Buffer Allocation in General Single-Server Queueing Networks

F. R. B. Cruz‡
fcruz@est.ufmg.br
A. R. Duarte†
andersonrdufmg.br
T. van Woensel‡
T.v.Woensel@tm.tue.nl
January 4, 2007

Abstract — The optimal buffer allocation in queueing network systems is a difficult stochastic, non-linear, integer mathematical programming problem. Moreover, the objective function, the constraints or both are usually not available in closed-form, making the problem even harder. A good approximation for the performance measures is thus essential for a successful buffer allocation algorithm. A recently published two-moment approximation formula for predicting the optimal buffer allocation in general service time single queues is examined in details, based on which a new algorithm is proposed for buffer allocation in single-server general service time queueing networks. Computational results are shown to evaluate the efficacy of the approach in generating optimal buffer allocation patterns.

Keywords — Buffer allocation; queues; networks.

1 INTRODUCTION

Manufacturing, telecommunication, and material handling systems are just few examples of practical interest that may be represented by finite buffer queueing networks. Because of the critical costs for buffer space, it is crucial to optimally determine the buffer spaces in order to ensure maximum performance at the lowest possible cost. The buffer allocation problem (BAP) is computationally hard to solve as the BAP is usually formulated as a stochastic, non-linear, integer mathematical programming problem. The BAP is to find optimal values for the buffer sizes $K$ such that the blocking probability $p_K$ is below some pre-specified threshold $\varepsilon$ for all queues in the network. In this article, the buffer allocation problem will focus on networks of $M/G/1/K$ queues, which in Kendall’s notation considers Markovian arrivals, generally distributed service times, one single server, and the total capacity of $K$ items, including the item in service. In this case, the BAP is a complicated problem as it involves general service time queues configured in arbitrary networks [19] as seen in Fig. 1. No closed-form objective functions are available for these types of networks. Thus, one needs to rely on approximations.

One of the objectives of this article is to extensively compare approximations for the blocking probability, $p_K$, the probability that an arriving entity finds the queueing system at its capacity, in general single-server queues. One of the most regarded performance measures of queueing systems, the blocking probability is a building block for buffer allocation formulations. Another objective is to assess the accuracy of a novel implementation [3] for the Generalized Expansion Method (GEM), a well-know method for performance evaluation of finite queueing networks, applied to networks of $M/G/1/K$ queues. Finally, the third objective is to propose a simple algorithm for the buffer allocation problem in general, finite, single-server queueing networks, and to apply it, getting insights into such a challenging network design problem.

This article is organized as follows. In Sec. 2 the BAP is defined as a non-linear mathematical programming formulation and a short literature review is presented on the different algorithms developed in the past for similar problems. Some of the most effective approximations for $p_K$ are extensively compared in Sec. 3. In Sec. 4 the performance evaluation algorithm for finite queueing networks is detailed and its accuracy is attested. Then, Sec. 5 describes the proposed algorithm to solve the BAP. Computational results evaluating the efficacy of the new buffer allocation algorithm are discussed in Sec. 6. Finally, Sec. 7 closes the article with final comments and topics for future research in the area.

2 BUFFER ALLOCATION PROBLEM

2.1 Problem formulation

The BAP is concerned with how much space needs to be allocated in order to guarantee that the probability of loosing clients (or delaying them) is below a certain threshold. In its simplest definition the BAP seeks the lowest integer $K > 0$ such that $p_K \leq \varepsilon$ for some accept-
able threshold \( \varepsilon \in (0, 1) \). It is assumed that the system utilization \( \rho \) (that is, the ratio between the arrival rate and the service rate, \( \rho = \lambda / \mu \)) is below 1.0, because an optimum may not exist for \( K \) if \( \rho \geq 1.0 \) (see Kimura [12]).

The BAP may be defined by a multi-objective non-linear mathematical programming formulation with integer decision variables \( x_i \equiv K_i \) for the \( i \)th \( M/G/1/K \) queue. However in this article only the following single objective formulation will be considered

\[
Z = \min \sum_i x_i, \tag{1}
\]

\[
\text{s.t.}:
\]

\[
\Theta(x) \geq \Theta^{\text{min}}, \tag{2}
\]

\[
x_i \in \mathbb{N}, \forall i, \tag{3}
\]

which minimizes the total buffer allocation to the network, \( \sum_i x_i \), subject to providing a minimum total throughput \( \Theta^{\text{min}} \). In this formulation, \( \Theta^{\text{min}} \) is some threshold throughput, not superior to the total external arrival rate, \( \Lambda = \sum_i \Lambda_i \), and \( x_i \) is the buffer \( K \) allocation to the \( i \)th \( M/G/1/K \) queue, including those in service. Although similar to a linear integer mathematical programming problem, the formulation does not model directly the buffer allocation because \( \Theta(x) \) is a function hard to define, involving the arrival rates, the service rates, and other parameters and variables in the queueing network.

2.2 Literature Overview

The BAP literature can roughly be divided into four methodological approaches: simulation methods, metaheuristics, dynamic programming, and search methods. In the following paragraphs, a short overview of these approaches will be presented.

The simulation methods aim to represent the actual systems by means of robust assumptions. In other words, general probability distributions are used to model the various aspects of the system, such as inter-arrival times, batch size of the arrivals, service times, among others. Simulation methods are usually very general and efficient but the price paid usually is a great computational effort that may reduce the size of treatable instances. However, successful uses of simulation methods have been reported by researchers, such as, for instance, Soyster et al. [21], for series queueing networks, and Baker et al. [2], for general topologies.

Metaheuristics are very popular methods nowadays, mainly because of the increasing computational capacity available. Typical techniques that fall into this area include simulated annealing, tabu search, and more recently, generic algorithms. The advantages of metaheuristics are the absence of all those restrictive assumptions usually required by the traditional methods and the ability of avoiding local optima traps in the seek of the global optimum. The disadvantage is that usually the metaheuristics must be tailored to the special structure of the problem. Among others, a successful case of use was reported by Spinellis et al. [22], for buffer allocation in tandem networks of \( M/M/c/K \) queues.

Dynamic programming is another powerful and reasonable approach for the BAP. Usually the exponential space complexity of dynamic programming methods reduces their applicability to very small size instances. However, the approach has been proved successful in many cases. For instance, Kubat and Sumita [14] and Yamashita and Altiiok [24] reported results for networks of \( M/M/1/K \) queues in series and Yamashita and Onvural [25], for general topologies.

Finally, there are the search methods, which try to solve the problems avoiding the combinatorial explosion of possible solutions by choosing those solutions that are close to the optimum results. Their main disadvantage is their restrictive assumptions, such as concavity and convexity, that may limit the applicability. In the past, search methods were also successful in solving the BAP. In series topologies, the BAP was solved by search methods by Altiiok and Stidham [1], for networks of \( M/M/1/K \) queues, and by Hillier and So [8], for

![Figure 1: Queueing network in an arbitrary topology.](image-url)
3 Blocking Probability

Accurate approximations for the blocking probability for \( M/G/1/K \) systems, \( p_K \), will be presented in the following paragraphs. They are based on finite Markovian systems, \( M/M/1/K \), but approximations based on infinite queueing systems are also common. The relevance of accurate approximations for the BAP is apparent when we take into account that the throughput in a single queue is the following function of the arrival rate and the blocking probability

\[
\theta = \lambda (1-p_K).
\]

### 3.1 Markovian Systems

The blocking probability expression for a finite Markovian system is well-known [7]

\[
p_K = \frac{(1-\rho)\rho^K}{1-\rho^{K+1}},
\]  

for \( \rho \neq 1 \), being possible then to express \( K \) in terms of \( \rho \), the system utilization, and \( p_K \), the blocking probability, as follows

\[
K = \left\lceil \frac{\ln\left(\frac{\rho}{1-\rho\rho}\right)}{\ln(\rho)} \right\rceil,
\]

in which \( \lceil x \rceil \) is the lowest integer not inferior to \( x \), and the blocking probability may be expressed in terms of the threshold throughput, \( \Theta^{\text{min}} \), and the arrival rate, \( \lambda \), as

\[
p_K \leq 1 - \Theta^{\text{min}}/\lambda.
\]

Expression (5) is not only useful for the optimal buffer allocation for individual Markovian queues but it is also useful for networks of general service time queueing systems, as shortly it will be apparent.

### 3.2 Gelenbe’s Approximation

Generally speaking, approximations developed in the past for the blocking probability are based on infinite queues. Actually, many of them could be adapted for \( M/G/1/K \) queues. A survey by Makens [16], for instance, analyzed five different approximations and concluded that Gelenbe’s formula [6] was efficient for the majority of the cases tested. Gelenbe’s approximation is based on an approximation of the discrete queuing process by a continuous diffusion process. The blocking probability is given by

\[
p_k = \frac{\lambda(\mu - \lambda)e^{-\gamma} - \gamma (\mu - \lambda)(k-1)\lambda e^{-\gamma}}{\mu^2 - \lambda^2 e^{-2\gamma}(\mu - \lambda)(k-1)\lambda e^{-\gamma}}.
\]  

in which \( \lambda \) is the arrival rate, \( \mu \), the service rate, \( e^2 = \text{Var}(T_a)/E(T_a)^2 \) is the squared coefficient of variation of the inter-arrival time, \( T_a \), and \( e^2 = \text{Var}(T_s)/E(T_s)^2 \) is the squared coefficient of variation of the service time, \( T_s \). From Eq. (6), it is possible to explicitly get the optimal buffer allocation

\[
K = \frac{2\lambda - 2\mu + \ln \left( \frac{\lambda P_k\rho^2}{\lambda - \mu} + \frac{\rho}{\lambda - \mu} \right) + \ln \left( \frac{\lambda P_k\rho^2}{\lambda - \mu} + \frac{\rho}{\lambda - \mu} \right)}{2(\lambda - \mu)}.
\]

Taking both the squared coefficients of variation of the inter-arrival time and the service time equal to one, \( e^2 = e^2 = 1 \), Eq. (7) results in the optimal buffer allocation of Markovian single queues, \( M/M/1/K \). Notice that the resulting expression will not be exactly the \( M/M/1/K \) formula, Eq. (5), because Gelenbe’s expression is an approximation. However, as noticed by Makens [16] and by Smith and Cruz [19], Gelenbe’s expression is accurate for Markovian system, while is not accurate for deterministic service time systems, \( M/D/1/K \).

### 3.3 Two-moment Approximation

The two-moment approximation scheme is based on a weighted combination of some approximation for the optimal buffer expressions for Markovian systems, \( M/M/1/K \), denoted by \( K^M \), and for deterministic service time systems, \( M/D/1/K \), denoted by \( K^D \). Tijm’s formula [23] is one two-moment approximation that has been shown to be very good in practice. It is given by

\[
K^T_{\text{Tijm}}(e^2) = e^2 K^M + (1-e^2) K^D,
\]

for \( e^2 \geq 0 \). Clearly, Tijm’s formula is exact for the extreme cases, i.e., \( e^2 = 0 \) and \( e^2 = 1 \), if exact expressions are known for \( K^e \) and \( K^D \).

Kimura’s formula [13] is another good two-moment approximation, which is a little simpler as it uses as a basis only an approximation for the optimal pure buffer expression of Markovian systems

\[
B^K_{\text{Kimura}}(e^2) = B^M + \text{NINT} \left[ \frac{(e^2-1)}{2} \sqrt{\mu} B^M \right],
\]

\[
B^M = \frac{\lambda(\mu - \lambda)e^{-\gamma} - \gamma (\mu - \lambda)(k-1)\lambda e^{-\gamma}}{\mu^2 - \lambda^2 e^{-2\gamma}(\mu - \lambda)(k-1)\lambda e^{-\gamma}}.
\]
in which $\text{NINT}[x]$ is the nearest integer to $x$. Important to say about Kimura’s formula is that it estimates the pure buffer without the space for the customers in service (that is, $B = K - 1$, for $M/G/1/K$ systems), while Tijm’s formula includes those in service.

Recently, Smith [17] proposed the following two-moment approximation for $M/G/1/K$ queues, based on Kimura’s formula

$$B_s^{\text{Smith}}(c^2_s) = \left[ \frac{\ln \left( \frac{PK}{1 - \rho} \right)}{\ln(\rho)} \right]_{B_s^{M}=K^{M-1}} - 1 + \left( c^2_s - 1 \right) \frac{1}{2} \sqrt{\rho} \left[ \frac{\ln \left( \frac{PK}{1 - \rho} \right)}{\ln(\rho)} \right]_{B_s^{M}=K^{M-1}} - 1 \right)$$

in which expression (5), subtracted by the space for the single server, is used as the estimate for the optimal pure buffer allocation of Markovian systems, $B_s^{M}$. Now, factoring the terms of the approximation, the following simplified expression for the optimal buffer size in $M/G/1/K$ is given

$$B_s^{\text{Smith}}(c^2_s) = \left[ \frac{\ln \left( \frac{PK}{1 - \rho} \right) - \ln(\rho)}{2 \ln(\rho)} \right] \left( 2 + \sqrt{\rho c^2_s} - \sqrt{\rho} \right)$$

Notice also that Eq. (11) yields the same expression as Eq. (5), if $c^2_s = 1$ and the space for the server is added. Additionally, as a side effect of Eq. (11), it is possible to obtain a closed-form approximate expression for the blocking probability of single $M/G/1/K$ queues

$$p_K = \rho \left( \frac{2 + \sqrt{\rho c^2_s - \sqrt{\rho} (2K - 1)}}{2 + \sqrt{\rho c^2_s - \sqrt{\rho}}} \right) (-1 + \rho)$$

As it will be seen in the following sections, Eq. (12) will be useful for computing performance measures of queueing networks of $M/G/1/K$ systems.

### 3.4 Computational Experiments

A series of computational experiments was performed to test the efficacy of the blocking probabilities given by the Markovian formula, Eq. (4), Gelenbe’s formula, Eq. (6), and Smith’s formula, Eq. (12). For the buffer sizes the values $K \in \{2, 4, 8, 16\}$ were considered. For each one of the buffer sizes, Markovian, $c^2_s = 1.0$, hyperexponential, $c^2_s = 0.5$, and hyperexponential service time systems, $c^2_s = 2.0$, were tested. Because no exact blocking probabilities were available, the results had to be compared with simulations, obtained from Gamma distributions, for the general service times, and 20,000 simulated time units to approach steady state. ARENA was the simulation system employed (for details, see Kelton et al. [10]). The simulation results presented are averages from 30 replications. The mean standard errors are too small to be noticed in the graphs presented in Fig. 2–Fig. 4. Some studies have been published for $M/G/1/K$ single queues [17, 19] but not at the extend seen in this article.

### Markovian Systems

Results for the first set of experiments, done for Markovian systems, that is, $c^2_s = 1.0$, are presented in Fig. 2. These experiments were performed to validate the implementations as all of them should yield the same results, which they indeed do in most of the cases. Actually, only for $K = 2$ and $\rho < 1.0$ divergences were noticed involving Gelenbe’s formula. Notice that as $K$ increases the blocking probabilities get close to zero when $\rho < 1.0$, which is a logical and expected behavior.

### Hyperexponential Systems

The results for hyperexponential systems, with $c^2_s = 0.5$, are available in Fig. 3. For hyperexponential systems the Markovian approximation is an upper bound for the blocking probabilities, as it always overestimates the simulation results, assumed here as reference. Thus, it is clear that by simply using Markovian approximations for hyperexponential systems one will tend to allocate larger buffer spaces than necessary.

Taking again the simulations as reference, Gelenbe’s approximation underestimates the blocking probabilities when the system utilization is below unity but tends to overestimate them otherwise. On the other hand, Smith’s approximation seems to be more accurate than Gelenbe’s approximation and less dependent on $\rho$. For large buffer sizes, the blocking probabilities tend to be zero for those cases in which the system utilization is below unity, $\rho < 1.0$. However, it is noticeable that although the approximations may disagree considerably for small buffer sizes they all tend to produce similar estimates as the buffer size increases.

### Hyperexponential Systems

Results for hyperexponential systems, with $c^2_s = 2.0$, are presented in Fig. 4. The Markovian approximations may be seen as a lower bound for the blocking probabilities, as their values always underestimate the simulation results, taken here as references. The inadequacy of Markovian approximations for hyperexponential systems for optimal buffer allocation purposes is confirmed. In this case, one will allocate less buffer space than necessary.

In comparison with the simulations results, Gelenbe’s approximations overestimate the blocking probabilities...
Figure 2: Comparisons for $p_K$ for Markovian systems.
Figure 3: Comparisons for $p_K$ for hypoexponential systems with $c_2 = 0.5$. 
Figure 4: Comparisons for $p_K$ for hyperexponential systems with $c_2^2 = 2.0$. 
just in the most appropriate range of $\rho$ (for system utilization less than the unity, $\rho < 1.0$). By its side, Smith’s approximation presents estimates close to the simulation results independent on the $\rho$.

As observed for hypoexponential systems, the blocking probabilities tend to be close to zero for system utilization below the unity as the buffer size increases. Also similarly to hypoexponential systems, all approximations tend to agree for large buffer size systems.

4 Performance Evaluation Algorithm

4.1 Generalized Expansion Method

Notice that, in order to solve the optimization problem given by Eq. (1), (2), and (3), one will need an estimate for the throughput, $\Theta(x)$. An algorithm available is the Generalized Expansion Method (GEM), successfully used in the past to estimate performance measures for arbitrarily configured finite queueing networks.

```
algorithm G(V, A, P, \Lambda, \mu, c^2)
/* preempt all nodes */
\lambda_i \leftarrow \Lambda \forall i \in V
Q \leftarrow \emptyset
while Q \neq V
  choose j \in (V \setminus Q)
  if i \in Q, \forall (i,j) \in A then
    /* evaluate performance for node j */
    compute $p_{K_j}$
    compute $\theta_j = \lambda_j \times (1 - p_{K_j})$
    /* forward information */
    for all $l$, such that $(j,l) \in A$ do
      $\lambda_l \leftarrow \lambda_l + p_{(j,l)} \times \theta_j$
    end for
    /* update set Q */
    Q \leftarrow Q \cup \{j\}
  end if
end while
/* reevaluate all nodes */
\bar{Q} \leftarrow \emptyset
while \bar{Q} \neq V
  if j \in Q, \forall (i,j) \in A then
    /* update performance measure */
    $E[T_{i,j}] \leftarrow \min E[T_{i,j}];$
    s.t.: $\theta_i \leq \theta_{\text{max}}$
    $E[T_{i,j}] \geq \frac{1}{\mu_i}$
    $p_{K_i} = \mu_i - \lambda_i \times (1 - p_{K_i})$
    Eq. (12)
    $\theta_i \leftarrow \lambda_i \times (1 - p_{K_i})$
  end if
end while
/* label node as evaluated */
Q \leftarrow Q \cup \{k\}
end if
/* write final results */
write $p_{K_i}, \theta_i$ for all $i \in V$
end algorithm
```

Figure 5: Algorithm for performance evaluation.

Well described in many articles, in particular in the recently published article by Kerbache and Smith [11], the GEM is basically a combination of node-by-node decomposition and repeated trials, in which each queue is analyzed separately and then corrections are made in order to take into account the interrelation between the queues in the network. The GEM uses type I blocking, that is, the upstream node gets blocked if the service on a customer is completed but it cannot move downstream due to the queue at the downstream node being full. This is sometimes referred to as blocking after service, which is prevalent in most production and manufacturing, transportation, and similar systems. The implementation used here in this work is based on a recently proposed implementation of the GEM by Cruz and Smith [3], suitable for light to moderate traffic, and, for the first time, used in networks of $M/G/1/K$ queues. The algorithm may be seen in Fig. 5.

The GEM starts by reading all relevant information from the network under analysis, including the set of vertexes in the networks, $V$, the set of arcs, $A$, and the routing matrix, $P \equiv \{p_{(i,j)}\}$, which defines the probabilities of an entity to choose one or another path. As it is seen in the article by Kerbache and Smith [11], the GEM consists on creating for each finite queue following another finite queue (see Fig. 6), represented by vertex $j$, an auxiliary vertex $h_j$, modeled as an $M/G/\infty$ queue. When an entity arrives at the system, vertex $j$ may be blocked with probability $p_{K_j}$, or unblocked, with probability $(1 - p_{K_j})$. Under blocking, the entities are rerouted to vertex $h_j$ for a delay while node $j$ is busy. Vertex $h_j$ helps to accumulate the time an entity has to wait before entering vertex $j$ and to compute the effective arrival rate to vertex $j$.

Thus, the algorithm chooses an arbitrary node, $j$, from set $V$ but not from set $Q$ (in which $Q$ is the set of nodes already evaluated), such that for all arc $(i,j) \in A$, vertex $i$ has been evaluated already. Then, vertex $j$ has computed its blocking probability $p_{K_j}$, from Eq. (12), and its arrival rate, from $\theta_j = \lambda_j \times (1 - p_{K_j})$, $\lambda_j = \Lambda_j + \sum_{j} \lambda_j$. These service rates are then forwarded as arrival rates to the downstream nodes (if they exist), and vertex $j$ is included into set $Q$. For instance, in the network illustrated in Figure 1, a possible valid sequence to perform pre-evaluations is $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 6$.

Notice that the GEM also includes a reevaluation step, designed to guarantee flow conservation, that is, $\theta_j \leq \lambda_j + \sum_{k \epsilon \eta(i,j) \in A} \theta_i p_{ij}$, for all $j \in V$. The reevaluation step is a labeling algorithm working in reverse. For the network presented in Figure 1, a possible valid sequence to perform the reevaluations is $6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$, because a node can only be reevaluated if all of its successors were reevaluated already. The reevaluation algorithm corrects the estimates by means of adjustments in the expected service time of each node $i$, $E[T_{i,j}]$. Further details will not be given in this article.

The interested reader may find all of the details in the recent article by Cruz and Smith [3].
4.2 Computational Experiments

A series of computational experiments was performed to attest for the accuracy of the proposed implementation of the GEM for networks of \( M/G/1/K \) queues. However, only networks of \( M/G/1/2 \) queues were considered because those are the most critical cases, since all the approximations tend to agree for large buffer systems, as shown in Section 3.

We run experiments in three basic topologies, series, merge, and split, combined into several values for the system utilization, \( \rho \), and for the squared coefficient of variation, \( c^2 \). As long as no exact results were available for the configurations tested, the results were compared with simulations. ARENA was the simulation system employed (for details, see Kelton et al. [10]). For the non-Markovian service times (that is, \( c^2 = \{0.5, 2.0\} \)), we used a 2-stage gamma distribution, with convenient settings for the shape and scale parameters. In order to assure the steady-state regime, 200,000 time units were used as the simulation times with a warm-up period of 2,000 time units. The simulation results presented are the averages over 20 replications, to get reduced mean standard errors. Slightly longer and shorter simulations and replications were tested but the results (not shown) did not change significantly.

Table 1 presents the results for the experiments, obtained from a PC Pentium(R) 4 CPU 3.00GHZ, 960 MB RAM running the Microsoft(R) Windows XP. In the column labeled ‘analytical’, we give the throughput result from the GEM for each of the cases. We then compare this analytical result with the average result obtained via the simulation. The column \( \delta \) refers to the half-width of the 95% confidence interval. Notice that the Monte Carlo errors were kept quite small. Also included in these tables is the \( \% \) deviation for the analytical results on the throughput, column \( \Delta \% \theta \), from which we can see that the analytical results may be quite far away from the simulation (exact) results (see results in boldface in Table 1). Mainly the results from this GEM implementation get worse as the system gets overloaded. Notice that for the split topology the results are quite good, because the flow in excess is rejected right away in the first node being then divided into two nodes with equal service rate. Also noticeable is that quality of the approximations is quite dependent on the squared coefficient of variation of the service time. In fact, the \( \% \) deviation finds its highest values with the highest \( c^2 \).

Concluding, from the simulation CPU times reported for a single evaluation of the performance measures for the queueing networks, it is apparent that simulation based techniques may not be quite effective for optimization purposes of these queueing networks, unless the system is very small, because typically hundreds or thousands of performance evaluations may be required for the optimization algorithms. On the other hand, the analytical results are shown to be quite reliable and satisfactory. Additionally, assuming that this new implementation of the GEM is for optimization purposes, we should not worry too much about these high deviations under overloaded traffic.

5 BUFFER ALLOCATION ALGORITHM

5.1 A Lagrangean Relaxation Approach

The optimization problem that will be examined here is given by Eq. (1), (2), and (3). In the formulation, \( x_i \) becomes the decision variable under optimization control, that is, \( x_i \equiv K_i \), for the \( i \)th queue.

A possible way to solve the problem is through the Lagrangean relaxation, a technique that consists in relaxing the complicating constraints and including them in the objective function as a penalty. Among the classical references to the Lagrangean relaxation, the article by Fisher [5] could be cited. A recently published tutorial about the Lagrangean relaxation by Lemaréchal [15] is another reference for the technique.

Thus, one way to incorporate the throughput constraint is through a penalty function. Defining a dual variable \( \alpha \) and relaxing the constraint (2), the following penalized problem is given

\[
L(\alpha) = \min \left\{ \sum_{i=1}^{N} x_i + \alpha \left( \Theta^{\text{min}} - \Theta(x) \right) \right\} \leq 0
\] (13)
Table 1: Results for the generalized expansion method.

### Series Topology

<table>
<thead>
<tr>
<th>ρ</th>
<th>$c_2^s$</th>
<th>$\theta$</th>
<th>$\delta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>CPU*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>0.9783</td>
<td>0.9928</td>
<td>0.0011</td>
<td>1.5</td>
<td>0.9783</td>
<td>0.9928</td>
<td>0.0011</td>
<td>1.5</td>
<td>0.9783</td>
<td>0.9928</td>
</tr>
<tr>
<td>1.0</td>
<td>0.9737</td>
<td>0.9910</td>
<td>0.0008</td>
<td>1.7</td>
<td>0.9737</td>
<td>0.9910</td>
<td>0.0008</td>
<td>1.7</td>
<td>0.9737</td>
<td>0.9910</td>
<td>0.0008</td>
</tr>
<tr>
<td>2.0</td>
<td>0.9643</td>
<td>0.9873</td>
<td>0.0008</td>
<td>2.3</td>
<td>0.9643</td>
<td>0.9873</td>
<td>0.0008</td>
<td>2.3</td>
<td>0.9643</td>
<td>0.9873</td>
<td>0.0008</td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>1.8530</td>
<td>1.9477</td>
<td>0.0018</td>
<td>4.9</td>
<td>1.8530</td>
<td>1.9477</td>
<td>0.0018</td>
<td>4.9</td>
<td>1.8530</td>
<td>1.9477</td>
</tr>
<tr>
<td>1.0</td>
<td>1.8225</td>
<td>1.9348</td>
<td>0.0013</td>
<td>5.8</td>
<td>1.8225</td>
<td>1.9348</td>
<td>0.0013</td>
<td>5.8</td>
<td>1.8225</td>
<td>1.9348</td>
<td>0.0013</td>
</tr>
<tr>
<td>2.0</td>
<td>1.7675</td>
<td>1.9072</td>
<td>0.0011</td>
<td>7.3</td>
<td>1.7675</td>
<td>1.9072</td>
<td>0.0011</td>
<td>7.3</td>
<td>1.7675</td>
<td>1.9072</td>
<td>0.0011</td>
</tr>
</tbody>
</table>

### Split Topology

<table>
<thead>
<tr>
<th>ρ</th>
<th>$c_2^s$</th>
<th>$\theta$</th>
<th>$\delta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>CPU*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>0.9904</td>
<td>0.9930</td>
<td>0.0010</td>
<td>0.3</td>
<td>0.9930</td>
<td>0.9962</td>
<td>0.0032</td>
<td>1.3</td>
</tr>
<tr>
<td>1.0</td>
<td>0.9884</td>
<td>0.9912</td>
<td>0.0010</td>
<td>0.3</td>
<td>0.9912</td>
<td>0.9944</td>
<td>0.0032</td>
<td>1.3</td>
<td>0.9884</td>
</tr>
<tr>
<td>2.0</td>
<td>0.9842</td>
<td>0.9873</td>
<td>0.0009</td>
<td>0.3</td>
<td>0.9873</td>
<td>0.9905</td>
<td>0.0032</td>
<td>1.3</td>
<td>0.9842</td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>1.9322</td>
<td>1.9479</td>
<td>0.0017</td>
<td>0.8</td>
<td>1.9479</td>
<td>1.9655</td>
<td>0.0076</td>
<td>1.6</td>
</tr>
<tr>
<td>1.0</td>
<td>1.9173</td>
<td>1.9352</td>
<td>0.0014</td>
<td>0.9</td>
<td>1.9352</td>
<td>1.9528</td>
<td>0.0076</td>
<td>1.6</td>
<td>1.9173</td>
</tr>
<tr>
<td>2.0</td>
<td>1.8892</td>
<td>1.9121</td>
<td>0.0012</td>
<td>1.2</td>
<td>1.9121</td>
<td>1.9297</td>
<td>0.0076</td>
<td>1.6</td>
<td>1.8892</td>
</tr>
<tr>
<td>4.0</td>
<td>0.5</td>
<td>3.5683</td>
<td>3.6475</td>
<td>0.0020</td>
<td>2.2</td>
<td>3.6475</td>
<td>3.6651</td>
<td>0.0076</td>
<td>4.4</td>
</tr>
<tr>
<td>1.0</td>
<td>3.4851</td>
<td>3.5778</td>
<td>0.0016</td>
<td>2.6</td>
<td>3.5778</td>
<td>3.5955</td>
<td>0.0076</td>
<td>4.4</td>
<td>3.4851</td>
</tr>
<tr>
<td>2.0</td>
<td>3.3506</td>
<td>3.4542</td>
<td>0.0017</td>
<td>3.0</td>
<td>3.4542</td>
<td>3.4729</td>
<td>0.0076</td>
<td>4.4</td>
<td>3.3506</td>
</tr>
</tbody>
</table>

### Merge Topology

<table>
<thead>
<tr>
<th>ρ</th>
<th>$c_2^s$</th>
<th>$\theta$</th>
<th>$\delta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>$\theta$</th>
<th>$\Delta %$</th>
<th>CPU*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>0.3995</td>
<td>0.3990</td>
<td>0.0007</td>
<td>-0.1</td>
<td>0.3990</td>
<td>0.3986</td>
<td>0.0008</td>
<td>1.3</td>
</tr>
<tr>
<td>1.0</td>
<td>0.3994</td>
<td>0.3995</td>
<td>0.0008</td>
<td>0.0</td>
<td>0.3995</td>
<td>0.3991</td>
<td>0.0008</td>
<td>0.0</td>
<td>0.3994</td>
</tr>
<tr>
<td>2.0</td>
<td>0.3992</td>
<td>0.3991</td>
<td>0.0006</td>
<td>0.0</td>
<td>0.3991</td>
<td>0.3987</td>
<td>0.0008</td>
<td>0.0</td>
<td>0.3992</td>
</tr>
<tr>
<td>2.0</td>
<td>0.5</td>
<td>0.7961</td>
<td>0.7956</td>
<td>0.0010</td>
<td>-0.1</td>
<td>0.7956</td>
<td>0.7951</td>
<td>0.0010</td>
<td>4.3</td>
</tr>
<tr>
<td>1.0</td>
<td>0.7953</td>
<td>0.7948</td>
<td>0.0013</td>
<td>-0.1</td>
<td>0.7948</td>
<td>0.7943</td>
<td>0.0013</td>
<td>4.3</td>
<td>0.7953</td>
</tr>
<tr>
<td>2.0</td>
<td>0.7936</td>
<td>0.7920</td>
<td>0.0010</td>
<td>-0.2</td>
<td>0.7920</td>
<td>0.7915</td>
<td>0.0010</td>
<td>4.3</td>
<td>0.7936</td>
</tr>
<tr>
<td>4.0</td>
<td>0.5</td>
<td>1.5718</td>
<td>1.5682</td>
<td>0.0012</td>
<td>-0.2</td>
<td>1.5682</td>
<td>1.5668</td>
<td>0.0012</td>
<td>13.3</td>
</tr>
<tr>
<td>1.0</td>
<td>1.5655</td>
<td>1.5575</td>
<td>0.0010</td>
<td>-0.5</td>
<td>1.5575</td>
<td>1.5560</td>
<td>0.0010</td>
<td>13.3</td>
<td>1.5655</td>
</tr>
<tr>
<td>2.0</td>
<td>1.5530</td>
<td>1.5334</td>
<td>0.0012</td>
<td>-1.3</td>
<td>1.5334</td>
<td>1.5319</td>
<td>0.0012</td>
<td>13.3</td>
<td>1.5530</td>
</tr>
</tbody>
</table>

*simulation CPU time in minutes.
Notice that for any feasible vector \( x \) — that is, a vector \( x \) for which the constraints (2) and (3) hold — the term \( \alpha (\Theta_{\text{min}} - \Theta(x)) \) must be non-positive and will be a penalty of the objective function related to the difference between the threshold throughput, \( \Theta_{\text{min}} \), and the effective throughput, \( \Theta(x) \). Thus, it follows that \( L(\alpha) \leq Z \), that is, \( L(\alpha) \) is an inferior limit for \( Z \), the optimal solution for the BAP, since removing the constraint (2) cannot increase the optimal value \( Z \) (for a detailed discussion on this issue, the reader is referred to the article by Fisher [5]).

As a way to approximately solve the BAP we propose to set the threshold throughput \( \Theta_{\text{min}} \) exactly to the total external arrival rate \( \Lambda \), which will then serve as the input to the performance evaluation algorithm, described in the earlier section, which will compute the corresponding throughput \( \Theta(x) \). Thus, under the assumption that the threshold throughput \( \Theta_{\text{min}} \) is exactly the total external arrival rate \( \Lambda \), the best (highest) possible inferior limit is given by Theorem 1.

**Theorem 1** If \( \Theta_{\text{min}} \) is exactly the total external arrival rate \( \Lambda \), then the highest inferior limit, \( L(\alpha^*) = \max_{\alpha \geq 0} L(\alpha) \), is achieved for \( \alpha^* \rightarrow \infty \).

**Proof:** It follows from \( \Theta(x) \) being a non-decreasing function of \( x \), as it is seen in Fig. 7, and also from the Lagrangean function, \( L(\alpha) \), which is the minimum of linear functions of \( \alpha \),

\[
L(\alpha) = \min \left\{ \sum_{i=1}^{N} x_i + \alpha \left( \Theta_{\text{min}} - \Theta(x) \right) \right\},
\]

with non-negative intercepts and non-negative slopes with

\[
\lim_{x \to \infty} \left( \Theta_{\text{min}} - \Theta(x) \right) = 0,
\]

which results in a non-decreasing convex envelopment, as it is seen in Fig. 8.

The best Lagrangean multiplier \( \alpha \), as defined by Theorem 1, is not practical because one would need that

\[
\left( \Theta_{\text{min}} - \Theta(x) \right) = 0,
\]

which yields \( x_i \to \infty, \forall i \). On the other hand, if a small difference, say \( \left( \Theta_{\text{min}} - \Theta(x) \right) = \varepsilon \), is acceptable, it must hold that

\[
\alpha \left( \Theta_{\text{min}} - \Theta(x) \right) \leq 1,
\]

because, otherwise, it would be better to spend one more unity of buffer space to some \( i \)th queue, \( x_i \), to increase \( \Theta(x) \) (remind that \( \Theta(x) \) is a non-decreasing function of \( x \)). Thus, it is possible to define a corresponding \( \alpha_\varepsilon \) as follows

\[
\alpha_\varepsilon \leq 1/\left( \Theta_{\text{min}} - \Theta(x) \right),
\]

which, assuming \( \left( \Theta_{\text{min}} - \Theta(x) \right) \leq 10^{-3} \), yields \( \alpha_\varepsilon = 10^3 \).

### 5.2 Search Algorithm

The Lagrangean relaxation of the BAP, \( L(\alpha) \), plus an additional relaxation of the integrity constraints for \( x_i \) is a classical unconstrained optimization problem. Among all possible algorithms to solve the BAP, a derivative free search algorithm was used, which is seen in Fig. 9, for its simplicity, and also efficiency, as it will be seen.

The algorithm starts by reading the inputs, that is, the number of vertexes in the networks, \( V \), the number of arcs, \( A \), the routing matrix \( P \equiv [p_{(i,j)}] \), which defines...
the probabilities of an entity to choose one or another path. Also read are the vector of external arrival rates, \( \Lambda \), service rates, \( \mu_i \), squared coefficient of variation of service rates, \( c_i^2 \), and an initial buffer allocation vector, \( x^{(0)} \). With these values, the algorithm take the objective function

\[
 f(x) = \sum_{i=1}^{N} x_i + \alpha (\Theta_{\text{min}} - \Theta(x)),
\]

which is optimized only in relation to the first coordinate of vector \( x \), keeping fixed the remaining coordinates. The process is repeated for the second coordinate and so on, until the last coordinate is reached. A completely new vector \( x^{(n+1)} \) is obtained and compared with the previous vector \( x^{(1)} \). If the Euclidean distance between these two vectors is less than a pre-specified value \( \epsilon \), the algorithm stops. Otherwise, the hole process keeps running until the convergence is reached. Actually the algorithm is a classical derivative-free direct search method.

```
algorithm
read G(V, A, P), \( \Lambda \), \( \mu_i \), \( c_i^2 \), \( x^{(0)} \)
\( x^{(opt)} \leftarrow x^{(0)} \)
repeat
\( x^{(1)} \leftarrow x^{(opt)} \)
for \( i = 1 \) until \( n \)
\( \parallel \) unidirectional search \( \parallel \)
\( x^{(i+1)} \leftarrow \arg \min_{x \in \mathbb{R}^n} f(x^{(i)}) + \varepsilon \)
end for
if \( f(x^{(n+1)}) < f(x^{(1)}) \) then
\( x^{(opt)} \leftarrow x^{(n+1)} \)
end if
until \( ||x^{(opt)} - x^{(1)}|| < \epsilon \)
write \( x^{(opt)} \)
end algorithm
```

Figure 9: Algorithm for optimal buffer allocation.

6 Experimental Results

All algorithms were implemented in FORTRAN, taking advantage of the subroutines already developed for similar problems [17, 19] and are available upon request. The experiments were run for tandem, split, and merge queues, as presented in Fig. 10.

The arrival rates considered were \( \Lambda = \Theta_{\text{min}} = \{1.0, 2.0, 4.0\} \) users/s, the service rates, \( \mu_i = 10.0, \forall i \), resulting in the system utilization \( \rho = \{0.1, 0.2, 0.4\} \), combined with several values for the squared coefficient of variation, \( c_i^2 = \{0.5, 1.0, 2.0\} \), and number of nodes, \|V\| = \{3, 7, 15\}. The results are presented in Tab. 2.

From Tab. 2, it is seen that the pattern found in the small networks essentially becomes the pattern for the large networks. Also, the optimal allocation is clearly dependent on the \( c_i^2 \). The results make sense, are stable, and are symmetrical for the split and merge topologies. Notice that the buffer allocation is uniform across the series topology. This type of result is similar to the uniform buffer allocation results of de Kok [4] but the well-known bowl phenomenon [9] was not verified here. The bowl phenomenon is present in the optimal buffer allocations constrained to a maximal number of total buffer allocated, as, for instance, in the model presented in the article by Hillier and So [9].

In order to see how close to the optimal are the generated patterns, it is interesting to compare the results with those of simulation. Experiments with ARENA with 200,000 time units, 2,000 time units warm-up and 20 replications were found to yield fairly stable results and short 95% confidence intervals. For all the non-exponential service times, a 2-stage gamma distribution was used to capture the general service times with non-unit \( c_i^2 \). The results for some of the series queues are seen in Tab. 3.

The result for a three-node series topology was analyzed in some detail. Notice that the best solutions correspond closely to the lowest \( L(\alpha) \) (see Tab. 3, in boldface). The general conclusion is that optimization algorithm tends to allocate more space than necessary to ensure the desired performance. Also noticeable is that the CPU time for the simulations grows quickly as the number of node in the network increases indicating that the simulation may not be the most efficient tool for optimizing but it is certainly usefull for assessing the quality of solutions via other methods.

As a final note on the computational performance of the algorithm, it is important to know how it will behave with small changes in the input. The running times, in seconds, for the cases studied in Tab. 2 are presented in Figure 11, reporting boxplots in function of the topology and the number of nodes in the network.

The running times seem to be independent on the topology but they certainly will increase with the number of nodes. This increase, although not too drastic, is followed by an increase in the variability, which indicates that the running times may be less predictable for large networks.

7 Summary and Conclusions

One major difficulty in dealing with the buffer allocation problem (BAP), in general, and for \( M/G/1/K \) queues, in particular, is to find good approximate expressions for the performance measures of interest. The BAP is made much more difficult when queues are configured in networks, in which blocking after service frequency occurs and complicates the analysis. In this article, some of the most effective approximations for the blocking probability, a crucial performance measure for the BAP treated here, were extensively compared. The approximation by Smith seemed to be the most accurate for the cases tested and was used for solving the BAP.

The algorithm proposed is based on a Lagrangean relaxation, a technique that has been proved efficient in solving optimization problems with complicate con-
Figure 10: Topologies tested.

The Lagrangean relaxation enables one to avoid hard optimization formulations by relaxing complicating constraints and including them into the objective function as a penalty. Important properties of the relaxed problem were derived, which made possible the development of a search algorithm, considerably simpler than the algorithm previously published for the same problem by Smith and Cruz [19]. In comparison with the exact simulation results, the algorithm seemed to produce very fast and accurate solutions and can be
Table 2: Buffer allocation results.

<table>
<thead>
<tr>
<th>series topology</th>
<th>3</th>
<th>7</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>$c^*_2$</td>
<td>$K$</td>
<td>$K$</td>
</tr>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>3333333333333333</td>
<td>(3333333333333333)</td>
</tr>
<tr>
<td>1.0</td>
<td>(3333333333333333)</td>
<td>(3333333333333333)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(4444444444444444)</td>
<td>(4444444444444444)</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>5555555555555555</td>
<td>(5555555555555555)</td>
</tr>
<tr>
<td>1.0</td>
<td>(5555555555555555)</td>
<td>(5555555555555555)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(6666666666666666)</td>
<td>(6666666666666666)</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.5</td>
<td>(7777777777777777)</td>
<td>(7777777777777777)</td>
</tr>
<tr>
<td>1.0</td>
<td>(8888888888888888)</td>
<td>(8888888888888888)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(101010101010101010)</td>
<td>(101010101010101010)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split topology</th>
<th>3</th>
<th>7</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>$c^*_2$</td>
<td>$K$</td>
<td>$K$</td>
</tr>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>33222222221111</td>
<td>(33222222221111)</td>
</tr>
<tr>
<td>1.0</td>
<td>(33222222221111)</td>
<td>(33222222221111)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(43222222221111)</td>
<td>(43222222221111)</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>54332222222222</td>
<td>(54332222222222)</td>
</tr>
<tr>
<td>1.0</td>
<td>(54332222222222)</td>
<td>(54332222222222)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(64332222222222)</td>
<td>(64332222222222)</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.5</td>
<td>(75443333333333)</td>
<td>(75443333333333)</td>
</tr>
<tr>
<td>1.0</td>
<td>(86443333333333)</td>
<td>(86443333333333)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(10655444333333)</td>
<td>(10655444333333)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>merge topology</th>
<th>3</th>
<th>7</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>$c^*_2$</td>
<td>$K$</td>
<td>$K$</td>
</tr>
<tr>
<td>0.1</td>
<td>0.5</td>
<td>22222222222222</td>
<td>(22222222222222)</td>
</tr>
<tr>
<td>1.0</td>
<td>(22222222222222)</td>
<td>(22222222222222)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(22222222222222)</td>
<td>(22222222222222)</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.5</td>
<td>22222222223345</td>
<td>(22222222223345)</td>
</tr>
<tr>
<td>1.0</td>
<td>(22222222223345)</td>
<td>(22222222223345)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(22222222223345)</td>
<td>(22222222223345)</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.5</td>
<td>33344475</td>
<td>(33344475)</td>
</tr>
<tr>
<td>1.0</td>
<td>(33344475)</td>
<td>(33344475)</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>(34455610)</td>
<td>(34455610)</td>
<td></td>
</tr>
</tbody>
</table>

(*) Earmarked experiments checked by simulation (see Table 3).

used in the design of production systems.

Topics for future research in the area include extensions to systems that have loops, such as systems with captive pallets and fixtures. Also of interest is the study of algorithms for multi-server general service time queueing networks.

Acknowledgements

The research of prof. Frederico Cruz has been partially funded by the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) of the Ministry for Science and Technology of Brazil, grants 201046/1994-6, 301809/1996-8, 307702/2004-9, 472066/2004-8, and 472877/2006-2, by the FAPEMIG (Fundação de Apoio à Pesquisa do Estado de Minas Gerais), grants CEX-289/98 and CEX-855/98, and PRPq-UFMG, grant 4081-UFGM/RTR/FUNDO/PRPq/99.

REFERENCES


Table 3: Simulation results for tandem queues.

| $\lambda$ | $c^2$ | $|V|$ | $x$ | $\Theta(x)$ | $\delta$ | $L(\alpha)$ | CPU* |
|----------|-------|------|-----|------------|--------|-------------|------|
| 1.0      | 0.5   | 3    | (2 2 2) | 0.9928    | 0.0011  | 13.19       | 1.68 |
|          |       |      | (2 2 3) | 0.9928    | 0.0011  | 14.18       | 1.67 |
|          |       |      | (2 3 3) | 0.9928    | 0.0011  | 15.17       | 1.68 |
|          |       |      | (3 3 3) | 0.9994    | 0.0012  | 9.59        | 1.47 |
|          |       |      | (3 3 4) | 0.9999    | 0.0009  | 10.07       | 1.70 |
|          |       |      | (3 4 4) | 1.0000    | 0.0009  | 11.00       | 1.70 |
|          |       |      | (4 4 4) | 1.0000    | 0.0013  | 12.00       | 1.70 |
| 2.0      | 1.0   | 7    | (3 3 3 3 3 3) | 1.9861   | 0.0014  | 34.90       | 7.27 |
|          |       |      | (4 4 4 4 4 4) | 1.9966   | 0.0010  | 31.40†      | 7.28 |
|          |       |      | (5 5 5 5 5 5) | 1.9994   | 0.0013  | 35.60†      | 7.28 |
|          |       |      | (6 6 6 6 6 6) | 1.9996   | 0.0016  | 42.40       | 7.30 |
|          |       |      | (7 7 7 7 7 7 7) | 2.0001  | 0.0021  | 48.90       | 7.65 |

*CPU time for simulation in Arena in minutes.
† Best solution via simulation.
‡ Best solution via optimization algorithm.

Figure 11: Running times in function of the topology and number of nodes.


Buffer allocation


