On the Performance Estimation and Resource Optimisation in Process Petri Nets

Ricardo J. Rodríguez*, Jorge Júlvez, José Merseguer
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, María de Luna 1, 50018 Zaragoza, Spain
{rjrodriguez, julvez, jmerse}@unizar.es

***Abstract*—Many artificial systems can be modelled as discrete dynamic systems in which resources are shared among different tasks. The performance of such systems, which is usually a system requirement, heavily relies on the number and distribution of such resources. The goal of this paper is twofold: first, to design a technique to estimate the steady-state performance of a given system with shared resources; second, to propose a heuristic strategy to distribute shared resources so that the system performance is enhanced as much as possible. The systems under consideration are assumed to be large ones, such as Service Oriented Architectures (SOA) systems, and modelled by a particular class of Petri nets called process Petri net. In order to avoid the state explosion problem inherent to discrete models, the proposed techniques make intensive use of linear programming problems.**

***Index Terms*—Performance evaluation, Petri nets, software performance, Discrete Event Systems**

I. Introduction

NOWADAYS, the majority of systems in several domains (such as manufacturing, logistics or web services) are complex systems using shared resources. Usually, the number of resources is the key for the system to obtain a good throughput (defined as jobs completed per unit of time) for a large number of users/clients. However, the number of resources (for example, the number of servers) cannot be always incremented in the desired way: in the real world, each project of a new system manages a budget, and this budget limits the number of resources that can be acquired.

Many of these artificial systems can be naturally modelled as Discrete Event Systems (DES). Unfortunately, these systems are usually large what makes the exact computation of their performance a highly complex computational task. The main reason for this complexity is the well-known state explosion problem. As a result, a task that requires an exhaustive state space exploration becomes unachievable in reasonable time for large systems.

The framework of this paper is the one of DES dealing with the resource allocation problem, also called Resource Allocation Systems (RAS) [1], modelled with Petri nets; more precisely, we will focus on process Petri nets [2]. A large number of works in the literature deal with RAS from a qualitative point of view (computing deadlock avoidance [3]–[7] or siphons structures [8], [9]), whilst our vision here is different: we focus on the quantitative point of view. In particular, the goals of the paper are: 1) to efficiently estimate the throughput of a system and 2) to find a near-optimal distribution of resources for the so called process Petri nets. To the best of our knowledge, this resource optimisation issue has not been studied in the research community for process Petri nets.

To fulfil these goals, in this paper we propose, in first place, an iterative strategy to compute upper throughput bounds closer to the real throughput[1] than the ones that can be achieved in previous works [10], [11], and in second place, a heuristic iterative strategy to gauge in the best possible way the number of resources needed so that the overall system throughput is maximised. Both strategies use linear programming techniques for which polynomial complexity algorithms exist, so they offer a good trade-off between accuracy and computational complexity.

Let us summarise how the strategies presented here work. The strategy for getting sharper (i.e., closer to the real throughput) upper throughput bounds is based on the computation of *bottlenecks*. It calculates in a first step the slowest part of the system, that is, the initial bottleneck of the system. After that, in each iteration the most likely part of the system to be constraining the current bottleneck is calculated, and the union of both parts is considered to calculate the new upper throughput bound. The heuristic strategy for resource optimisation tries to calculate the number of resources the current bottleneck needs, so that when this number of resources is added, it is no longer the bottleneck.

Both strategies can be applied to any real-life application whose Petri net model matches with the net class considered in this paper, i.e., process Petri net. This kind of real-life applications can be found in manufacturing, logistics or dissimilar systems such as web services. In general, such applications represent real-life problems where resources are shared.

Running example. Let us consider a simple supermarket, where customers arrive and look for the products they want to buy. After spending some time in the supermarket, the customer wants to pay for the products, and a supermarket cashier attends him/her. The customer may choose to pay either in cash with a certain probability $p \in [0 \dots 1]$, or by credit card (with a probability $1 - p$), so the cashier will need a Point of Sales (PoS) terminal to complete the payment.

*This work was partially supported by Spanish project DPI2010-20413.

[1]The notion of real throughput refers to the throughput of the system modelled, which can be calculated by exact analysis or simulation.

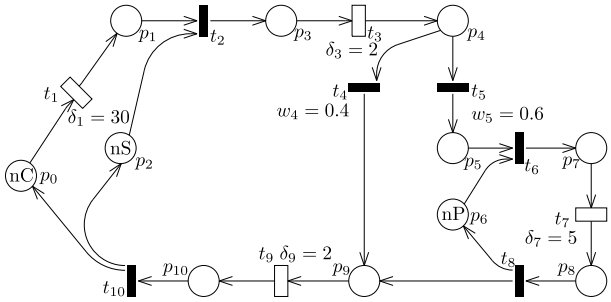


Figure 1. Example of a supermarket system.

Figure 1 depicts a Petri net (PN) modelling the supermarket system. The PN represents the number nC of clients (initial marking of place p_0) and the number nS of cashiers (initial marking of place p_2) who attend the customers and the PoS terminals, represented by the initial marking nP of place p_6. Immediate transitions are represented by a black box, while exponential transitions are depicted by a white box. The think time of customers is represented by transition t_1, which follows an exponential distribution with mean $\delta_1 = 30$ minutes, while transition t_3 represents the time for attending customers, which follows an exponential distribution with mean $\delta_3 = 2$ minutes. The choice of the mode of payment is represented by place p_4. A payment in cash occurs with a probability $w_4 = 0.4$, while credit card payment happens with a probability $w_5 = 0.6$. The use of the PoS terminals (represented by transition t_7) takes, in terms of time, about 5 minutes, i.e., $\delta_7 = 5$. Finally, the cashier spends, on average, $\delta_7 = 2$ minutes on finishing the customer request, which is represented by transition t_9.

With the above PN configuration, it is interesting to know, for example, where the bottleneck of the system is, that is, what the slowest part of the system is: is it the cashier's work? is it the use of the PoS terminal? Another question of interest is whether the system's resources are enough to attend an expected number of customers. Supposing there exists a budget to spend in the supermarket, and knowing the cost of hiring new cashiers and buying new PoS terminals, where and in which ratio should the money be spent? These are the kind of questions we are dealing with in this paper.

Suppose an initial marking of $nC = 5$ expected customers, $nS = 2$ cashiers and $nP = 2$ PoS terminals. With this initial configuration, no matter how many new cashiers were hired or how many new PoS terminals were bought, because the resources are not constraining the system: there are enough resources to attend those customers with such a think time ($\delta_1 = 30$ minutes). Nevertheless, if the number of expected customers is set to $nC = 100$ and the same think time is considered, the bottleneck of the system is in the number of cashiers. This indicates that new hirings should be done if it is desired to attend customers with such a think time.

The balance of the paper is as follows. Section II discusses the related work. In Section III some basic concepts are introduced, such as the kind of PNs we are dealing with. Then, in Section IV a new iterative algorithm for performance estimation is presented, while a new resource optimisation technique is explained in Section V. Section VI introduces a case study to prove our methods and the experiments carried out, with its conclusions. Finally, Section VII summarises the main contributions of this paper.

II. Related Work

Performance estimation using PNs is a topic which has been broadly studied. Some works are concerned to the exact computation of analytical measures of the performance [12], while others overcome the state explosion problem providing performance bounds [10], [11], [13]–[15] . The use of performance bounds, on which our approach is based, avoids the necessity of calculating the whole state space. The advantage of using performance bound computation is the reduced computing time, but its drawback is the difficulty to assess how accurate the computed bound is with respect to the real system performance.

One of the first works on performance bounds computation is [13], where strongly connected Marked Graphs (MGs) with deterministic timing are considered, and the reachability of the computation bound is proved. Some other works that compute performance bounds use linear programming techniques [10], [11], in the same way that our approach. These bounds are frequently calculated by using the first order moment (i.e., the mean) of the distributions associated to the firing delay. Such bounds were improved in [14] for the particular case of MGs by using regrowing techniques (that is, by adding more components to the initial bottleneck of the net). In [15], the second order moment is used to obtain a sharper (i.e., more accurate) performance bound.

Other works provide bounds for queueing systems instead of PN models like our approach does, e.g., [16]–[18]. Haddad et al. give in [16] space complexity upper and lower bounds for Stochastic Petri nets with product-form solution. In [17], Casale et al. propose performance upper and lower bounds for closed queueing networks with general independent and non-renewal services. They use linear programming techniques on the queue activity probabilities. Osogami and Raymond provide in [18] upper and lower bounds on the tail distribution of the transient waiting time for a general independent services queue. They use the two first moments of the service time and interarrival time, and solve it through semidefinite programming (SDP), a convex optimisation technique used for optimisation of complex systems. On the contrary, our approach uses first order moment and linear programming techniques.

Resource optimisation and its usage have been already studied for workflow Petri nets (WF-nets) [19] or some variants [20]–[22]. The underlying PN model of WF-nets are free choice nets (FCNs). However, the kind of systems we are considering cannot be modelled through FCNs: in the systems we consider, it may exist conflicts in the resources acquirement synchronisation, which is not allowed in FCNs. Li et al. propose in [19] an approach to estimate the resource availability by using Continuous Time Markov Chains (CTMCs) and compute the turnaround time (i.e., the shortest response time) by performing reduction operations on the

original WF-net. This performance analysis has an exponential complexity in the worst case, whilst our approach has a polynomial complexity due to the use of linear programming (LP) techniques. Resource usage could be computed in our approach by calculating the average marking of resource places in the PN system. Wang and Zeng provide in [20] a method for computing the best implementation case for a workflow represented by a PN model, based on the reachability graph. Such a method, however, can suffer scalability problems if the workflow size is large. Van Hee et al. give in [21] an algorithm to compute optimal resource allocation in stochastic WF-nets. Such an algorithm suffers from scalability problems because its complexity depends on the number of resources. On the contrary, our approach only depends on the net structure, no matters the number of resources in the system. Therefore, for large systems with great number of resources our approach is more tractable than the one in [21]. Chen et al. propose in [22] a new PN model, called Resource Assignment Petri Net (RAPN), to define how resources are shared and assigned among different and concurrent project activities. The computation of the execution project time considers deterministic timing and, unlike our approach, such a new PN model is not able to model activities acquiring and releasing resources in an intermittent way.

Another important issue related to resource sharing is deadlock prevention. The common use of system resources in concurrent systems may lead to deadlock problems, i.e., a process waits for the evolution of other process/es, while the latter is/are also waiting for the former to evolve. In order to deal with such problems, there exist deadlock prevention or avoidance policies which may be applied for assuring the liveness property and therefore to avoid deadlocks [3]–[7], [23], [24]. As this issue has been broadly studied in the literature (a recently published review can be found in [7]), and is not the main focus of this paper, we assume that all the PNs considered here are live.

With respect the aforementioned works, the contributions of this paper are the following. In first place, we provide a method to compute upper throughput bounds in a more accurate way than the upper bounds that can be achieved with the aforesaid works. In second place, we provide a heuristic iterative strategy to distribute, for a given budget, the number of resources in the best possible way so that the overall system throughput is maximised.

III. Preliminary Concepts and Definitions

Some basic concepts are introduced in this section regarding to the special class of Petri nets we are considering, and its main characteristics. Firstly, we define Petri nets in the untimed framework and the process Petri net formalism. Lastly, timed Petri net systems (visit ratios, average marking and steady-state throughput) are defined. In the following, the reader is assumed to be familiar with Petri nets (see [25] for a gentle introduction).

A. Untimed Petri nets

Definition 1: A Petri net [25] is a 4–tuple $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$, where:

- P and T are disjoint non-empty sets of *places* and *transitions* ($|P| = n$, $|T| = m$) and
- $\mathbf{Pre}$ ($\mathbf{Post}$) are the pre–(post–)incidence non-negative integer matrices of size $|P| \times |T|$.

The *pre-* and *post-set* of a node $v \in P \cup T$ are respectively defined as ${}^\bullet v = \{u \in P \cup T | (u,v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v,u) \in F\}$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. A Petri net is said to be *self-loop free* if $\forall p \in P, t \in T \; t \in {}^\bullet p$ implies $t \notin p^\bullet$. *Ordinary* nets are Petri nets whose arcs have weight 1. The *incidence matrix* of a Petri net is defined as $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

A vector $\mathbf{m} \in \mathbb{Z}_{\geq 0}^{|P|}$ which assigns a non-negative integer to each place is called *marking vector* or *marking*.

Definition 2: A *Petri net system*, or *marked Petri net* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$, is a Petri net $\mathcal{N}$ with an *initial marking* $\mathbf{m_0}$.

The set of markings *reachable* from $\mathbf{m_0}$ in $\mathcal{N}$ is denoted as $RS(\mathcal{N}, \mathbf{m_0})$ and is called the *reachability set*.

A place $p \in P$ is $k - bounded$ if, and only if, $\forall \mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0}), \mathbf{m}(p) \leq k$. A net system $\mathcal{S}$ is *k-bounded* if, and only if, each place is k-bounded. A net system is *bounded* if, and only if, there exists some k for which it is k-bounded. A net $\mathcal{N}$ is *structurally bounded* if, and only if, it is bounded no matter which $\mathbf{m_0}$ is the initial marking.

A transition $t \in T$ is *enabled* at marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}(\cdot, t)$, where $\mathbf{Pre}(\cdot, t)$ is the column of $\mathbf{Pre}$ corresponding to transition t. A transition t enabled at $\mathbf{m}$ can *fire* yielding a new marking $\mathbf{m}' = \mathbf{m} + \mathbf{C}(\cdot, t)$ (*reached* marking). This is denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m}'$. A sequence of transitions $\sigma = \{t_i\}_{i=1}^n$ is a *firing sequence* in $\mathcal{S}$ if there exists a sequence of markings such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_n} \mathbf{m}_n$. In this case, marking $\mathbf{m}_n$ is said to be *reachable* from $\mathbf{m_0}$ by firing σ, and this is denoted by $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}_n$. The *firing count vector* $\boldsymbol{\sigma} \in \mathbb{Z}_{\geq 0}^{|T|}$ of the firable sequence σ is a vector such that $\boldsymbol{\sigma}(t)$ represents the number of occurrences of $t \in T$ in σ. If $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, then we can write in vector form $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, which is referred to as the *linear* (or *fundamental*) *state equation* of the net.

Two transitions t, t' are said to be in *structural conflict* if they share, at least, one input place, i.e., ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Two transitions t, t' are said to be in *effective conflict for a marking* $\mathbf{m}$ if they are in structural conflict and they are both enabled at $\mathbf{m}$. Two transitions t, t' are in *equal conflict* if $\mathbf{Pre}(\cdot, t) = \mathbf{Pre}(\cdot, t') \neq \mathbf{0}$, where $\mathbf{0}$ is a vector with all entries equal to zero.

A transition t is *live* if it can be fired from every reachable marking. A transition t is *dead* for a reachable marking $\mathbf{m}$ if and only if $\forall \mathbf{m}' \in RS(\mathcal{N}, \mathbf{m}), \neg(\mathbf{m} \xrightarrow{t} \mathbf{m}')$. A marked Petri net $\mathcal{S}$ is *live* when every transition is live.

A *p-semiflow* is a non-negative integer vector $\mathbf{y} \geq \mathbf{0}$ such that it is a left anuller of the net's incidence matrix, $\mathbf{y} \cdot \mathbf{C} = 0$ (in the sequel, we omit the transpose symbol in the matrices and vectors for clarity). A p-semiflow implies a token conservation law independent from any firing of transitions. A *t-semiflow* is a non-negative integer vector $\mathbf{x} \geq \mathbf{0}$ such that is a right anuller of the net's incidence matrix, $\mathbf{C} \cdot \mathbf{x} = 0$. A p- (or t-)semiflow $\mathbf{v}$ is *minimal* when its support, $\|\mathbf{v}\| = \{i | \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other p- (or t-)semiflow, and the greatest common divisor of its elements is one. For

example, the PN depicted in Figure 1 has three minimal p-semiflows: $\|\mathbf{y}_1\| = \{p_0, p_1, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$, $\|\mathbf{y}_2\| = \{p_2, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$ and $\|\mathbf{y}_3\| = \{p_6, p_7, p_8\}$. A Petri net is said to be *conservative* (*consistent*) if there exists a p-semiflow (t-semiflow) which contains all places (transitions) in its support.

A Petri net is said to be *strongly connected* if there is a directed path joining any pair of nodes of the graph. A *state machine* is a particular type of ordinary Petri net where each transition has exactly one input arc and exactly one output arc, that is, $|t^\bullet| = |^\bullet t| = 1, \forall t \in T$. In this work, we focus on process Petri nets, which are defined as follows [2]:

Definition 3: A process Petri net is a strongly connected self–loop free Petri net $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ where:

1) $P = P_0 \cup P_S \cup P_R$ is a partition such that $P_0 = \{p_0\}$ is the *process-idle place*, $P_S \neq \emptyset, P_S \cap P_0 = \emptyset, P_S \cap P_R = \emptyset$, P_S is the set of *process-activity places* and $P_R = \{r_1, \ldots, r_n\},\ n > 0, P_R \cap P_0 = \emptyset$ is the set of *resources places*;
2) The subnet $\mathcal{N}' = \langle P \setminus P_R, T, \mathbf{Pre}, \mathbf{Post}\rangle$ is a strongly connected state machine, such that every cycle contains p_0.
3) For each $r \in P_R$, there exist a unique minimal p-semiflow associated to r, $\mathbf{y}_r \in \mathbb{N}^{|P|}$, fulfilling: $\|\mathbf{y}_r\| \cap P_R = \{r\}, \|\mathbf{y}_r\| \cap P_S \neq \emptyset, \|\mathbf{y}_r\| \cap P_0 = \emptyset$ and $\mathbf{y}_r(r) = 1$. This establishes how each resource is reused, that is, they cannot be created nor destroyed.
4) $P_S = \bigcup_{r \in P_R} (\|\mathbf{y}_r\| \setminus \{r\})$.

Definition 3 implies that process Petri nets are conservative and consistent.

Let $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ be a process Petri net. A vector $\mathbf{m_0} \in \mathbb{Z}_{\geq 0}^{|P|}$ is called *acceptable initial marking* [2] of $\mathcal{N}$ iff: 1) $\mathbf{m_0}(p) \geq 1,\ p \in P_0$; 2) $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$; and 3) $\mathbf{m_0}(r) \geq \mathbf{y}_r(r),\ \forall r \in P_R$, where $\mathbf{m_0}(r)$ is the *capacity* of the resource r and $\mathbf{y}_r$ is the unique minimal p-semiflow associated to r.

Definition 4: A *process Petri net system*, or *marked process Petri net* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0}\rangle$, is a process Petri net $\mathcal{N}$ with an *acceptable initial marking* $\mathbf{m_0}$.

In this work, we assume that the first acquired resource in process Petri nets under study is a resource that represents the maximum capacity of the process, being its capacity always greater than the number of instances in the process-idle place. Therefore, such a resource place becomes implicit and we do not consider it for the analysis.

B. Timed Petri nets

Definition 5: A Timed Process Petri net (TPPN) system is a tuple $\langle \mathcal{S}, \mathbf{s}, \mathbf{r}\rangle$, where $\mathcal{S}$ is a process Petri net system, $\mathbf{s} \in \mathbb{R}_{\geq 0}^{|T|}$ is the vector of average service times of transitions, and $\mathbf{r} \in \mathbb{N}_{>0}^{|T|}$ is the vector of rates associated to transitions.

If $\mathbf{s}(t) > 0$, then transition t is a timed transition. Otherwise, i.e., $\mathbf{s}(t) = 0$, transition t is an immediate one. It will be assumed that all transitions in conflict are immediate. An immediate transition t in conflict will fire with probability $\dfrac{\mathbf{r}(t)}{\sum_{t' \in A} \mathbf{r}(t')}$, where A is the set of enabled immediate transitions in conflict. The firing of immediate transitions consumes no time. When a timed transition becomes enabled, it fires following an exponential distribution with mean $\mathbf{s}(t)$. There exist different semantics for the firing of transitions, being *infinite* and *finite* server semantics the most frequently used. In this work, we will assume that the timed transitions work under infinite server semantics.

The average marking vector, $\overline{\mathbf{m}}$, in an ergodic Petri net system is defined as [26]:

$$\overline{\mathbf{m}}(p) \underset{AS}{=} \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau \mathbf{m}(p)_u du \tag{1}$$

where $\mathbf{m}(p)_u$ is the marking of place p at time u and the notation $\underset{AS}{=}$ means *equal almost surely*.

Similarly, the steady-state throughput, χ, in an ergodic Petri net is defined as [26]:

$$\chi(t) \underset{AS}{=} \lim_{\tau \to \infty} \frac{\boldsymbol{\sigma}(t)_\tau}{\tau} \tag{2}$$

where $\boldsymbol{\sigma}(t)_\tau$ is the firing count of transition t at time τ.

By definition, all the places of a TPPN are covered by p-semiflows, and therefore it is structurally bounded. In this work, we will assume that the TPPN under study is a live and structurally bounded net with Freely Related T-semiflows (i.e., a FRT-net) [27]. The range of nets fulfilling these conditions are relatively broad. Examples of TPPNs that are FRT-nets are: TPPNs in which $\mathcal{N}'$ is choice-free; TPPNs in which $\mathcal{N}'$ satisfies that every conflict is an equal conflict. It is known that the continuous time Markov chain associated to these nets is ergodic [27], what implies the existence of the above limits.

The vector of visit ratios expresses the relative throughput of transitions in the steady state. The visit ratio $\mathbf{v}(t)$ of each transition $t \in T$ normalised for transition t_i , $\mathbf{v}^{t_i}(t)$, is expressed as follows:

$$\mathbf{v}^{t_i}(t) = \frac{\chi(t)}{\chi(t_i)} = \boldsymbol{\Gamma}(t_i) \cdot \chi(t),\ \forall t \in T \tag{3}$$

where $\boldsymbol{\Gamma}(t_i) = \dfrac{1}{\chi(t_i)}$ represents the *average inter-firing time* of transition t_i.

In FRT-nets, the vector of visit ratios $\mathbf{v}$ exclusively depends on the structure of the net and on the routing rates [27]. Thus, the vector of visit ratios $\mathbf{v}$ normalised for transition t_i, $\mathbf{v}^{t_i}$, can be calculated by solving the following linear system of equations [27]:

$$\begin{aligned} \begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v}^{t_i} &= 0 \\ \mathbf{v}^{t_i}(t_i) &= 1 \end{aligned} \tag{4}$$

where $\mathbf{R}$ is a matrix containing the rates $\mathbf{r}(t)$ associated to transitions in equal conflict.

IV. Performance Estimation

In this section we present a new iterative algorithm to compute upper throughput bounds of a timed process Petri net system. Such an algorithm is based on the computation of p-semiflows. Each p-semiflow has associated a subnet composed of the places in the support of the p-semiflow. Given that such a subnet satisfies a conservation law, and it synchronises

with other subnets in the overall system, its throughput, if the subnet is considered isolated, imposes an upper throughput bound for the overall system. The proposed iterative strategy considers initially the p-semiflow with lowest throughput, and its associated subnet is called initial bottleneck. Then, such a bottleneck is increased by adding the associated subnet to the subnet associated to the next most constraining p-semiflow.

A. Little's Law and Upper Bounds

The Little's formula [28] conditions, involves the average number of customers L in the system, the throughput, λ, and the average time spent by a customer within the system, W.

$$L = \lambda \cdot W \tag{5}$$

Let p be a place such that $|p^\bullet| = 1$, and $p^\bullet = \{t\}$, then the pair (p,t) can be seen as a simple queueing system to which, if the limits of *average marking* and *steady-state throughput* exist, the Little's formula can be directly applied [27]:

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{r}(p) \tag{6}$$

where $\mathbf{Pre}(p,t) \cdot \chi(t)$ is the output rate of tokens from place p, which in steady state is equal to the input rate, and $\mathbf{r}(p)$ is the average residence time at place p, i.e., the average time spent by a token in place p.

The average residence time, $\mathbf{r}(p)$, is the sum of the average waiting time due to a possible synchronisation and the average service time, $\mathbf{s}(t)$. Therefore, equation (6) becomes:

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{r}(p) \geq (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{s}(t) \tag{7}$$

where the service time $\mathbf{s}(t)$ is a lower bound for the average residence time $\mathbf{r}(p)$, i.e., $\mathbf{s}(t) \leq \mathbf{r}(p)$, since place p has only one output transition. Given that conflicting transitions are assumed to be immediate, equation (7) can also be applied to any pair (p,t) being $t \in p^\bullet$ and t a transition in conflict. Hence, the following system of inequalities can be derived [27] from (3) and (7):

$$\mathbf{\Gamma}(t_i) \cdot \overline{\mathbf{m}} \geq \mathbf{Pre} \cdot \mathbf{D^{t_i}} \tag{8}$$

where $\mathbf{\Gamma}(t_i)$ is the average interfiring time of transition t_i and $\mathbf{D^{t_i}}$ is the vector of *average service demands of transitions*, $\mathbf{D^{t_i}}(t) = \mathbf{s}(t) \cdot \mathbf{v}^{t_i}(t)$ (the vector of visit ratios $\mathbf{v}^{t_i}$ is normalised for transition t_i). In the sequel, we omit the superindex t_i in $\mathbf{D^{t_i}}$ for clarity.

After some manipulations of equation (8), a lower bound for the *average inter-firing time* of transition t_i, $\mathbf{\Gamma^{lb}}(t_i)$, can be computed by solving the following LP problem (LPP) [27]:

$$\begin{aligned} \mathbf{\Gamma}(t_i) \geq \mathbf{\Gamma^{lb}}(t_i) = \; & maximum\ \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ & subject\ to\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{9}$$

As a side product of the solution of (9), $\mathbf{y}$ represents the *slowest* p-semiflow of the system, thus LPP (9) can also be seen as a search for the most constraining p-semiflow. This p-semiflow will be the one with highest ratio $\frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$. Therefore, an upper bound $\Theta(t_i)$ for the steady-state throughput can be calculated as the inverse of the lower bound for the average inter-firing time $\mathbf{\Gamma^{lb}}(t_i)$, that is, $\Theta(t_i) = \frac{1}{\mathbf{\Gamma^{lb}}(t_i)}$.

B. Next Slowest P-semiflow

The LPP shown in (9) was the basis in [14] for developing an iterative algorithm to compute upper bounds in Stochastic Marked Graphs. Unfortunately, the proposed algorithm is not applicable to more general nets than Marked Graphs, hence we pursue for an alternative method.

The new algorithm will follow a similar strategy: firstly, the initial bottleneck is computed using (9). Then, in each iteration step the next slowest p-semiflow connected to the subnet associated to the current bottleneck is added to it.

Let us suppose the p-semiflow $\mathbf{y}^*$ represents the initial bottleneck, i.e., $\mathbf{y}^*$ is obtained from the solution of (9). The following constraint forces that some other p-semiflow $\mathbf{y}$, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, is connected to $\mathbf{y}^*$: $\sum_{p \in V} \mathbf{y}(p) > 0$, where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$ (that is, there exist places in the support of $\mathbf{y}$ which share output transitions with places in the support of $\mathbf{y}^*$). Hence, the p-semiflow $\mathbf{y}$ with highest ratio $\frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$ connected to $\mathbf{y}^*$ can be searched by solving the following LPP:

$$\begin{aligned} & maximum\ \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ & subject\ to\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y}(p) > 0,\ \forall p \in Q \\ & \sum_{p \in V} \mathbf{y}(p) > 0 \end{aligned} \tag{10}$$

where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$, and $Q = \{q \in P, q \in \|\mathbf{y}^*\|\}$.

As a result of LPP (10), we will obtain the p-semiflow $\mathbf{y}$, which will be a linear combination of $\mathbf{y}^*$ and the next most constraining p-semiflow.

The strict inequality in (10) could lead us to numerical problems since the lower the value of $\sum_{p \in V} \mathbf{y}(p)$, the higher the value of the optimisation function. The appendix discusses this issue and shows that the solution proposed in the following can be applied in practice. A way to solve this is by reformulating $\sum_{p \in V} \mathbf{y}(p) > 0$ into $\sum_{p \in V} \mathbf{y}(p) \geq h$, where h is strictly positive. The problem now is to set an appropriate value for h. A high value can make constraints $\mathbf{y} \cdot \mathbf{m_0} = 1$ and $\sum_{p \in V} \mathbf{y}(p) \geq h$ incompatible leading to an infeasible LPP. A valid value of h can be obtained by searching a real number that is lower than each component of a p-semiflow $\mathbf{y}$ that covers all places and satisfies $\mathbf{y} \cdot \mathbf{m_0} = 1$. Such a value can be obtained by means of the following LPP:

$$\begin{aligned} &maximum\ h \\ &subject\ to\ \mathbf{y}\cdot\mathbf{C}=\mathbf{0} \\ &\qquad\qquad \mathbf{y}\cdot\mathbf{m_0}=1 \\ &\qquad\qquad \mathbf{y}\geq h\cdot\mathbf{1} \\ &\qquad\qquad h>0 \end{aligned} \tag{11}$$

where $\mathbf{1}$ is a vector with all entries equal to one.

The obtained value h ensures the feasibility of the following LPP, which is just a reformulation of (10):

$$\begin{aligned} &maximum\ \mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D} \\ &subject\ to\ \mathbf{y}\cdot\mathbf{C}=\mathbf{0} \\ &\qquad\qquad \mathbf{y}\cdot\mathbf{m_0}=1 \\ &\qquad\qquad \mathbf{y}(p)\geq h,\ \forall p\in Q \\ &\qquad\qquad \sum_{p\in V}\mathbf{y}(p)\geq h \end{aligned} \tag{12}$$

where $V=\{v|v\in {}^\bullet(\|\mathbf{y}^*\|^\bullet)\setminus\|\mathbf{y}^*\|\}$, and $Q=\{q\in P, q\in\|\mathbf{y}^*\|\}$.

As it is said, the last constraint, $\sum_{p\in V}\mathbf{y}(p)\geq h$, imposes that the support of $\mathbf{y}$ corresponds to the p-semiflow connected to $\mathbf{y}^*$ with highest $\frac{\mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D}}{\mathbf{y}\cdot\mathbf{m_0}}$.

C. An Iterative Strategy to Compute Upper Throughput Bounds

This subsection presents an iterative strategy to obtain an improved upper throughput bound in TPPNs. The strategy calculates, in a first step, the initial throughput bound of the system with the LPP (9) and takes the subnet associated to $\mathbf{y}$ as the initial bottleneck. Then, in each iteration the subnet associated to the p-semiflow that is potentially more constraining than the others is added to the bottleneck, and after that, the throughput is calculated. Note that such an addition in each iteration is restricting the behaviour of the system, what implies a lower throughput. The iteration process stops when no significant improvement of the bound is achieved.

The algorithm in Figure 2 represents the strategy used to compute throughput upper bounds. The algorithm needs as input the TPPN system to be analysed, $\langle\mathcal{S},\mathbf{s},\mathbf{r}\rangle$, and a degree of precision ($\varepsilon>0$) to be achieved. As output, the upper throughput bound, Θ, and the places belonging to the bottleneck of the TPPN, $\mathbf{Q}$, are obtained. The degree of precision ε will be used for the stopping criterion of the iterative strategy.

In first place, the initial upper throughput bound is calculated by LPP (9) (step 1). Then, the value of h such that ensures feasibility of the LP is computed by using the LPP shown in (11). The iteration process (steps 4–9) is repeated until no significant improvement is achieved with respect to the last iteration or the last obtained bottleneck contains all places in its support. In the worst case, only one place will be added in each iteration. Therefore, the algorithm complexity is polynomial due to the LPP.

Input: $\langle\mathcal{S},\mathbf{s},\mathbf{r}\rangle$, ε
Output: Θ, $\mathbf{Q}$

1: $\{\Theta,\mathbf{y}\}=$ Upper throughput bound and components of the initial bottleneck of $\langle\mathcal{S},\mathbf{s},\mathbf{r}\rangle$ according to (9)
2: Calculate value h by solving LPP (11)
3: $\Theta'=0$; $\mathbf{Q}=\{p\in P, p\in\|\mathbf{y}\|\}$
4: **while** $\frac{\Theta-\Theta'}{\Theta}\geq\varepsilon$ **and** $\mathbf{Q}\neq P$ **do**
5: $\quad\mathbf{V}=\{v|v\in{}^\bullet(\mathbf{Q}^\bullet)\setminus\mathbf{Q}\}$
6:

$$\begin{aligned} &maximum\ \mathbf{y}'\cdot\mathbf{Pre}\cdot\mathbf{D} \\ &subject\ to\ \mathbf{y}'\cdot\mathbf{C}=\mathbf{0} \\ &\qquad\qquad \mathbf{y}'\cdot\mathbf{m_0}=1 \\ &\qquad\qquad \mathbf{y}'(p)\geq h,\ \forall p\in\mathbf{Q} \\ &\qquad\qquad \sum_{p\in V}\mathbf{y}'(p)\geq h \end{aligned}$$

7: $\quad\Theta'=\Theta$
8: $\quad\Theta=$ Throughput of the net composed by the p-semiflow $\mathbf{y}'$
9: $\quad\mathbf{Q}=\{p\in P, p\in\|\mathbf{y}'\|\}$
10: **end while**

Figure 2. The iterative strategy algorithm for computing upper throughput bounds.

In step 5, the places that share output transitions with some place contained in the support of $\mathbf{y}$ are calculated. Step 6 corresponds to the LPP (12). Finally, in step 8 the throughput of the subnet associated to the new bottleneck is considered as the new upper bound. The throughput is calculated by solving the Markov Chain [25] associated to the current bottleneck when it can be computed in practical time, or by simulation otherwise.

Example. Consider again the supermarket example shown in Figure 1. Let the initial marking be $nC=21$, $nS=4$ and $nP=2$. The vector of visit ratios $\mathbf{v}$ normalised for transition t_1, is $\mathbf{v}^{t_1}=\{1.0,1.0,1.0,0.4,0.6,0.6,0.6,0.6,1.0,1.0\}$. According to LPP (9) (step 1 of the algorithm in Figure 2) the critical bottleneck is composed by $\|\mathbf{y}\|=\{p_0,p_1,p_3,p_4,p_5,p_7,p_8,p_9,p_{10}\}$, that is, the p-semiflow which corresponds to the customers' life-cycle. Such a result indicates that the system has, in average, enough resources to attend the expected incoming customers. The upper throughput bound (normalised for transition t_1) of the critical bottleneck is $\Theta(t_1)=0.567521$ (result of LPP (9)) and the value which guarantees the feasibility of the problem is $h=0.037037$ (step 2). The places sharing output transitions with places in $\|\mathbf{y}\|$, i.e., connected to the critical bottleneck, are p_2 and p_6 (calculated in step 5), each one corresponds to the resources of the system, supermarket cashiers and PoS terminals, respectively. The result of LPP in step 6 allows to regrow the current bottleneck, imposing that $\mathbf{y}'(p_2)+\mathbf{y}'(p_6)\geq h$ (that is, one of them, at least, must be contained on the support of $\mathbf{y}'$), and gives the new bottleneck which is composed by $\|\mathbf{y}'\|=\{p_0,p_1,p_2,p_3,p_4,p_5,p_7,p_8,p_9,p_{10}\}$. The new throughput is $\Theta'(t_1)=0.514220$ (step 8), which represents

an improvement of 9.3919% with respect to the previous bottleneck. Note that the place added is the one representing the number of cashiers (i.e., p_2).

Let us assume that $\varepsilon = 0.001$. As the relative difference between Θ and Θ' is 0.093919 (as commented previously), the iteration process carries on. At this point, the only place that is not connected to the critical bottleneck is p_6, which corresponds to the number of PoS terminals. By solving the LPP in step 6 the new bottleneck is obtained, which has all places of the system in its support (i.e., $\|\mathbf{y}\| = P$), and the new throughput is $\Theta = 0.480642$. So, as the support of the new bottleneck contains all places of the net, the iteration process finishes. The new throughput Θ represents an improvement of 6.5299% with respect to the previous bottleneck, and a total improvement of 15.3085% with respect to the initial bottleneck.

The proposed iterative strategy is applied to a larger system in Section VI.

V. Resource Optimisation

In this section we propose a heuristic strategy to gauge the number of resources a system should allocate. Our approach for resource optimisation is similar to the Goldratt's principle [29]: once the system's bottleneck is identified, the associated resource is incremented.

A. Calculating the Next Constraining Resource

Let us recall the LPP (9) to calculate an upper throughput bound. The most constraining p-semiflow, $\mathbf{y}$, will have just one marked place in its support due to the net structure (as explained in Section III). Assume that the marked place corresponds to a resource place (not the process-idle place), then given that $\mathbf{y}$ constraints the throughput of the whole system, the addition of more instances to the resource place will result in an increase of the system throughput. At a certain moment, the resource becomes saturated and adding more instances does not improve the throughput. This occurs because the constraining p-semiflow has changed. Note that the upper throughput bound will linearly increment with the number of tokens of the resource place, i.e., as it is the only place in $\|\mathbf{y}\|$ having tokens and the equation $\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is linear.

Hence, the resource r_1 contained on the support of the most constraining p-semiflow $\mathbf{y}_{r_1}$, can be incremented until $\mathbf{y}_{r_1}$ is no longer the bottleneck p-semiflow. Let ${\mathbf{m_0}}^{\Delta}$ be the initial marking vector $\mathbf{m_0}$ with an increment α_1 of the resource r_1, i.e.,

$$ {\mathbf{m_0}}^{\Delta} = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \end{cases} \tag{13} $$

The p-semiflow $\mathbf{y}_{r_1}$ is not the only constraining p-semiflow if the following equation holds:

$$ \frac{\mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y}_{r_1} \cdot {\mathbf{m_0}}^{\Delta}} \leq \frac{\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y}_{r_2} \cdot {\mathbf{m_0}}^{\Delta}} \tag{14} $$

where $\mathbf{y}_{r_2} \neq \mathbf{y}_{r_1}$ is a p-semiflow. Note that the p-semiflow $\mathbf{y}_{r_2}$ will contain in its support the next most constraining resource r_2, and, by definition, $r_1 \neq r_2$.

The number α_1 of instances of the resource place r_1, contained in the most constraining p-semiflow $\mathbf{y}_{r_1}$, needed to be added to obtain the next constraining resource r_2, contained in the next most constraining p-semiflow $\mathbf{y}_{r_2}$, can be easily computed by solving the following LPP:

$$ \begin{array}{lrll} minimum & \alpha_1 & & \\ subject\ to & \mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} & = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D} & \\ & \mathbf{y}_{r_2} \cdot \mathbf{C} & = 0 & \\ & \mathbf{y}_{r_2}(r_1) & = 0 & \\ & \mathbf{y}_{r_2} \cdot {\mathbf{m_0}}^{\Delta} & = \mathbf{y}_{r_1} \cdot {\mathbf{m_0}}^{\Delta} & \\ & {\mathbf{m_0}}^{\Delta} & = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \end{cases} & \\ & \alpha_1, \mathbf{y}_{r_2} & \geq 0 & \end{array} \tag{15} $$

where $\mathbf{y}_{r_1}$ is the p-semiflow which contains r_1 in its support, $\mathbf{y}_{r_2}$ is the p-semiflow which contains r_2 in its support and ${\mathbf{m_0}}^{\Delta}$ represents the initial marking vector $\mathbf{m_0}$ with the increment α_1 in r_1.

Constraints $\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}$ and $\mathbf{y}_{r_2} \cdot {\mathbf{m_0}}^{\Delta} = \mathbf{y}_{r_1} \cdot {\mathbf{m_0}}^{\Delta}$ are both parts (dividend and divisor, respectively) of equation (14) equalled. Constraint $\mathbf{y}_{r_2} \cdot \mathbf{C} = 0$ ensures that $\mathbf{y}_{r_2}$ is a left anuller of the incidence matrix, hence a p-semiflow of the net. Finally, constraint $\mathbf{y}_{r_2}(r_1) = 0$ is added to avoid a product of two optimisation variables (the variable α_1 and the variable $\mathbf{y}_{r_2}(r_1)$ in equation $\mathbf{y}_{r_2} \cdot {\mathbf{m_0}}^{\Delta} = \mathbf{y}_{r_1} \cdot {\mathbf{m_0}}^{\Delta}$). Moreover, the variable $\alpha_1 \in \mathbb{R}_{\geq 0}$ therefore, the linearity of the optimisation problem is ensured.

Both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained when the LPP is solved. Note that the increment of a resource r_1 does not affect to the ratio $\frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$ of any other minimal p-semiflow $\mathbf{y}$ which contains in its support other resource (see definition of process Petri nets class in Section III). Notice that, as in the Section IV, an LPP is used to solve a problem that deals with integer values as the number of resources. This relaxation to the real domain remarkably decreases the complexity of the approach (the complexity of solving an LPP is polynomial), and has the cost of some lost of precision in the results. The LPP (15) can easily be extended to, once both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained, calculate the next constraining resource and the number of tokens, i.e., instances, to increment the marking of both places:

$$
\begin{array}{lll}
minimum & & \alpha_1 + \alpha_2 \\
subject\ to & \mathbf{y'} \cdot \mathbf{Pre} \cdot \mathbf{D} & = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D} \\
& \mathbf{y'} \cdot \mathbf{C} & = 0 \\
& \mathbf{y'}(r_1) = 0, & \mathbf{y'}(r_2) = 0 \\
& \mathbf{y'} \cdot \mathbf{m_0}^{\Delta} & = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta} \\
& \mathbf{y'} \cdot \mathbf{m_0}^{\Delta} & = \mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta} \\
& \mathbf{m_0}^{\Delta} & = \left\{ \begin{array}{ll} \mathbf{m_0}(p), & p \notin \{r_1, r_2\} \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \\ \mathbf{m_0}(p) + \alpha_2, & p = r_2 \end{array} \right. \\
& \alpha_1, \alpha_2, \mathbf{y'} & \geq 0
\end{array}
\tag{16}
$$

where $\mathbf{m_0}^{\Delta}$ represents the initial marking vector $\mathbf{m_0}$ with the increment α_1 of the place r_1 and the increment α_2 of the place r_2, and $\mathbf{y}_{r_1}$ ($\mathbf{y}_{r_2}$) is the p-semiflow which contains r_1 (r_2) in its support.

As in LPP (15), constraint $\mathbf{y'} \cdot \mathbf{C} = 0$ ensures that $\mathbf{y'}$ is a left anuller of the incidence matrix, and hence $\mathbf{y'}$ is a p-semiflow of the net. Besides, constraints $\mathbf{y'}(r_1) = 0$ and $\mathbf{y'}(r_2) = 0$ ensure linearity of the optimisation problem. Constraints $\mathbf{y'} \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta}$, $\mathbf{y'} \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta}$ are the key of this LPP because from those equations both values of α_1 and α_2 can be obtained.

Note that $\mathbf{y'} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is not a constraint in LPP (16). This is a consequence of the result of LPP (15): from the latter LPP where r_1 is calculated, it is imposed that $\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}$. The addition of this constraint is not adding new information to the LPP (16).

The LPP (16) can be generalised for more resources, as it is shown at step 5 of the algorithm in Figure 3.

B. An Iterative Strategy for Resource Optimisation

This subsection presents a heuristic iterative strategy that aims at maximising the throughput by increasing the number of resources appropriately. The main idea of the strategy is to estimate the *inflexion points* where the constraining p-semiflows change, and hence to estimate the increment of resources needed. More precisely, each unit of a resource has associated a cost and the strategy establishes how to spend a given budget such that the throughput is maximised. The strategy ends either when there is no budget to spend, when all resources have been dimensioned or when the last computed p-semiflow points out to increment the process-idle place.

The algorithm in Figure 3 shows the resource optimisation heuristic strategy. As input, the algorithm needs the TPPN system to be analysed, $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, the set of resources and the process-idle place of the system, R and p_0 (respectively), the assigned budget to be spent, $budget$, and the vector of cost c, which assigns a cost c_i to each of the resource r_i contained in R. The output is the number of items n_i to increment each resource r_i.

Firstly, an upper throughput bound $\mathbf{y}_1$ of $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$ is calculated according to LPP (9). After that, the iteration process (steps 3–10) is repeated until the last assignment of resources has consumed the available budget, all resources have been dimensioned (i.e., there is enough budget for adding as many instances of resources as needed) or the last computed place to be incremented its marking matches with the process-idle place.

Input: $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, R, p_0, $budget$, c
Output: n

1: Calculate initial bottleneck $\mathbf{y}_1$ by solving LPP (9)
2: $k = 0$; $cost = 0$; $n^{'} = \mathbf{0}$
3: **while** $cost < budget$ **and** $k \neq |R|$ **and** $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ **do**
4: $k = k + 1$; $cost' = cost$; $n = n^{'}$; $\mathbf{A} = \{p | p \in P, p \in \|\mathbf{y}_j \cap R\|\}$, $\forall j \in \{1 \dots k\}$
5:

$$
\begin{array}{ll}
minimum & \sum_{j=1}^{k} \alpha_j \\
subject\ to & \mathbf{y_{k+1}} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y_1} \cdot \mathbf{Pre} \cdot \mathbf{D} \\
& \mathbf{y_{k+1}} \cdot \mathbf{C} = \mathbf{0} \\
& \mathbf{y_{k+1}} \cdot \mathbf{m_0^{\Delta}} = \mathbf{y_j} \cdot \mathbf{m_0^{\Delta}},\ \forall j \in \{1 \dots k\} \\
& \mathbf{m_0}^{\Delta} = \left\{ \begin{array}{ll} \mathbf{m_0}(p) + \alpha_j, & p \in \mathbf{A} \\ \mathbf{m_0}(p), & otherwise \end{array} \right. \\
& \mathbf{y_{k+1}}(p) = 0,\ p \in \mathbf{A} \\
& \mathbf{y_{k+1}},\ \alpha_j \geq 0,\ \forall j \in \{1 \dots k\}
\end{array}
$$

6: $cost = 0$; $n^{'} = \mathbf{0}$
7: **for each** α_j, $\forall j \in \{1 \dots k\}$ **do**
8: $r_j = \|\mathbf{y}_j\| \cap R$; $n_j^{'} = \lceil \alpha_j \rceil$
9: $cost = cost + \lceil \alpha_j \rceil \cdot c_i$
10: **end for each**
11: **end while**
12: **if** $k \leq |R|$ **and** $cost \leq budget$ **then**
13: $n = n^{'}$
14: **end if**
15: **if** $k < |R|$ **and** $cost \leq budget$ **and** $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ **then**
16: $assignRestOfBudget(budget - cost, \langle \mathcal{S}, \mathbf{s} \rangle, R, c, n)$
17: **end if**

Figure 3. The resource optimisation heuristic strategy algorithm.

Step 5 calculates, in each iteration, the number of items of a resource which need to be incremented to obtain the next restrictive resource. Remark that the LPP in step 5 is a generalisation of the LPP (15). After that, the cost of incrementing such a number of instances of the resources is computed. Note that the ceiling integer of the value α_j is taken as result. This is motivated by two reasons: firstly, we are assuming that the number of instances of the resources must be a natural number; and secondly, if the resource is not saturated it will still be the restrictive resource.

Finally, step 12 checks whether all resources have been assigned and the cost of new resources does not exceeds the given budget. If it is so, the last assignment is taken as the valid one. Step 15 checks whether there exists some resource that has not been assigned, the last resources assignment does not overwhelm the given budget and the last computed p-semiflow does not contain the process-idle place. When these

conditions are fulfilled, it means that the remaining budget may be spent incrementing the system throughput. A procedure is invoked ($assignRestOfBudget$, step 16) for spending the rest of the assigned budget to increment the resources as much as possible. Note that the assignment of the remaining budget is an NP-problem, similar to the Bounded Knapsack Problem (BKP) [30]. Several heuristics can be used, as for example, a "round-trip" algorithm which tries to increment all the resources per round, or at least until the last resource which can be incremented.

Let us illustrate the use of this strategy through the supermarket example, depicted in Figure 1. Suppose an initial marking of $nC = 30$, $nS = 2$ and $nP = 2$, an initial budget $\$30,000$ dollars. Besides, a new hiring of a supermarket cashiers costs $\$5,000$ dollars while a new PoS terminal has a price of $\$700$ dollars. The initial bottleneck is $\|\mathbf{y_1}\| \cap R = \{p_{nS}\}$, that is, the subnet associated to the customers' cashiers. Therefore, this result is giving us the following information: to attend 30 customers whose think time follows an exponential distribution of mean 30 minutes, more supermarket cashiers are needed to attend them. The LPP at step 5 gives, in the first iteration, the increment of new cashiers needed, $\alpha_1 = 2.666$, and the new constraining p-semiflow, which corresponds to the use of the PoS terminals. So, at least three new cashiers ($\lceil \alpha_1 \rceil$) are needed to attend the customers.

As the cost of hiring new cashiers is $\$5,000$ dollars and the initial budget is $\$30,000$ dollars, the new hirings can be done and there is still money which remains to be spent, so a new iteration can take place. The LPP at step 5 gives, in the second iteration, the values of $\alpha_1 = 3.6752$ and $\alpha_2 = 0.4322$. Hence, to attend the customers, four new hirings and one more PoS terminals are needed. As the cost of these increments are, in total, $\$20,700$ dollars, the increment of resources can be carried out. Now, the unassigned budget is $\$9,300$ and we can follow incrementing both resources in parallel. Indeed, the relation between both resources is known thanks to the equalities of ratios.

In this case, even though some budget remains to be spent, the new constraining p-semiflow contains the process-idle place, that is, the place representing customers. Thus, the resources of the system (cashiers and PoS terminals) have been optimally calculated to attend 30 customers whose think time follows an exponential distribution of mean 30 minutes. In this way, the algorithm has calculated that to attend such customers, at least four more supermarket cashiers and one PoS terminals are needed.

Note that it may happen that the LPP at step 5 returns the p-semiflow containing the process-idle place in the first iteration. This fact would indicate that the system has enough resources to attend such a number of customers with such a think time. Therefore, the strategy is also able to compute when a system with an initial configuration is able to support the estimated workload, or otherwise, to compute the number of instances of resources needed to be able to support it.

VI. Case Study: a Secure Database System

In this section we introduce a case study to test our approach. We consider the design of a Secure Database System (SDBS) deployed as a Web Service which stores sensible information. Besides, there exist users which are eventually accessing to this information. A real application of this kind of system is, for instance, a web server keeping customer's data of an insurance company or a bank web server keeping customer's balance accounts.

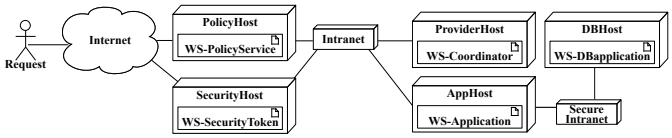


Figure 4. SDBS Deployment.

A. Description

Figure 4 shows the actual deployment of the SDBS, which includes the hardware resources (depicted as cubes) and their network links (arrows between cubes or proper cubes in the case of intranets). Software modules (depicted as squares) are deployed into hardware resources. The architecture of the system is as follows: there exist a policy host, a security host, a provider host, an application host and a database host. Moreover, the latter is isolated and reachable only through a secure intranet connected to the application host. Note that each of these hosts deploys a concrete service or software module.

Following Figure 5, the SDBS works as follows: a user interacts with an application outside the system (`WS-Requester`), which collects its personal data and the type of operation required (let us assume it will be an update of personal user data) by the user. This information is summarised on a request. Each request incoming into the system needs a security token to be identified before accessing the system, which is provided by the security host through `WS-SecurityToken` service. Once the security token is retrieved, the request is accordingly signed and then the policy host is requested for accessing, which checks the request (`WS-PolicyService`). When the permission is granted, it invokes the `WS-Coordinator` service, which communicates with the `WS-Application` service (located in the application host). The latter host has access to the database application (`WS-DBapplication`) through a secure intranet. Finally, the DB application definitively updates the user request into the DB and an acknowledge report is returned back through the system.

The Petri net (PN) representing the behaviour of the SDBS system is depicted in Figure 6. Each resource is represented by a dark grey place in the PN: p_7 (policy host), p_{18} (security host), p_{26} (provider host), p_{28} (application host) and p_{31} (database host); while user's requests are represented by the process-idle place p_0 (depicted in light grey). The number of instances of each resource is summarised in Table I(b), and they will be represented by tokens in the respective place. Note that different number of tokens in p_0 will be used for sensitive analysis in experiments.

The acquire (release) of a resource is represented by an immediate transition with an input (output) arc. For instance,

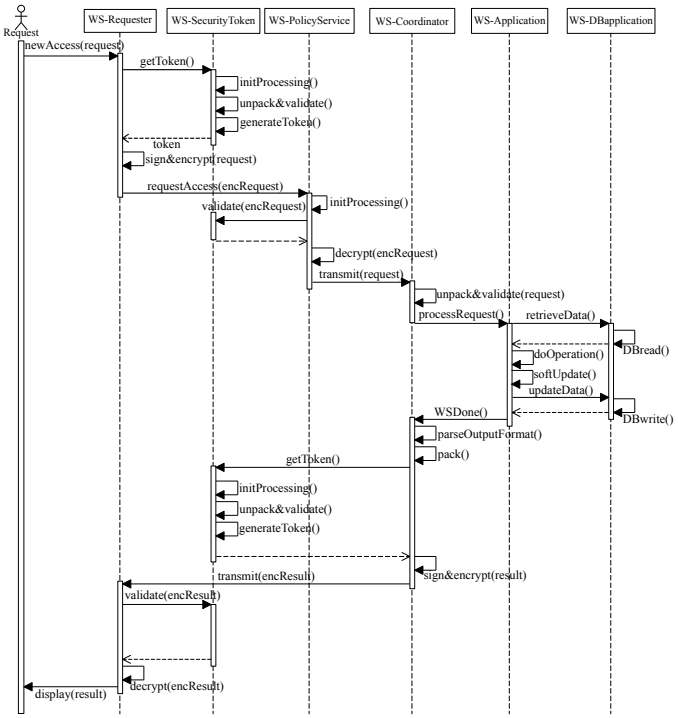


Figure 5. SDBS Update Customer's Data scenario.

transition t_3 represents the acquire of the security host, while t_7 represents the release of such a resource.

Each one of the activities, self-messages in Figure 5, has been transformed into an exponential transition in the PN with its corresponding duration (given in Table I(a)(c)). Each message, exchanged through a net, among two resources (e.g., *getToken()*) gives rise in the PN to an exponential transition (e.g., T_2) whose delay is that of the net involved (e.g., *$delayNet*). We have assumed that the operations/messages needed for establishing communication through the secure intranet are more expensive (in computing time terms). For this reason, we have set an upper delay for the secure intranet (*$intranetLag*) than for the insecure intranet (*$secIntraLag*). For simplicity, we have assumed the same delay for each message on the intranet communication irrespective of its size.

The workload is defined by the number of requests from users concurrently accessing the SDBS, which is parametrised by the variable *$nRequests*, an input parameter for the analysis. The number of hosts (security host, policy host, etc.) has been indicated using variables (*$nSec*, *$nPolicy*, etc.). Finally, the throughput of the intranets is considered through variables *$intranetLag* and *$secIntraLag*. Values for all these input variables used for the experiments in the next section and its corresponding transitions/places on the PN appear in Table I.

B. Experiments and Discussion

In this section we test our approach by performing a set of experiments in the Petri net that accurately represents the SDBS. After applying our approach, the obtained results will be discussed.

1) **Performance Estimation***:* We have carried out the regrowing strategy (algorithm in Figure 2, Section IV-C) to estimate the throughput of the SDBS system with a different number of requests. The overall strategy has been implemented in MATLAB, while throughput computation of the SDBS has been performed with the GreatSPN tool. The GreatSPN tool has been run in an Intel Pentium IV 3.6GHz with 2GiB RAM DDR2 533MHz host machine.

Table II shows the results obtained in the set of experiments with the parameters set as explained previously. The first column indicates the number of requests, followed by the number of *regrowing steps*. We have called regrowing step to each iteration of the loop of algorithm in Figure 2. For each number of requests considered in the experiments, we have simulated the whole system. Such results are indicated in the first row of each experiment. In the next column, it is shown the size of the bottleneck (in number of places and transitions) produced by the algorithm and its percentage with respect to the total size. Then, it is shown the result of the upper throughput bound computed by the algorithm. Such a bound is computed by solving the underlying Markov Chain when it is computationally feasible [12] or by simulating the net otherwise. Note that in case of simulation the upper throughput bound value is the mean of simulation values, and the real upper throughput bound value is within an interval of $\pm 4\%$ with a confidence level of 95%. The next two columns show, in first place, the percentage of increasing/decreasing improvement of one bound with respect to the previous upper throughput bound, and secondly, the accuracy of the computed bound with respect to the throughput of the whole system. The negative relative errors are caused by the confidence level and accuracy degree used in the experiments. Finally, the last column shows the execution time consumed for computing the upper throughput bound of the PN system. We have distinguished whether the computation of the upper throughput bound has been achieved by exact analysis († symbol) or by simulation (no symbol).

Note that the computation of the throughput of the whole system takes in all cases longer than one day of simulation time to finish, even the evaluated system is an academic example. For larger systems, simulations may need a long convergence time, and therefore the usefulness of bounds computation is proved.

The degree of precision (ε) of algorithm in Figure 2 has been set to 10^{-3}. As it can be observed, the initial bottleneck with lowest requests (15, 20) corresponds to the underlying state machine (it is the result of removing resource places from the net in Figure 6). Again, this result indicates that the system's resources are well-dimensioned for attending such a number of requests. In the case of 15 requests, in each iteration step there exists no significant improvement (near to 6% in two iterations) and the regrowing strategy finishes in few steps. However, the greatest improvement occurs when requests reach 20 units. In such a case, the first regrowing achieves an improvement near to 8%, reaching over 13% in the next iteration.

It is interesting to remark what happens when the requests are incremented up to 21. For that value, the initial bottleneck is produced by one of the system's resources (specifically, the number of DB application hosts). This implies that the

Transition	Parameter	Value(s)
T_1	*$reqRate*	$0.2ms$
T_2, T_8, T_{10}, T_{49}, T_{53}	*$delayNet*	$2.5ms$
T_{13}, T_{16}, T_{19}, T_{23}, T_{36}, T_{41}, T_{46}	*$intranetLag*	$0.2ms$
T_{26}, T_{29}, T_{31}, T_{34}	*$secIntraLag*	$0.5ms$
T_4, T_{43}	*$initProc1*	$1ms$
T_{15}, T_{22}, T_{52}	*$validate*	$0.3ms$
T_6, T_{45}	*$genToken*	$0.5ms$
T_9, T_{48}	*$sign*	$0.8ms$
T_{18}, T_{55}	*$decrypt*	$1ms$
T_{12}	*$initProc2*	$0.3ms$
T_5, T_{44}	*$unpack*	$0.1ms$
T_{40}	*$pack*	$0.1ms$
T_{39}	*$parseOutput*	$0.3ms$

(a)

Place	Parameter	Value(s)
p_0	*$nRequests*	$\{15, 20, 21, 22, 23 \dots 30\}$
p_7	*$nSec*	5
p_{18}	*$nPolicy*	10
p_{26}	*$nCoords*	10
p_{28}	*$nApps*	5
p_{31}	*$nDBapps*	2

(b)

Trans.	Parameter	Value(s)
T_{28}	*$DBread*	$0.2ms$
T_{30}	*$perform*	$0.6ms$
T_{32}	*$consistency*	$0.2ms$
T_{33}	*$DBupdate*	$0.2ms$
T_{56}	*$processResult*	$1.5ms$

(c)

Table I
EXPERIMENTS PARAMETERS.

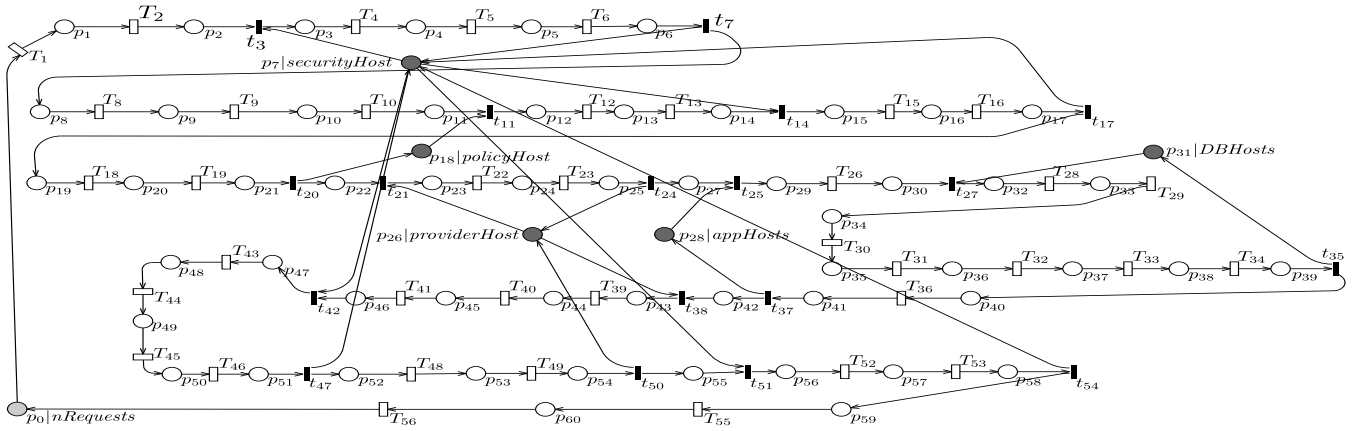


Figure 6. Petri net of the SDBS. Resource places are depicted in dark grey, whilst process-idle place in light grey.

Number of requests	Regrowing step	Size $\lvert P\rvert$ (%)	Size $\lvert T\rvert$ (%)	Through-put	Partial improvement	Bound error	Execution time (s)
15	(full system)	61 (100%)	56 (100%)	0.525685			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.551637	-	4.7045%	$5.87s$
	1	57 (93.44%)	56 (100%)	0.533037	3.3718%	1.3792%	$122.94s$
	2	58 (95.08%)	56 (100%)	0.522379	1.9995%	-0.6330%	$751.20s$
	3	59 (96.72%)	56 (100%)	0.522346	0.0063%	-0.6393%	$34256.97s$
20	(full system)	61 (100%)	56 (100%)	0.652313			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.735930	-	11.3621%	$5.80s$
	1	57 (93.44%)	56 (100%)	0.675957	8.1493%	3.4979%	$302.60s$
	2	58 (95.08%)	56 (100%)	0.637812	5.6431%	-2.2735%	$300.17s$
	3	59 (96.72%)	56 (100%)	0.637860	-0.0075%	-2.2658%	$3166.09s$
21	(full system)	61 (100%)	56 (100%)	0.671806			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	9.3063%	$0.18s^{\dagger}$
	1	57 (93.44%)	56 (100%)	0.697133	5.8871%	3.6331%	$826.82s$
	2	58 (95.08%)	56 (100%)	0.653556	6.2509%	-2.7924%	$280.46s$
	3	59 (96.72%)	56 (100%)	0.653116	0.0673%	-2.8616%	$2216.06s$
22	(full system)	61 (100%)	56 (100%)	0.687808			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	7.1459%	$0.18s^{\dagger}$
	1	57 (93.44%)	56 (100%)	0.713762	3.6422%	3.6362%	$2763.5s$
	2	58 (95.08%)	56 (100%)	0.666148	6.6709%	-3.2515%	$502.95s$
	3	59 (96.72%)	56 (100%)	0.667222	-0.1612%	-3.0853%	$1502.62s$
$23 \dots 30$	(full system)	61 (100%)	56 (100%)	0.700056			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	5.4925%	$0.18s^{\dagger}$
	1	14 (22.95%)	13 (23.21%)	0.740733	0.0011%	5.4915%	$0.262s^{\dagger}$

Table II
EXPERIMENT RESULTS FOR NUMBER OF REQUESTS $\{15, 20, 21, 22, 23 \dots 30\}$.

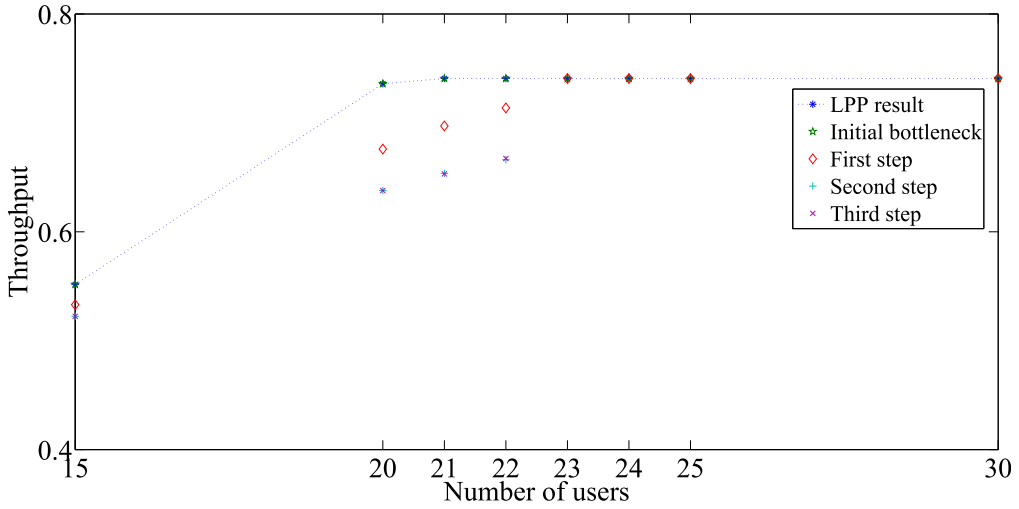


Figure 7. Throughput of the SDBS with variable number of users.

throughput bound of the system will keep the same for any number of requests over 21 (see *Average thr.* of first regrowing step for number of requests greater than 21). In other words, requests will start waiting to be attended if its number is higher or equal than 21. Besides, note that when the requests are greater than 23, in the second iteration step there is an improvement of the upper throughput bound lower than $10^{-3}\%$.

As it is said previously, the most important improvement occurs when the number of requests is 20. In just one iteration step, the initial throughput bound is improved in a value near to 8%. This indicates that the proposal method is more useful (i.e., it gets a significant improvement in the upper throughput bound in few iterations) if the resources and requests are more well-balanced. Besides, note that as it is shown by the execution time, the simulation of the whole PN becomes unfeasible for large systems.

The throughput results have been plotted in Figure 7. The throughput is drawn for each number of requests and for each step. Besides, the result of LPP (9) has also been drawn (dot line). LPP values match the throughput values of the initial bottleneck. As expected, the result of solving the LPP (9) (dot line) is an upper bound of all the rest of values. As it can be seen, the improvement of the upper throughput bound for each regrowing step is almost insignificant in the case of requests lower than 20 or greater than 25. While the number of requests reaches near to 20, the relative difference between the throughput of the initial upper throughput bound and the first iteration becomes greater, which reaches its maximum in the case of 20 requests. After that point, it becomes lower even tending to a minimum difference near to zero (see, for instance, the case of 30 requests).

Finally, the execution time shown in last column in Table II remarks that the bigger size of the net, the longer it takes to complete simulation. Note that small additions in the net (i.e., just one place) normally cause an execution time with one or two orders of magnitude than previous executions. However, the improvement of the upper thr. bound is not so significant to justify such an amount of execution time.

The main conclusions that can be extracted from both experiments can be summarised as follows:

- there exists a number of requests (*inflexion point*) from which the initially most restrictive p-semiflow of the system changes, and around such an inflexion point the accuracy of the initial throughput bound is low. This is motivated because when the slowest p-semiflow of the system is much slower than the others, it dominates over them and the system throughput is determined by the throughput of such a p-semiflow, so the initial throughput bound is usually quite accurate. However, when several p-semiflows have similar speeds, none of them dominates over the others, and hence the initial throughput bound, which considers just one p-semiflow, is less accurate;
- the improvement of the upper bound gets specially a significant improvement in the proximity of the inflexion point.

As future work, we pretend to keep researching on the performance estimation based on performance bounds, and trying to obtain some quality bound characterisation. The use of LP problems and the token/delay ratio between p-semiflows on a PN system could be useful for this goal.

As the reader can imagine, it would be nice if we were able to compute directly such inflexion points, which is the goal in the next set of experiments.

2) **Resource Optimisation***:* For these experiments, the number of requests has been set to $nRequests = 100$, whilst the initial number of resources remains unchanged: 5 security hosts, 10 policy hosts, 10 coordination hosts, 5 application hosts and 2 DB application hosts (summarised in Table I). Let us suppose a budget of $\$20,000$ and the following costs per resource: $\$3,500$ per security host (represented by place p_7), $\$1,000$ per policy host (place p_{18}), $\$2,000$ per coordinator host (place p_{26}), $\$500$ per application host (place p_{28}) and $\$500$ per DB application host (place p_{31}). The prices of the hosts are reflecting either the cost of the physical hardware or the cost of reimplementing the services.

Applying the optimisation strategy introduced in Section V, the initial restrictive resource is the number of DB application

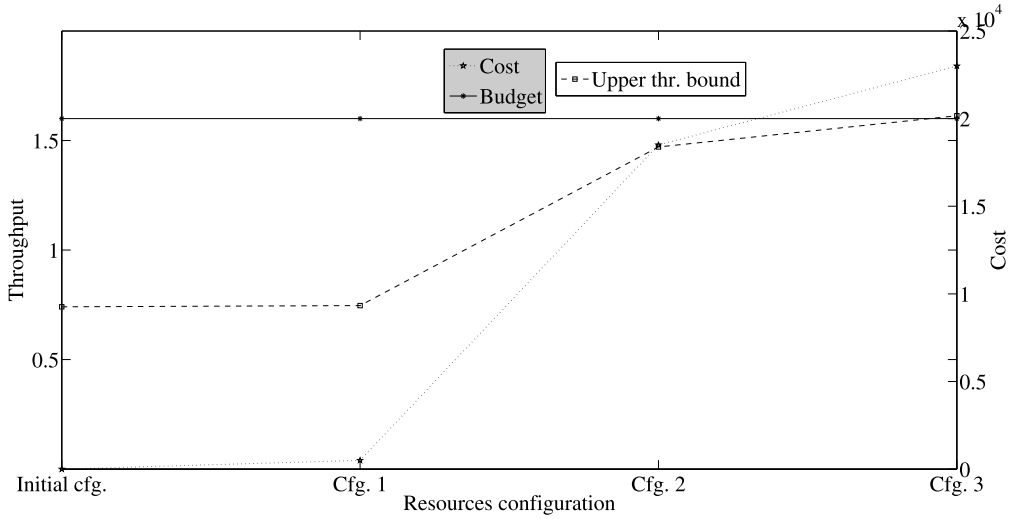


Figure 8. Different resources configuration and its associated cost.

hosts, $\$nDBapps$ (initial tokens of place p_{31}). The algorithm in Figure 3 computes the new restrictive resource, security hosts, and the number of DB application hosts needed to be incremented (which is just one host). As the cost of incrementing is $\$500$ per DB application host and there is a budget of $\$20,000$, it can be carried on. The strategy continues looking for the next restrictive resource. So, the second iteration gives as result the new restrictive resource (application host) and the new instances of DB application and security hosts, respectively, 2 and 5 units. The increment of such resources has a cost of $\$18,500$, so it can be afforded. The new restrictive resource, after the third iteration, is the number of coordinator hosts. This time, it is needed to increment in 6 units the security hosts, in 3 units the DB application hosts and in 1 unit the application hosts with respect to the initial configuration. This last assignment has a cost greater than the initial budget, so the iteration process finishes and the previous assignment is taken as the valid one (5 security hosts and 2 DB application hosts). Moreover, there is no possibility of spending the rest of the budget (which ascends to $\$1,500$) Therefore, the optimisation strategy ends.

Hence, with the initial configuration and the given budget, it is needed to increase the number of security hosts in 5 units and the number of DB application hosts in 2 units for getting the system resources optimally distributed and the throughput maximised.

Figure 8 plots the upper throughput bound (dash line) of each configuration of resources, its associated cost (dot line, in dollars) and the total assigned budget (solid line). *Initial cfg.* (initial configuration) is 5 security hosts, 10 policy hosts, 10 coordination hosts, 5 application hosts and 2 DB application hosts. *Cfg. 1* refers to the increment in one unit of DB application hosts, whilst *Cfg. 2* indicates the last assignment of resources computed: the increment in 5 security hosts and in 2 DB application hosts. Finally, *Cfg. 3* refers to the configuration which cannot be afford with such a budget ($\$20,000$): an increment in 6 units the security hosts, in 3 units the DB application hosts and in 1 unit the application hosts with respect to the initial configuration. As it can be observed in the Figure 8, the cost of the last resources configuration exceeds the assigned budget, so the solution for the resource distribution is the previous configuration.

It is also remarkable the evolution of the upper throughput bound. With the initial configuration, the upper throughput bound is $\Theta = 0.740740$. In the first configuration, the upper throughput bound increments in a 0.75% ($\Theta = 0.746271$), while in the second configuration it increments near to a 100% ($\Theta = 1.470598$). Finally, with the third configuration the upper throughput bound increases in a 9.68% ($\Theta = 1.612920$).

VII. Conclusions

The formalism of Petri nets allows one to model the behaviour of a large class of artificial systems in which resources are shared by the different tasks. The performance of these systems, which is usually measured as the number of completed operations per time unit, is often a system requirement. Unfortunately, in most cases of interest it is not possible to compute the exact performance of a system in a reasonable time due to the state explosion problem inherent to large discrete systems. Thus, the explosion problem poses difficulties not only to compute exactly the performance of an existing system, but also to design correctly new systems.

This paper has focused on the class of process Petri nets which allows a wide variety of modelling possibilities while offering interesting analysis properties. For this class of nets two iterative strategies have been proposed. The first one aims at estimating efficiently the performance of a given system. Such an estimation is carried out by computing increasingly larger system bottlenecks. The goal of the second strategy is, given an initial budget and a cost of each resource, to gauge the number of instances of each resource so that the system performance is maximised and the budget is not exceeded. This has been achieved by exploiting the linear dependence of the performance bounds with respect to the number of resources.

Given that both techniques make intensive use of linear programming techniques and the number of required iterations is usually low, their complexity and computational time are also low. The proposed strategies have been applied to a process Petri net modelling a Secure Database System. The performance of such a system has been evaluated for different workloads, and a distribution of resources that maximises the throughput for a given budget has been estimated. We have developed a tool, PeabraiN [31], which implements the strategies here presented to make easier their use to the practitioners. It enables to compute either performance estimation or resource optimisation in systems modelled with Petri nets. As future work, we plan to research on the quality of the initial upper bound obtained and to extend both strategies to more general Petri net classes.

REFERENCES

[1] J. Colom, "The Resource Allocation Problem in Flexible Manufacturing Systems," in *Applications and Theory of Petri Nets*, ser. LNCS, W. van der Aalst and E. Best, Eds. Springer Berlin / Heidelberg, 2003, vol. 2679, pp. 23–35.

[2] F. Tricas, "Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems," Ph.D. dissertation, Universidad de Zaragoza, May 2003.

[3] F. Tricas, F. Vallés, J. Colom, and J. Ezpeleta, "An Iterative Method for Deadlock Prevention in FMS," in *Discrete Event Systems. Analysis and Control*, R. Boel and G. Stremersch, Eds., Kluwer Academic Publishers. Boston, USA: Kluwer Academic Publishers, 08/2000 2000, pp. 139–148.

[4] J. Ezpeleta and R. Valk, "A Polynomial Deadlock Avoidance Method for a Class of Nonsequential Resource Allocation Systems," *IEEE T. Syst. Man. Cy. A.*, vol. 36, no. 6, pp. 1234–1243, 2006.

[5] N. Wu, M. Zhou, and Z. Li, "Resource-Oriented Petri Net for Deadlock Avoidance in Flexible Assembly Systems," *IEEE T. Syst. Man. Cy. A.*, vol. 38, no. 1, pp. 56–69, January 2008.

[6] J. Lopez-Grao and J. Colom, "On the Deadlock Analysis of Multi-threaded Control Software," in *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, September 2011, pp. 1–8.

[7] Z. Li, N. Wu, and M. Zhou, "Deadlock Control of Automated Manufacturing Systems Based on Petri Nets – A Literature Review," *IEEE T. Syst. Man. Cy. C.*, vol. 42, no. 4, pp. 437–462, July 2012.

[8] F. Tricas and J. Ezpeleta, "Computing Minimal Siphons in Petri Net Models of Resource Allocation Systems: A Parallel Solution," *IEEE T. Syst. Man. Cy. A.*, vol. 36, no. 3, pp. 532–539, 2006.

[9] S. Wang, C. Wang, M. Zhou, and Z. Li, "A Method to Compute Strict Minimal Siphons in a Class of Petri Nets Based on Loop Resource Subsets," *IEEE T. Syst. Man. Cy. A.*, vol. PP, no. 99, pp. 1–12, 2011.

[10] G. Chiola, C. Anglano, J. Campos, J. Colom, and M. Silva, "Operational Analysis of Timed Petri Nets and Application to the Computation of Performance Bounds," in *Proceedings of the 5th International Workshop on Petri Nets and Performance Models (PNPM)*. Toulouse, France: IEEE Computer Society Press, October 1993, pp. 128–137.

[11] J. Campos, G. Chiola, J. Colom, and M. Silva, "Properties and Performance Bounds for Timed Marked Graphs," *IEEE T. Circuits-I.*, vol. 39, no. 5, pp. 386–401, May 1992.

[12] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, ser. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[13] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Petri Nets," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, February 1974.

[14] R. J. Rodríguez and J. Júlvez, "Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing," in *Proceedings of the 7th European Performance Engineering Workshop (EPEW)*, ser. LNCS, vol. 6342. Springer, September 2010, pp. 175–190.

[15] Z. Liu, "Performance Bounds for Stochastic Timed Petri Nets," in *Proceedings of the 16th ICATPN*. Springer-Verlag, 1995, pp. 316–334.

[16] S. Haddad, P. Moreaux, M. Sereno, and M. Silva, "Product-Form and Stochastic Petri Nets: a structural approach," *Perform. Eval.*, vol. 59, pp. 313–336, March 2005.

[17] G. Casale, N. Mi, and E. Smirni, "Bound Analysis of Closed Queueing Networks with Workload Burstiness," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 13–24, June 2008.

[18] T. Osogami and R. Raymond, "Semidefinite Optimization for Transient Analysis of Queues," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, pp. 363–364, June 2010.

[19] J. Li, Y. Fan, and M. Zhou, "Performance Modeling and Analysis of Workflow," *IEEE T. Syst. Man. Cy. A.*, vol. 34, no. 2, pp. 229–242, March 2004.

[20] H. Wang and Q. Zeng, "Modeling and Analysis for Workflow Constrained by Resources and Nondetermined Time: An Approach Based on Petri Nets," *IEEE T. Syst. Man. Cy. A.*, vol. 38, no. 4, pp. 802–817, July 2008.

[21] K. V. Hee, H. Reijers, E. Verbeek, and L. Zerguini, "On the Optimal Allocation of Resources In Stochastic Workflow Nets," in *Proceedings of the 7th UK Performance Engineering Workshop*, K. Djemame and M. Kara, Eds., University of Leeds, Leeds, UK, 2001, pp. 23–34.

[22] Y.-L. Chen, P.-Y. Hsu, and Y.-B. Chang, "A Petri Net Approach to Support Resource Assignment in Project Management," *IEEE T. Syst. Man. Cy. A.*, vol. 38, no. 3, pp. 564–574, May 2008.

[23] J. Colom, J. Campos, and M. Silva, "On Liveness Analysis through Linear Algebraic Techniques," in *Proceedings of the Annual General Meeting of ESPRIT Basic Research Action 3148 Design Methods Based on Nets (DEMON)*, Paris, France, June 1990.

[24] H. Hu, M. Zhou, and Z. Li, "Liveness and Ratio-Enforcing Supervision of Automated Manufacturing Systems Using Petri Nets," *IEEE T. Syst. Man. Cy. A.*, vol. 42, no. 2, pp. 392–403, 2012.

[25] T. Murata, "Petri Nets: Properties, Analysis and Applications," in *Proceedings of the IEEE*, vol. 77, no. 4, April 1989, pp. 541–580.

[26] G. Florin and S. Natkin, "Necessary and Sufficient Ergodicity Condition for Open Synchronized Queueing Networks," *IEEE T. Software. Eng.*, vol. 15, no. 4, pp. 367–380, 1989.

[27] J. Campos and M. Silva, "Structural Techniques and Performance Bounds of Stochastic Petri Net Models," *Lecture Notes in Computer Science*, vol. 609, pp. 352–391, 1992.

[28] J. D. C. Little, "A Proof for the Queuing Formula: L= λ W," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.

[29] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*. North River Press, 1986.

[30] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer-Verlag, 2004.

[31] R. J. Rodríguez, J. Júlvez, and J. Merseguer, "PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation," in *Proceedings of the 12th International Conference on Application of Concurrency to System Designs (ACSD)*. IEEE, 2012, pp. 142–147.

APPENDIX

The strict inequality $\sum_{p\in V}\mathbf{y}(p) > 0$ in (10) is used to force that the components of places which belong to the next slowest p-semiflow are positive. Once the LPP (10) is solved, only the strictly positive components are selected. When the solver precision is not very high, zero components might not be distinguishable from positive components with low values. To avoid this, $\sum_{p\in V}\mathbf{y}(p) > 0$ is replaced by $\sum_{p\in V}\mathbf{y}(p) > h$, with a strictly positive h. Thus, we need to find a value $h > 0$ that keeps the feasibility of constraints $\mathbf{y}\cdot\mathbf{C} = \mathbf{0}$, $\mathbf{y}\cdot\mathbf{m_0} = 1$. A possible value h such that $\mathbf{y} \geq h\cdot\mathbf{1}$ and fulfils both equations can be calculated in the following way:

Recall that by the process PN structure, the number of p-semiflows is equal to $n+1$, where $n = |P_R|$ is the number of resources in the process PN system. Note as well that the initial marking $\mathbf{m_0}$ of the system will be $\mathbf{m_0}(p) > 0,\ \forall p \in P_R \cup P_0$, $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$. A p-semiflow $\mathbf{y}$ that covers all places can be computed by a linear combination of all minimal p-semiflows. Remember that each resource has associated a minimal p-semiflow (Property 3 of Definition 3).

Let us consider a system with n resources. Then, a linear combination of all minimal p-semiflows is $\mathbf{y} = \alpha_1\cdot\mathbf{y}_1 + \alpha_2\cdot$

$\mathbf{y}_2 + \cdots + \alpha_{n+1} \cdot \mathbf{y}_{n+1},\ \alpha_i > 0, \forall i \in \{1 \ldots (n+1)\}$. As $\mathbf{y}$ is a linear combination of p-semiflows, then $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ is fulfilled. However, factors α_i must be adjusted in order to properly fulfil equation $\mathbf{y} \cdot \mathbf{m_0} = 1$. An intuitive idea to make this is the following: as $\mathbf{y}(p) \cdot \mathbf{m_0}(p) > 0 \Leftrightarrow p \in P_R \cup P_0$, then $\mathbf{y} \cdot \mathbf{m_0} = 1$ can be reformulated as $\alpha_1 \cdot \mathbf{y}_1(p_{r_1}) \cdot \mathbf{m_0}(p_{r_1}) + \alpha_2 \cdot \mathbf{y}_2(p_{r_2}) \cdot \mathbf{m_0}(p_{r_2}) + \cdots + \alpha_{n+1} \cdot \mathbf{y}_{n+1}(p_{r_{n+1}}) \cdot \mathbf{m_0}(p_{r_{n+1}}) = 1$, where p_{r_i} represents the place associated to resource r_i, $\forall i \in \{1 \ldots n\}$, and $p_{r_{n+1}}$ is the process-idle place.

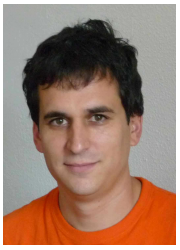
By the process PN structure, all positive values of $\mathbf{y}_i$ will be equal to one. Therefore, the values α_i that fulfil the equation $\mathbf{y} \cdot \mathbf{m_0} = 1$ can be easily calculated as:

$$\alpha_i = \frac{1}{\mathbf{m_0}(p_{r_i}) \cdot (n+1)},\ \forall i \in \{1 \ldots (n+1)\}$$

Hence, a possible value h that fulfils $\mathbf{y}(p) \geq h,\ \forall p \in P$ is, in this case, $h = min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$. Such a value is relating the number of resources in the system and the number of resources instances. Thus, the value of h for most systems of interest in practice is much higher than the numerical tolerance of the LPP solver (in this paper, the numerical tolerance of the LPP solver has been set to 10^{-5}).

As the objective function in LPP (11) is maximised, the value h obtained from that LPP will be, at least, equal to $min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$, that is: $h \geq min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$.

Ricardo J. Rodríguez is a Ph.D. candidate and a research assistant in the Computer Science and Systems Engineering Department at University of Zaragoza (Spain). He received his B.S. and M.S. in Computer Science Engineering from that university in 2008 and 2010, respectively. His research focuses on performance analysis and optimisation of large discrete event systems, secure software engineering and Fault-Tolerant systems. He has been visiting researcher in the School of Computer Science & Informatics at Cardiff University (United Kingdom). Rodríguez has been serving as a referee for international journals and for several international conferences and workshops. He is a member of the Aragon Institute of Engineering Research (I3A).



Jorge Júlvez received the M.S. and Ph.D. degrees in Computer Science Engineering in 1998 and 2005 from the University of Zaragoza (Spain). His Ph.D. was related to the study of qualitative and quantitative properties of continuous Petri nets. In 2005 he joined, as a PostDoc researcher, the Department of Software in the Technical University of Catalonia (Spain), where he spent three years. In 2008 he joined the Department of Computer Science and Systems Engineering in the University of Zaragoza (Spain), where he is currently associate professor. As a researcher, he has visited the Department of Electrical and Electronic Engineering in the University of Cagliari (Italy), the Department of Information Engineering in the University of Siena (Italy), the SYSTeMS Research group in the University of Ghent (Belgium), and the Computing Laboratory in the University of Oxford (United Kingdom). His current research is mainly related to the analysis and optimization of large discrete event systems.

José Merseguer is currently the Director of the Master in Computer Science and Systems Engineering at the University of Zaragoza, Spain. He teaches software engineering courses at graduate and undergraduate levels. Dr. Merseguer has developed postdoctoral research with Prof. Murray Woodside at Carleton University, Ottawa, Canada and with Prof. Robyn Lutz at Iowa State University, USA. He has also been visiting researcher at the Universities of Torino and Cagliari, Italy. His main research interests include performance and dependability analysis of software systems, UML semantics, and service-oriented software engineering. Dr. Merseguer received BS and MS degrees in computer science and software engineering from the Technical University of Valencia, and a PhD degree in computer science from the University of Zaragoza. Dr. Merseguer has been serving as a referee for international journals and as a program committee member for several international conferences and workshops. He is also a member of the Aragon Institute of Engineering Research (I3A).