However, the increase in the size of the state space is formidable: see Table 6 for a comparison of the sizes of the underlying CTMC (number of states and number of nonzero state-to-state transition rates) for the various models discussed in this paper. Note that the ACK model cannot be simplified when $LF = 100$, since, if Gen_m, Gen_a, Q_m, and Q_a were simply removed the way Gen and Q can be removed in the SPN-1 and SPN-2 models, the throughput of acknowledgments would not necessarily match that of messages, as it should. The column “Distributions” refers only to transitions $PutHdr$ and $PutData$ (or $PutHdr_m$, $PutData_m$, and $PutHdr_a$), all the other transitions retain an exponential distribution.

6 Results

In this section, we report the results for the SPN-1 and SPN-2 models obtained using fixed-point iteration with the software tool SPNP [10], and compared them with the results from a detailed discrete-event simulation.

Knowing that the throughput is bounded by the maximum bandwidth of the switch, we experimented first on the two models to observe how the “requested load factor”, that is the value used for LF, affects the "actual load factor", determined using the throughput of transition Gen:

$$\text{“actual load factor”} = \text{“throughput of transition } Gen\text{”} \cdot T_{Msg} = \Pr\{\#(Q) < K\} \cdot LF.$$

Figure 5 shows the results. As expected, the actual load factor coincides with LF until LF begins approaching the bound $N/(2N-1)$. Then, the actual load factor has a sharp bend and does not increase any more, suggesting that the system is indeed overloaded. In the SPN-1 and SPN-2 models, this simply says that the probability of finding place Q full ($\#(Q) = K$) increases linearly in LF after that point, so that the truncation imposed by the inhibitor arc ensures that the actual load factor remains constant. Note that SPN-1 is slightly more pessimistic than SPN-2, since its horizontal asymptote is lower. This can be explained by the fact that the definition of γ in SPN-1 results in slightly smaller values than in SPN-2. For high loads, the average number of headers in the scheduler table, σ, is quite large (above 14), hence the main contribution to the definition of all the values of γ in SPN-1 is simply the bound $N/(2N-1)$. Under the same conditions, in SPN-2, the value of γ_{typ} is a weighted average of $\Gamma(14, N-1)$ and $\Gamma(15, N-1)$, and all values $\Gamma(i, N-1)$ are reduced so that the new weighted average coincides with $N/(2N-1)$. However, the value of $\gamma(15)$ is sufficiently higher than $N/(2N-1)$ after the normalization to result in a higher overall throughput for the switch.

We study two quantities in particular: the expected number of free buffers and the message latency, defined as the expected amount of time elapsing from the instant a message is generated to the time it is completely transferred to its destination. In the SPN-1 and SPN-2 models, the first one can be easily defined as $\mathrm{E}[\#(AvBuf)]$. The computation of the second quantity must be done using Litttle's law. Compute first the expected number of messages in the system as $e = \mathrm{E}[\#(Q) + \#(WtRtr) + \#(InSch)] = \mathrm{E}[\#(Q) + 15 - \#(AvBuf)]$ (tokens in $WtData$ are not counted, since they are already accounted for by tokens in $WtRtr$ or in $InSch$). Then, compute the throughput τ, defined as a the average number of firings of any of the transitions in the SPN per unit of time (they have all the same throughput). For transition Gen, we have seen that this can be expressed as

$$\tau = \Pr\{\#(Q) < K\} \cdot LF/T_{Msg}.$$

Finally, the message latency w is obtained as $w = e/\tau$.

Figures 6 and 7 show the values of these two measures as a function of the actual load factor, for SPN-1 and SPN-2. In the same figure, these are compared to the results obtained from a detailed simulation model (indicated by "SIM") using the same distributional assumption of exponential timing. In SIM, the N sources and destinations are explicitly represented, and so is the detailed behavior of the router and of the crossbar switch and their interaction through the scheduler table. The simulation results have a 95% confidence interval of less than $\pm 1\%$, using the batch means method. The fixed-point iteration approach approximates the simulation results remarkably well, most of the error lying where the curves experience an abrupt change, going from a light-to-medium load to a heavy load.

We conclude this section with some observations on the behavior of the fixed-point method used. Figures 8 and 9 illustrate the number of iterations and, at the same time, the evolution of the value of the critical iteration parameter $\mathrm{E}[\#(InSch)]$, for various requested load factors. There is a large difference in the number of iterations for SPN-1 (Figure 8) and SPN-2 (Figure 9). We have been unable to explain this phenomenon so far. Also noticeable is the fact that the smallest number of iterations is required for values of LF corresponding to very high or very low loads, even if the value we chose to start the iteration, $\mathrm{E}[\#(InSch)] = 8$ in all cases, is closest to the final value of $\mathrm{E}[\#(InSch)]$ exactly for the cases requiring a large number of iterations. For example, only seven iterations are required to go from $\mathrm{E}[\#(InSch)] = 8$ to $\mathrm{E}[\#(InSch)] = 0.1202$, for $LF = 10$, or from $\mathrm{E}[\#(InSch)] = 8$ to $\mathrm{E}[\#(InSch)] = 14.44$, for $LF = 90$, while 60 iterations are required to go from $\mathrm{E}[\#(InSch)] = 8$ to $\mathrm{E}[\#(InSch)] = 6.126$, for $LF = 50$.

7 Extensions and future work

The FIFO policy can be substituted with other policies, which could select source-destination pairs in the scheduler table not based on their time of arrival, but according to other criteria. We intend to explore a GREEDY policy, which would allow the transfer $3 \mapsto 2$ in the example of Table 1 in addition to $\{1 \mapsto 0, 2 \mapsto 1, 0 \mapsto 3\}$.

Also of interest is an OPTIMAL policy. Such a policy would likely not be implementable in practice, because of its computational overhead and of the need to preempt ongoing transfers to maximize the number of concurrent transfers after each change of state. However, it could provide an ideal upper bound for how well the switch could be scheduled.

Regarding the fixed-point approach used, more work is certainly needed to understand its behavior and applicability. Empirical evidence suggests that it is appropriate when the system being modeled is highly modular and symmetric, but more theoretical work needs to be done to formalize this concept.

Finally, it might be instructive to model a bimodal workload with some short and some constant-sized long messages. In the simulation model, this is trivial. In the SPN model, however, timing delays coefficient of variation less than one are needed, such as the Erlang distributions. 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 [14, 6]. We are currently working on the design of a software package that will allow us to solve numerically 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 [8]).

8 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] D. P. Bhandarkar and S. H. Fuller. Markov chain models for analyzing memory interference in multiprocessor systems. In *Proc. 1st Annual Symp. Comp. Arch.*, pages 1–6, Dec. 1973.

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

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 504–530. Springer-Verlag, 1991.

[6] G. Ciardo. Stochastic Petri nets with discrete timing. In W. J. Stewart, editor, *Numerical Solution of Markov Chains '95*, Raleigh, NC, Jan. 1995. To appear.

[7] 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.

[8] 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.

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

[10] 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.

[11] R. Jain. *The art of computer systems performance analysis.* John Wiley & Sons, 1991.

[12] K. Jensen. Coloured Petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

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

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

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

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

[17] R. B. Rau. Interleaved memory bandwidth in a model of a multiprocessor computer system. *IEEE Trans. Comp.*, 28(9):678–681, Sept. 1979.

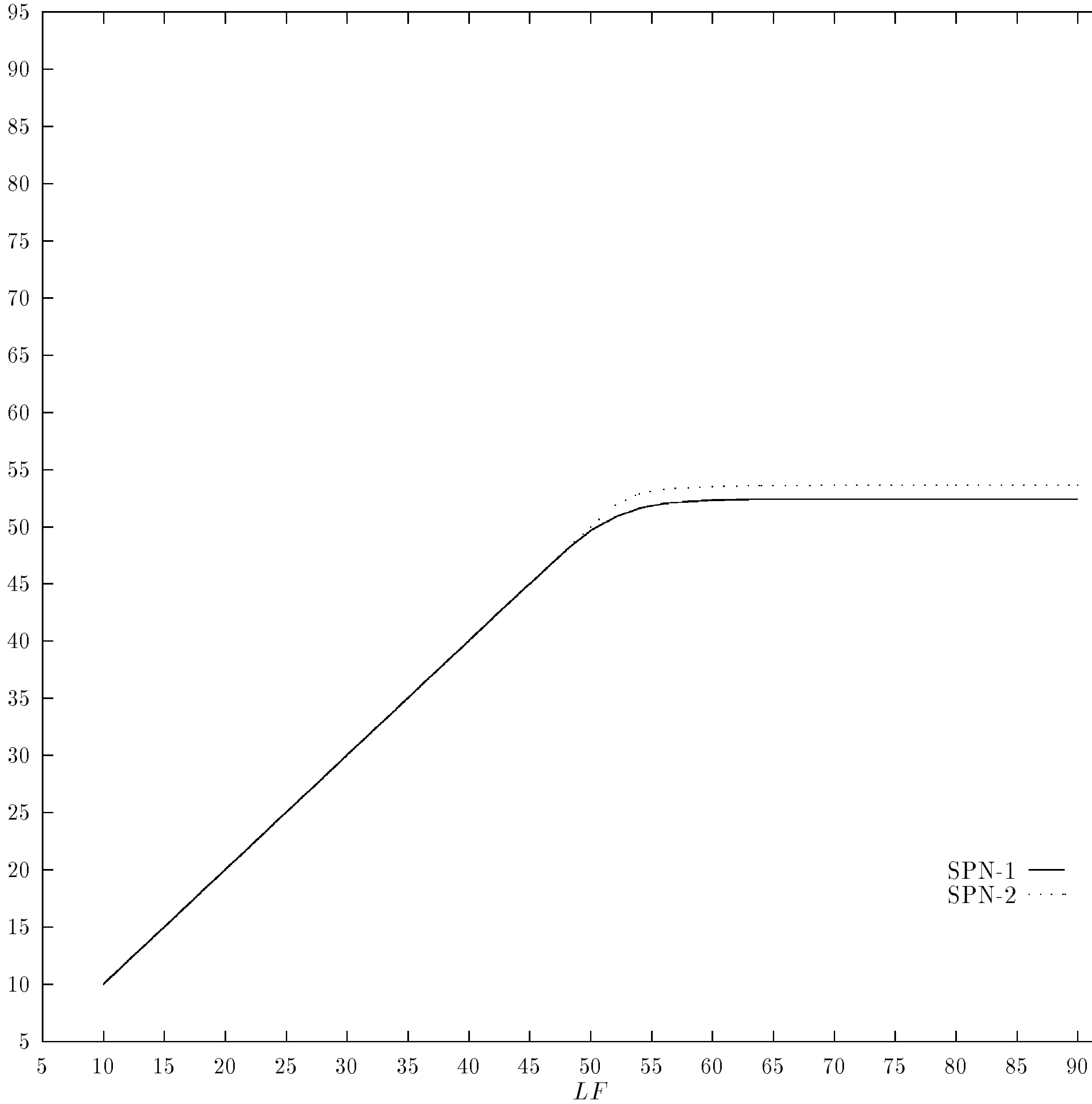

Figure 5: Actual load factor as a function of the requested load factor.

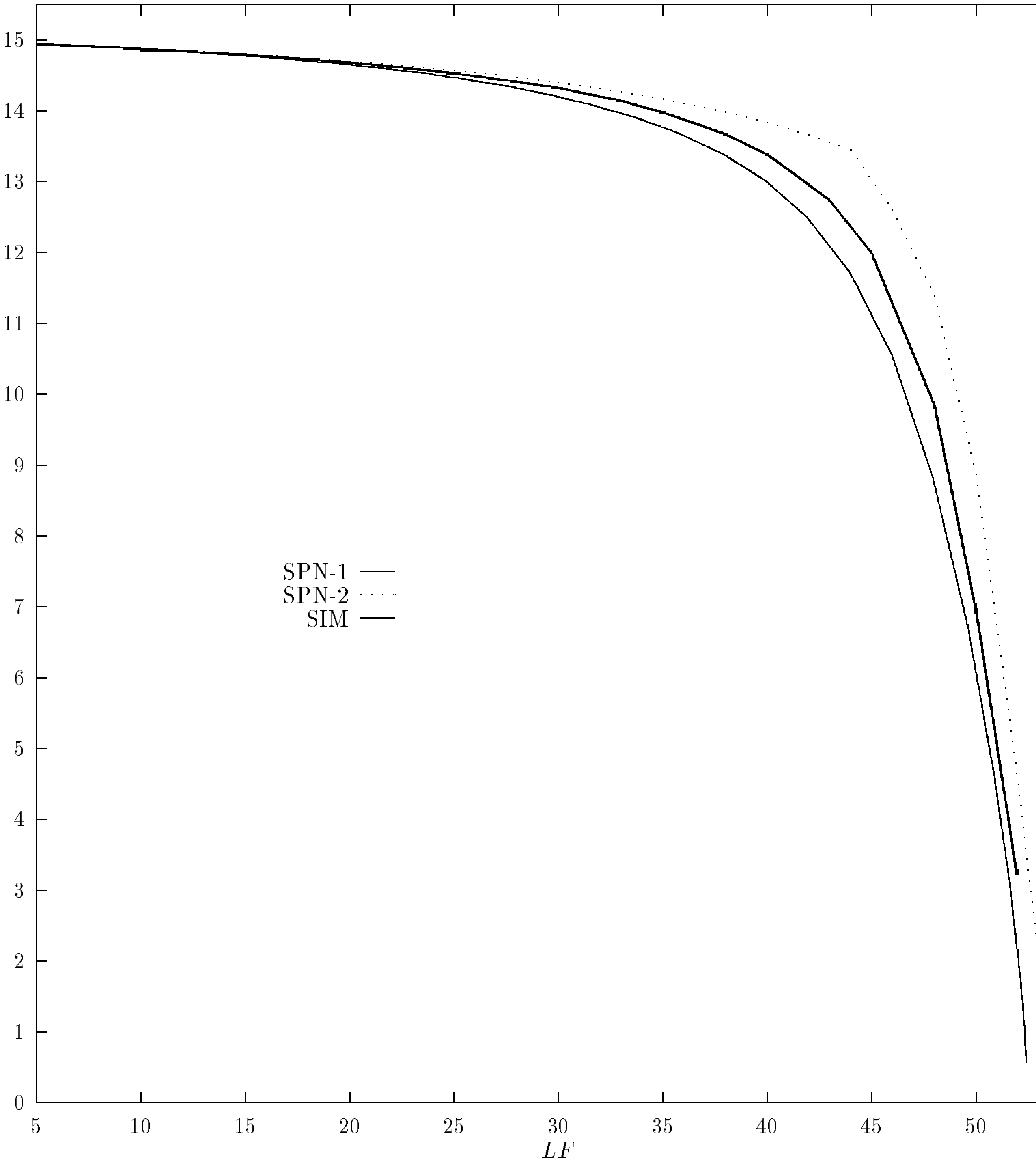

Figure 6: Expected number of free buffers as a function of the actual load factor.

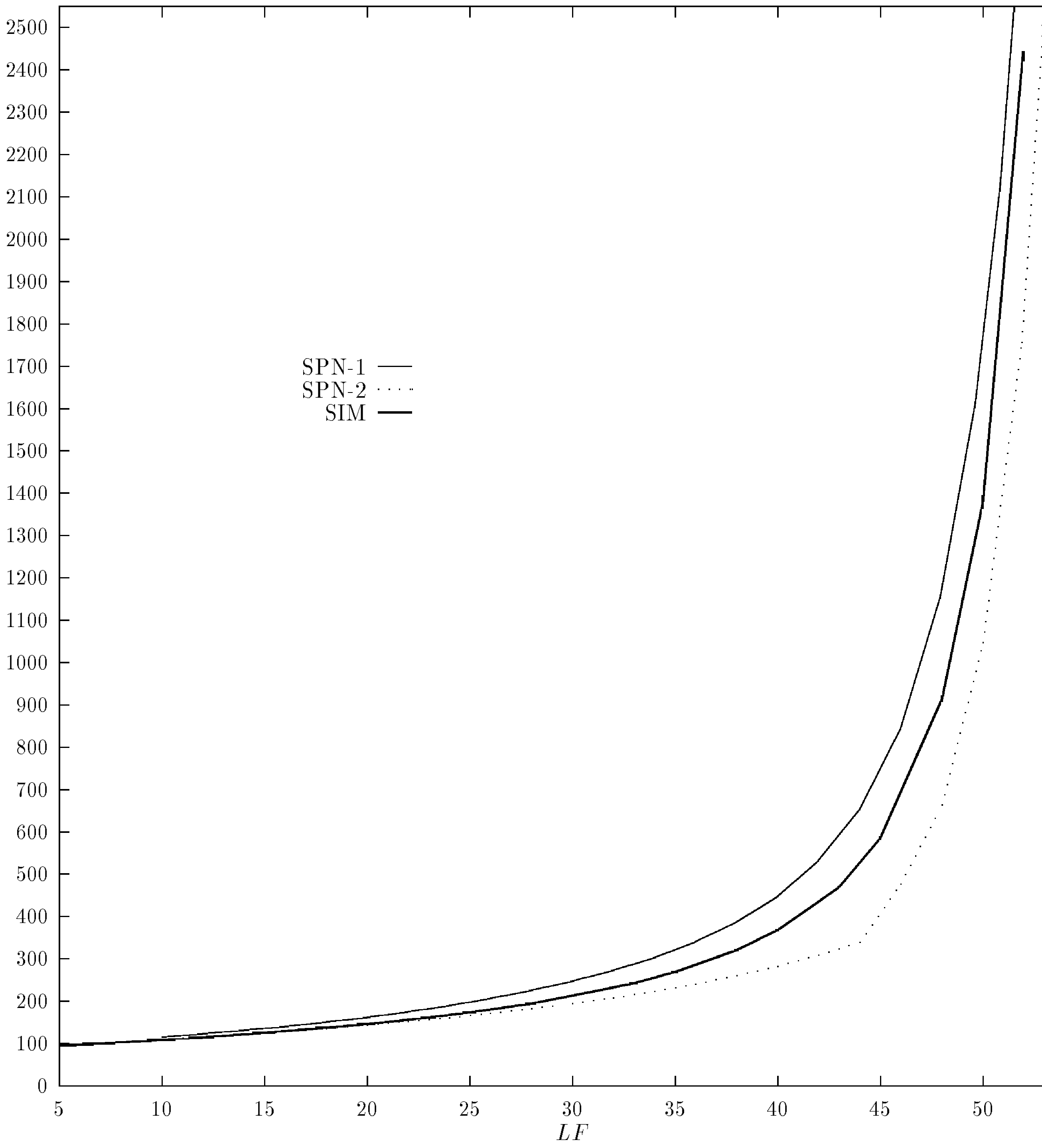

Figure 7: Message latency as a function of the actual load factor.

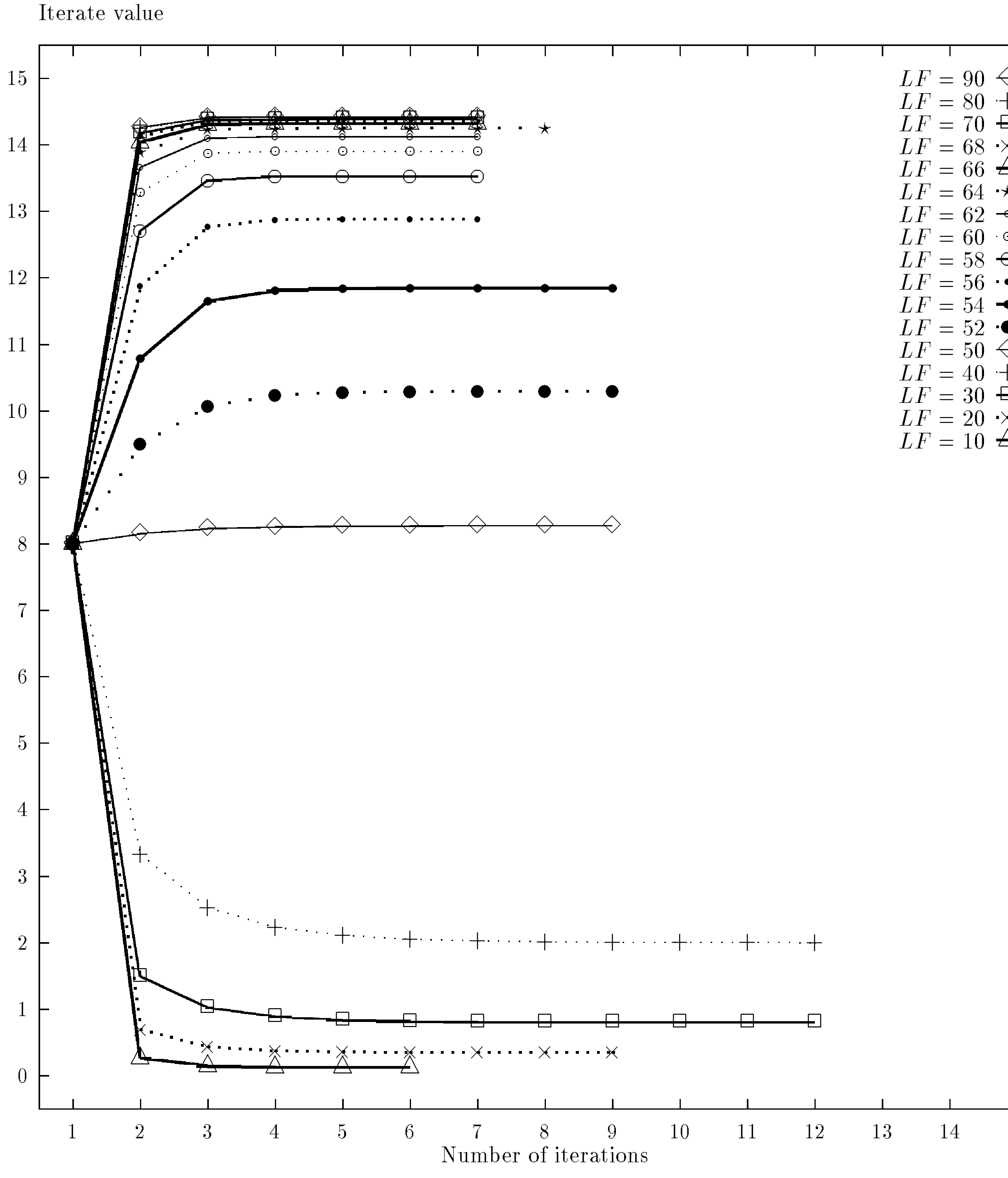

Figure 8: Convergence behavior of E[#($InSch$)] for the fixed-point iterative method (SPN-1).

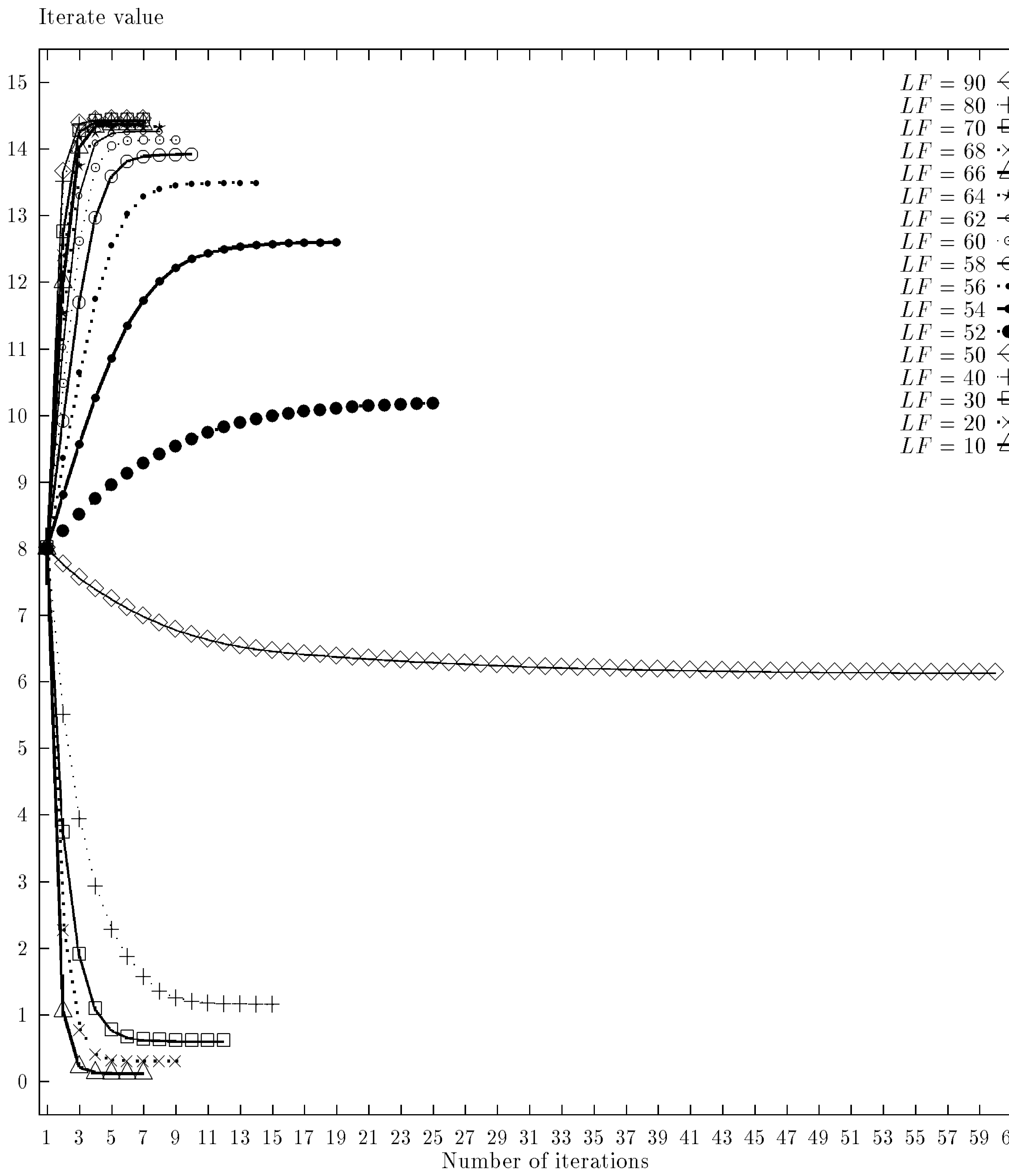

Figure 9: Convergence behavior of E[#($InSch$)] for the fixed-point iterative method (SPN-2).