Quantification and Compensation of the Impact of Faults in System Throughput

Ricardo J Rodríguez[†,*], Jorge Júlvez[‡], José Merseguer[‡]

[†]Babel Group, DLSIIS, Facultad de Informática

Universidad Politécnica de Madrid, Spain

[‡]Dpto. de Informática e Ingeniería de Sistemas

Universidad de Zaragoza, Spain

rjrodriguez@fi.upm.es, {julvez,jmerse}@unizar.es

Abstract

Performability relates the performance (throughput) and reliability of software systems whose normal behaviour may degrade due to the existence of faults. These systems, naturally modelled as Discrete Event Systems (DES) using shared resources, can incorporate Fault-Tolerant (FT) techniques to mitigate such a degradation. In this paper, compositional FT models based on Petri nets that make its sensitive performability analysis easier are proposed. Besides, two methods to compensate existence of faults are provided: an iterative algorithm to compute the number of extra resources needed, and an Integer-Linear Programming Problem (ILPP) that minimises the cost of incrementing resources and/or decrementing FT activities. The applicability of the developed methods is shown on a Petri net that models a secure database system.

[*]This work was partially supported by by CICYT - FEDER project DPI2010-20413 and by ARTEMIS Joint Undertaking nSafeCer under grant agreement no. 295373 and from National funding. Authors belong to the Group of Discrete Event Systems Engineering (GISED), partially co-financed by the Aragonese Government (Ref. T27) and the European Social Fund. **Corresponding author**: Ricardo J Rodríguez. Campus de Montegancedo, Facultad de Informática Dpto. de Lenguajes y Sistemas Informáticos e Ingeniería de Software, Universidad Politécnica de Madrid. 28660 Boadilla del Monte (Madrid), Spain. Email: `rjrodriguez@fi.upm.es`. Phone: (+34) 913365017 Fax: (+34) 913363669.

Keywords

Performability, Fault-Tolerant techniques, Petri nets, Integer-Linear Programming

1 Introduction

Performability [1] evaluates the performance (throughput) and the reliability of degradable systems, i.e., systems whose provided services may suffer some degradation due to errors and failures. Normally, degradable systems include Fault-Tolerant (FT) techniques [2, 3] that provide mechanisms to deal with failures inside the system and mitigate the consequences of faults. Some examples of FT techniques are: switching system requests between non-faulty components, adding watch-dogs for checking liveness of system components, or software exception handlers. A degradable system equipped with a FT technique is called a FT system.

Many FT systems are complex systems using shared resources that are compromised (i.e., they fail) by the activation of faults. These systems can be naturally modelled as Discrete Event Systems (DES) where resources are shared, also called Resource Allocation Systems (RAS) [4]. In this paper, we focus on FT systems using shared resources modelled as Petri nets (PNs) – more precisely, as process Petri nets [5]. This kind of PNs allows to model different instances of a single process that use shared resources (then competing among them) to complete. An extension of process Petri nets called S^4PR [5] can be used for modelling resource competition among structurally different processes.

Many studies evaluate the performability of a FT system through analytical models, usually represented as Markov processes [6,7]. These studies consider the FT systems modelled ad-hoc, and they do not provide any solution to mitigate the impact of activation of faults into the FT system. An evaluation of performability using Petri net-based models is presented in [8, 9]. Stochastic Activity Networks (SANs) are used in [8], associating reward rates directly with the markings of designated places and reward impulses with the completion of activities. Such an idea is extended for Generalised Stochastic Petri nets (GSPNs) by Bobbio in [9]. Another work that uses GSPN formalism is [10], where an extension of Fault Tree Analysis called Repairable Fault Trees (RFT) is presented. This extension allows the modelling and analysis of the repairing process by means of GSPNs.

A more recent approach is given by Reussner et al. in [11], where a compositional approach is presented using Markov chains as modelling formalism. Other works [12, 13] in the literature study the impact of error propagation on reliability, also focused on component-based systems.

Resource optimisation and its usage have been already studied for some class of Petri nets, namely Workflow Petri nets [14] or variants [15–17]. The work in [14] performs reduction operations on the original WF-net, having exponential complexity in the worst case. In [15], a method based on the reachability graph is presented. However, such a method can suffer scalability problems if the workflow size is large. Van Hee et al. give in [16] an algorithm to compute optimal resource allocation in stochastic WF-nets. Such an algorithm suffers as well from scalability problems because its complexity depends on the number of resources. In [17], Resource Assignment Petri Net (RAPN) is presented, that allows 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, RAPN is not able to model activities that utilise and release the same resource intermittently.

The contributions of this paper are threefold: firstly, we review the FT concepts [2, 3] and propose compositional PN models for FT techniques; secondly, we propose an iterative algorithm to compute the number of resources that mitigate the impact of activation of faults; and thirdly, we propose an Integer Linear Programming Problem (ILPP) that minimises the cost of compensation needed for maintaining a given throughput in a FT system.

Running example Let us consider a packet-routing algorithm inside a router where packets arrive and after checking source and destination of the packets, they are filtered following some defined rules. Figure 1 depicts a PN modelling such an algorithm. The PN marking represents the number nP of packets (initial marking of the process-idle place, p_0), the number nT of threads attending the incoming packets (initial marking of p_2) and the number nS of filtering-threads (initial marking of p_7). The number nC denotes the capacity of the system. We consider that this number is equal to the number nP of packets, therefore place p'_0 becomes implicit and we omit it for analysis. Packets arrive to the router following an exponential distribution of mean $\delta_0 = 5$ milliseconds[1]. The amount of time for checking packet headers (i.e., source, destination) is

[1]We use δ_i as an abbreviation for $\delta(T_i)$

represented by transition T_2, which follows an exponential distribution of mean $\delta_2 = 2$ milliseconds. The algorithm's decision is represented by the place p_5 and its outgoing arcs: either transition t_4 is fired (then the packet must be discarded, which happens with a probability of 0.75), or transition t_5 is fired. In the latter case, once some filtering-thread is available, it is used. Such a use is represented by T_7 and takes, on average, $\delta_7 = 1$ millisecond to complete. Finally, T_9 represents the final step of the algorithm, that consists in routing the packet(acknowledgement) properly to its destination(source) and takes, in terms of time, about 2 milliseconds, i.e., $\delta_9 = 2$.

[Figure 1 about here.]

This running example will be used henceforward to illustrate our approach. First, we will add to the PN depicted in Figure 1 a FT technique, and will compute the impact of faults in the system throughput. Then, we will apply our developed methods to compensate the throughput degradation.

The remainder of this paper is as follows. Section 2 introduces some basic concepts, such as FT concepts and Petri net theory. Then, Section 3 presents the proposed compositional PN models for FT techniques. Section 4 analyses, in first place, how conservative components are modified when adding the proposed PN models. It also presents the proposed iterative algorithm to compute the number of resources that mitigate the impact of activation of faults, and the ILPP that minimises the cost of compensation needed for maintaining a given throughput in a FT system. Section 5 shows a case study where both algorithms are tested. Finally, Section 6 summarises our findings and main contributions of this paper.

2 Preliminary Concepts

This section introduces some basic concepts that are needed to follow the rest of the paper. First, the concepts related to Fault Tolerance are introduced. Lastly, a background on Petri nets (PNs) and related concepts – such as upper throughput bounds – are introduced.

2.1 Fault Tolerance

Fault Tolerance (FT) aims at failure avoidance carrying out error detection and system recovery [2]. Figure 2 depicts the phases involved in a FT technique.

[Figure 2 about here.]

Error detection tries to identify the presence of an error in the system. It takes places either while the system is providing its services (concurrent), or when services are not being provided (preemptive). For instance, a hardware checking when the system boots up is a preemptive error detection technique.

Recovery techniques are aimed at handling possible errors and/or faults in the system and leading it to a state without detected errors. Recovery techniques may have two steps: an *error handling* (optional step), which tries to eliminate the presence of an error in the system; and *fault handling* (mandatory step), which tries to avoid the reactivation of the detected fault.

There are three common techniques when dealing with a detected error: *rollback*, when the system is conducted to a previous saved state (i.e., prior to error occurrence) without detected errors; *rollforward*, when the system is conducted to a new state without detected errors (in this case, later to error occurrence); and *compensation*, when there is enough redundancy to mask the error in the erroneous state.

Unlike rollback or rollforward that happen on demand, compensation may happen on demand or systematically, independently of the presence (or absence) of an error. For instance, an example of a compensation handling technique triggered on demand is an exception handler mechanism. In this paper, we consider that error handling takes place on demand.

The fault handling techniques that can be carried out to prevent faults from reacting again are: *diagnosis*, which records the origin (cause) of the error, locating where it happened and the type of error raised; *isolation*, which excludes (in a logical or physical way) faulty components from normal service delivery, so avoiding its participation in service delivery; *reconfiguration*, which reschedules service requests between non-failed components; and *reinitialisation*, which reconfigures the faulty system services by changing its configuration, stores this new configuration and reinitialises such affected services.

2.2 Petri Nets and Throughput Bounds

This section introduces some basic concepts regarding to the class of Petri net (PN) we are considering in this paper. Firstly, we define process Petri nets in the untimed framework. Then, timed Petri net systems and upper throughput bounds are defined. In the following, the reader is assumed to be familiar with Petri nets (see [18] for a gentle introduction).

2.2.1 Untimed Petri Nets

Definition 1. *A Petri net [18] (PN) 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 $\mathbf{m_0}$*.*

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.

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*.

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 marked Petri net $\mathcal{S}$ is *live* when every transition is live. In this paper, we assume that $\mathcal{S}s$ we work with are 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, $\mathbf{y}^\top \cdot \mathbf{C} = 0$. 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 support of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\| = \{i | \mathbf{v}(i) \neq 0\}$. A p-(or t-)semiflow $\mathbf{v}$ is *minimal* when its support 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. 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 net structure. 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 paper, we deal with Petri nets that model systems where resources are shared. Examples of this kind of systems can be found in manufacturing, logistics or web services systems. In general, these systems represent real-life problems where some items are processed and require the use of different resources (which are shared) during its processing. These systems can be naturally modelled in terms of process Petri nets, a subclass of Petri net whose inner structure is a strongly connected state machine. More formally:

Definition 3. *[5] A process Petri net (PPN) 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*,* $P_S \neq \emptyset$ *is the set of* process-activity places *and* $P_R = \{r_1, \ldots, r_n\}$*,* $n > 0$ *is the set of* resources places*;*

2. *The subnet* $\mathcal{N}' = \langle P_0 \cup P_S, T, \mathbf{Pre}, \mathbf{Post} \rangle$ *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. *$P_S = \bigcup_{r \in P_R}(\|\mathbf{y}_r\| \setminus \{r\})$. This implies that every place $p \in P_S$ belongs to the p-semiflow of at least one resource.*

Definition 3 implies that $PPNs$ are conservative and consistent. Intuitively, Definition 3 establishes a kind of nets where there is a process using different shared resources, every place in the net is covered by some p-semiflow and it uses some (at least one) resource, the number of instances of each resource remains constant and resources cannot change its type.

Let $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ be a PPN. A vector $\mathbf{m_0} \in \mathbb{Z}_{\geq 0}^{|P|}$ is called *acceptable initial marking* [5] of $\mathcal{N}$ if: 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*, i.e., number of items, 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 $\mathbf{m_0}$.

2.2.2 Timed Petri Nets

In order to be able to use Petri nets for systems performance evaluation, the inclusion of the notion of time must be considered. There are two ways of introducing the notion of time in Petri nets, either in places or transitions. Since transitions are representing the actions of a system, which have associated some duration, we associate such a duration to the firing delay of transitions [19]. Besides, we consider that the firing delays of transitions follow an exponential distribution functions.

A Petri net model where a set of exponential rates is considered (one for each transition in the model) is called a *Stochastic Petri net* (SPN) model [20,21]. These rates characterise the probability distribution function of the transition delay, which follow an exponential distribution function and are obtained as the inverse of the mean. These rates are considered to be marking-independent, i.e., its values are constant.

In this paper, we consider that the average service time of a transition t can be zero, i.e., it fires in zero units of time. These transitions are called *immediate transitions*. Otherwise, transition t is a *timed transition*. The exponential transitions are graphically represented by a white box, whilst immediate transitions are black boxes. It will be assumed that all transitions in conflict are immediate. An immediate transition t in conflict will fire with probability $\frac{\mathbf{r}(t)}{\sum_{t' \in A} \mathbf{r}(t')}$, where A is the set of enabled immediate transitions in conflict and $\mathbf{r}(t) \in \mathbb{N}_{>0}$ is the routing rate associated to transition t. The firing of immediate transitions consumes no time. When a timed transition becomes enabled, it fires following an exponential distribution with mean $\delta(t)$. More formally, we will consider the following timed Petri net classes:

Definition 5. *A* Stochastic Petri Net *(SPN)* [20] *system is a pair* $\langle \mathcal{S}, \delta, \mathbf{r} \rangle$ *where* $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *is a Petri net system,* $\delta \in \mathbb{R}^{|T|}_{\geq 0}$ *is a positive real function such that* $\delta(t)$ *is the mean of the exponential firing time distribution associated to transition* $t \in T$ *and* $\mathbf{r} \in \mathbb{N}^{|T|}_{>0}$ *is the vector of routing rates associated to transitions.*

Definition 6. *A* Stochastic Marked Graph *(SMG) is a Stochastic Petri net whose underlying Petri net is a Marked Graph.*

Definition 7. *A Stochastic Process Petri net (SPPN) system is a Stochastic Petri net system whose underlying Petri net is a Process Petri net.*

There exist different semantics for the firing of transitions, being *infinite* and *finite* server semantics the most frequently used. Given that infinite server semantics is more general (finite server semantics can be simulated by adding self-loop places), we will assume that the timed transitions work under infinite server semantics.

The average marking vector, $\overline{\mathbf{m}}$, in an ergodic [22] Petri net system is defined as [23]:

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

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 [23]:

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

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

By definition, all the places of a $SPPN$ are covered by p-semiflows, and therefore it is structurally bounded. In this work, we will assume that the $SPPN$ under study is a live and structurally bounded net with Freely Related T-semiflows (i.e., a FRT-net) [24]. It is known that the Markov process that describes the time evolution [21] of these nets is ergodic [24], i.e., when the observation period tends to infinite, the estimated values of average marking and steady-state throughput tend to a certain value, 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:

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

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

The visit ratios of two different transitions t, t' in equal conflict must be proportional to the corresponding routing rate $\mathbf{r}(t), \mathbf{r}(t')$ defining the conflict resolution condition $\mathbf{r}(t) \cdot \mathbf{v}^{t_i}(t') = \mathbf{r}(t') \cdot \mathbf{v}^{t_i}(t)$. This condition can be also written in vector form as:

$$\mathbf{R} \cdot \mathbf{v}^{t_i} = 0 \tag{4}$$

where $\mathbf{R}$ is a matrix containing as many rows as pairs of transitions in equal conflict.

In FRT-nets, the vector of visit ratios $\mathbf{v}$ exclusively depends on the structure of the net and on the routing rates [24]. 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 [24]:

$$\begin{gathered} \begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v}^{t_i} = 0 \\ \mathbf{v}^{t_i}(t_i) = 1 \end{gathered} \tag{5}$$

2.2.3 Performance Estimation

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) [24]:

$$\begin{aligned} \mathbf{\Gamma}(t_i) \geq \mathbf{\Gamma^{lb}}(t_i) = \ & \mathit{maximum}\ \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D^{t_i}} \\ & \mathit{subject\ to}\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{6}$$

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*, $\mathbf{D^{t_i}}(t) = \delta(t) \cdot \mathbf{v}^{t_i}(t)$ (the vector of visit ratios $\mathbf{v}^{t_i}$ is normalised for transition t_i)[2].

As a side product of the solution of (6), $\mathbf{y}$ represents the *slowest* p-semiflow of the system, thus LPP (6) can also be seen as a search for the most constraining p-semiflow. This p-semiflow will be the one with highest ratio $\dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$. 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, $\Theta(t_i) = \dfrac{1}{\mathbf{\Gamma^{lb}}(t_i)}$.

Let us recall that the vector of average service times of transitions δ does not depend on the marking. Otherwise, LPP 6 could not be applied, basically because having a δ depending on the marking will lead to a non-linear programming problem.

3 Compositional PN Models for Fault Tolerance

In this section, we provide compositional PN-based models for the Fault-Tolerant (FT) techniques based on the basic concepts of FT given in Section 2.1. Recall that a FT technique may involve both error detection – concurrent or preemptive – and recovery phases – divided in error handling (rollback, rollforward or compensation) and fault handling (diagnosis, isolation, reconfiguration or reinitialisation).

[2] In the sequel, we omit the superindex t_i in $\mathbf{D^{t_i}}$ for clarity

[Figure 3 about here.]

Consider we have a system modelled with a PN in which there is an activity (represented by a timed transition T_f) which is subject to fail. We called it *faulty transition*, as it may lead to a fault. Before adding any FT technique to the system, we apply a transformation rule $\mathcal{TR}$ in the PN. This transformation rule allows us to apply our approach in general case, and it is not modifying the behaviour of the original PN model anyhow.

Figure 3 shows how this transformation rule $\mathcal{TR}$ works: an immediate transition $t(t')$ and place ${}^\bullet T_f(T_f^\bullet)$ are added just from(to) transition T_f, and all input(output) places of transition T_f are accordingly connected to transition $t(t')$.

[Figure 4 about here.]

Figure 4 depicts the interaction between a PN that models the behaviour of a given system and a PN that models a FT technique. A PN-based FT model is subdivided in *Error Detection* and *Recovery* sub-models. Each sub-model respectively represents the phases involved in a FT technique. In the sequel, we explain each model and its interactions in detail.

3.1 PN Error Detection Model

Figure 5(a) depicts the PN model for error detection. The timed transition T_{detect} represents how long the error detection activity takes. Note that this transition is abstracting the behaviour for detecting an error, so that it may be refined into a more complex model representing error detection in more detail (*Detection phase* in Figure 5(a)). After error detection activity takes place, the presence of an error is discriminated. When an error arises (transition t_{err}), then a token is put on place $p|_{eed}$. Otherwise, a token is put on place $p|_{ned}$.

The integration between the Error Detection model and the System model is done through labelled places $p|_{sed}, p|_{eed}$ (a labelled place p is defined as $p|_{label}$). We have followed the compositional rules over the places defined in [25, 26] to combine models using labelled places: pairs of places with matching labels are superposed. Figure 5(a) depicts the places $p|_{sed}, p|_{ned}$ added to the system model. The origin of the incoming arc of place $p|_{sed}$ depends on the type of error detection, and synchronises the execution of error detection model with the system model: when concurrent, the arc added is the red-dashed one; otherwise (preemptive), the green-dotted arc is considered. Note that the place $p|_{ned}$ is synchronised with $T_f^\bullet$ (which is added to the system by transformation rule $\mathcal{TR}$).

[Figure 5 about here.]

This simple model allows us to represent the most common error detection techniques, e.g., to validate input data, or intermediate data generated and reused during faulty transition (it can be concurrently done), and to validate output after faulty transition execution (preemptive).

3.2 PN Recovery Model

Recovery phase involves two steps, a first (optional) step of error handling (rollback, rollforward or compensation) and a second one of fault handling technique (diagnosis, isolation, reconfiguration or reinitialisation).

Following the definitions given in [2], we have grouped the fault handling techniques in two groups: diagnosis and reinitialisation techniques; and isolation and reconfiguration. This decision is based on the abstracted behaviour of these techniques, as we explain henceforward. We have composed models that represent valid combinations of the recovery phase as it is shown in Table 1. This classification is made based on how the techniques work. For instance, we believe that a rollforward technnique cannot be combined with reconfiguration or reinitialisation, because reconfiguration switches the request to spare components, while reinitialisation updates and records a new system configuration. Thus, we consider that to move to a future correct state after recovering is unmeaning.

[Table 1 about here.]

Figure 6(a) shows the PN model of diagnosis and reinitialisation FT recovery techniques. Place $p|_{eed}$ is superposed with the one of *Error Detection* model, and place $p|_{T_f^\bullet}$ is superposed with place $T_f^\bullet$ in the system model. A token in place $p|_{eed}$ indicates that an error has been detected. Once transition t_{rm} is fired, a (optional) compensation activity may take place (*Compensation phase*). Then, recovery activity takes place (abstracted in *Recovery phase*). As in the previous model of error detection, we have represented compensation and recovery phases as a single timed transitions (T_c and T_{rec}, respectively). These transitions may be refined into a more complex models representing compensation and recovery activities in more detail.

Finally, the token flow is redirected through place $p|_{rtn}$. The superposition of this place depends on the error handling technique used: it will be a place which becomes eventually marked *after* the faulty transition T_f is fired (rollforward), or which was eventually marked *before* its firing (rollback). In both cases and to keep conservativeness of the model, place $p|_{rtn}$ must belong to the p-semiflow associated to the resource r (we called it *faulty resource*), being r the inner resource used by faulty activity. Although a transition T_f can represent an activity where several resources are being used, for the sake of simplicity in this paper we assume that the fault is caused by the use of the inner resource (i.e., the last one acquired). Otherwise, note that after recovering phase other resources acquired after faulty resource should be released to keep conservativeness.

The difference between diagnosis and reinitialisation technique can be established by the duration of recovery phase. For instance, when diagnosis technique is considered, the recovery phase will have a much lower duration than when reinitialisation is taken into account due to the actions that are performed.

[Figure 6 about here.]

Figure 6(b) shows the PN model of isolation and reconfiguration FT recovery techniques. This case is identical to the previous until the (optional) compensation phase. After the compensation phase takes place, the type of the fault is discriminated [2] as intermittent (that is, the fault is transient) or solid (i.e., the faults whose activation is reproducible). When the fault is intermittent, as proposed in [2], normal execution can keep going on and token is returned to place $p|_{rtn}$ (as before, the superposed place depends on the type of error detection). On the contrary, when a solid fault is detected, the faulty resource is excluded from normal service delivery – as indicated by both isolation and reconfiguration techniques – and the token is moved to the place $p|_{safe}$. We assume that place $p|_{safe}$ is superposed with the place previous to acquire the faulty resource r, i.e., $p|_{safe} = {}^\bullet t_{acq}$, where t_{acq} is the transition where faulty resource r is acquired.

In the case of isolation and reconfiguration, the recovery phase is called *Maintenance phase*, because it involves the participation of an external agent [2]. We have modelled maintenance phase as a single transition T_{MTTR} that represents the Mean Time To Repair (MTTR) spent on fixing the faulty resource. As in the previous case, this model can be refined to a more complex maintenance model. Anyhow, after maintenance phase takes place the fixed resource is returned to place $p|_{ir}$, which is superposed to the resource place p_r.

As in the previous techniques, the difference between isolation and reconfiguration technique can be established by the duration of maintenance phase. For instance, when isolation technique is considered, the maintenance phase will have a much greater duration than when reconfiguration is taken into account.

Finally, note that most of the FT techniques can be modelled with the proposed models. For instance, a *watchdog* can be modelled as a reconfiguration FT technique with concurrent error detection and rollforward (or rollback), and a *checkpointing and rollback* can be modelled as a reinitialisation FT technique. Unfortunately, other FT techniques, such as *n-version programming* or *combined proactive-reactive techniques* [27] cannot be adapted to the proposed model and some tweaks must be done. We aim to extend these models to cover all FT techniques as a future work.

[Figure 7 about here.]

Recall the PN of the running example depicted in Figure 1. Suppose that the filtering activity may fail, i.e., the faulty transition is T_7. The router manufacturer is interested in adding a watchdog (recall it can be modelled as a reconfiguration FT technique) into the algorithm such that the threads that fail (they are hanged) are discarded, and they are cleaned with a fixed internal timer. In this case, the error detection model is concurrent, as the failure can be detected during normal operation; and the error handling technique used is rollback: when an error is detected, the packet is filtered by another thread, when available.

The resulting PN after adding the FT technique described above is depicted in Figure 7. We assume that the detection activity takes, on average, $\delta_{detect} = 0.5$ milliseconds, and the recovery activity takes, on average, $\delta_{MTTR} = 2$ seconds. Let us suppose a probability of raising an error of 0.2, resulting the 5% of the times in a solid fault. This PN will be used in the next section for sensitive performability analysis.

4 Analysis of PN-based FT Models

This section introduces, in first place, how the conservative components (i.e., the p-semiflows) are modified when FT models are added to a PPN. Then, we perform a sensitive analysis on upper throughput bound of the PPN system with respect to the failure probabilities. Lastly, we propose an optimisation technique that tries to compensate the throughput degradation produced by the existence of faults.

4.1 Conservative Components

Let us analyse how minimal p-semiflows are modified. The addition of the proposed FT models transforms each p-semiflow $\mathbf{y}_r$ associated to a resource r that makes use of the faulty transition t_f (i.e, $\|\mathbf{y}_r\| \cap \{^\bullet t_f, t_f^\bullet\} \neq \emptyset$) into two p-semiflows $\mathbf{y}'_r, \mathbf{y}''_r, \mathbf{y}'_r \neq \mathbf{y}''_r$ such that $\|\mathbf{y}_r\| \subset \|\mathbf{y}'_r\|, \|\mathbf{y}_r\| \subset \|\mathbf{y}''_r\|$. This transformation is due to the fact that FT models consume/produce tokens from/to the original p-semiflows. These p-semiflows cover all places added by the FT technique, thus the net remains conservative.

[Table 2 about here.]

For instance, the minimal initial p-semiflows of the net in Figure 1 are: $\mathbf{y}_1 = \{p_0, p_1, p_3, p_4, p_5, p_6|_{safe}, p_8|_{rtn}, p_9, p_{10}, p_{11}\}$, $\mathbf{y}_2 = \{p_2, p_3, p_4, p_5, p_6|_{safe}, p_8|_{rtn}, p_9, p_{10}, p_{11}\}$ and $\mathbf{y}_3 = \{p_7|_{ir}, p_8|_{rtn}, p_9\}$. The minimal p-semiflows of the PN in Figure 1 that contain places from/to transition T_7 ($p_8|_{rtn}$ and p_9, respectively) are $\mathbf{y}_1$,$\mathbf{y}_2$ and $\mathbf{y}_3$. Thus, the new p-semiflows of PN in Figure 7 are the ones showed in Table 2.

Note that these new p-semiflows violate the third property of definition of PPN (see Section 2), given that there exist more than a single minimal p-semiflow containing the same resource, e.g., $\mathbf{y}'_2$ and $\mathbf{y}''_2$ contain the resource place p_2 on its support. Nevertheless, in the new net system it still holds that each minimal p-semiflow contains only one initially marked place.

4.2 Sensitive Analysis of Upper Throughput Bounds

As we have seen in the previous section, the p-semiflows of the PPN change once some of the proposal FT models are added. Recall that an upper throughput bound Θ of a PPN system is related to the slowest p-semiflow $\mathbf{y}$, i.e., $\Theta = \dfrac{\mathbf{y} \cdot \mathbf{m_0}}{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}$.

Given that in the considered nets all the components of minimal p-semiflows are equal to 1 and the only initially marked places are resource places, i.e., $\forall p \in \|\mathbf{y}_r\| \setminus \{r\}, \mathbf{m_0}(p) = 0$, the previous equation can be written as $\Theta = \dfrac{\mathbf{m_0}(r)}{\mathbf{y}_r \cdot \mathbf{Pre} \cdot \mathbf{D}}$ where $\mathbf{y}_r$ is minimal. Let us assume that after adding some FT technique, there are n minimal p-semiflows, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ that are modified. Thus, the throughput bound of the new net system is:

$$\Theta' = minimum(\Theta, minimum_{i=1}^{n} \frac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}) \tag{7}$$

where $\mathbf{y}_i$ is a minimal p-semiflow, i.e., $\forall p \in \|\mathbf{y}\|, \mathbf{y}(p) = 1$.

Recall the running example of the previous section. Suppose an initial marking of $nP = 10$, $nT = 2$ and $nS = 2$. The slowest p-semiflow is, with this configuration and before adding the FT technique (Figure 1), $\mathbf{y} = \{p_2, p_3, p_4, p_5, p_6|_{safe}, p_8|_{rtn}, p_9, p_{10}, p_{11}\}$; and the upper throughput bound is $\Theta = 0.470588$. After adding the proposed FT technique, the equations $\dfrac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}$ related to p-semiflows that change are:

$$\begin{aligned}
\mathbf{y}'_1 &\rightarrow \frac{\mathbf{m_0}(p_0)}{\delta_0 \cdot \mathbf{v}_0 + \delta_2 \cdot \mathbf{v}_2 + \delta_7 \cdot \mathbf{v}_7 + \delta_9 \cdot \mathbf{v}_9} \\
\mathbf{y}''_1 &\rightarrow \frac{\mathbf{m_0}(p_0)}{\delta_0 \cdot \mathbf{v}_0 + \delta_2 \cdot \mathbf{v}_2 + \delta_{detect} \cdot \mathbf{v}_{detect} + \delta_9 \cdot \mathbf{v}_9} \\
\mathbf{y}'_2 &\rightarrow \frac{\mathbf{m_0}(p_2)}{\delta_2 \cdot \mathbf{v}_2 + \delta_7 \cdot \mathbf{v}_7 + \delta_9 \cdot \mathbf{v}_9} \\
\mathbf{y}''_2 &\rightarrow \frac{\mathbf{m_0}(p_0)}{\delta_2 \cdot \mathbf{v}_2 + \delta_{detect} \cdot \mathbf{v}_{detect} + \delta_9 \cdot \mathbf{v}_9} \\
\mathbf{y}'_3 &\rightarrow \frac{\mathbf{m_0}(p_2)}{\delta_7 \cdot \mathbf{v}_7 + \delta_{MTTR} \cdot \mathbf{v}_{MTTR}} \\
\mathbf{y}''_3 &\rightarrow \frac{\mathbf{m_0}(p_0)}{\delta_{detect} \cdot \mathbf{v}_{detect} + \delta_{MTTR} \cdot \mathbf{v}_{MTTR}}
\end{aligned} \tag{8}$$

Note that as error detection is concurrent, there is no p-semiflow containing both faulty transition and error detection transition at the same time. Otherwise, the faulty transition appears in conjunction with error detection transition in all p-semiflows generated. Besides, in the case of concurrent error detection, the number of minimal p-semiflows to be checked can be simplified, taking only the generated one that it is $max(\delta_{detect}, \delta_{T_f})$. Thus, the p-semiflows of interest here are: $\mathbf{y}'_1$, $\mathbf{y}'_2$ and $\mathbf{y}'_3$ (as $\delta_7 > \delta_{detect}$). The throughputs of these p-semiflows are, respectively, $\Theta_1 = 1.073825$, $\Theta_2 = 0.463768$ and $\Theta_3 = 0.304762$.

Therefore, the new slowest p-semiflow is $\mathbf{y}'_3$, and the new upper throughput bound is $\Theta' = \Theta_3 = 0.304762$. That is, with the described configuration, *the addition of an isolation FT technique causes a degradation of* 35.23% *to the upper throughput bound of the system.*

We have performed a sensitive analysis of Θ_1, Θ_2 and Θ_3 with respect to the probability of errors $r_e, r_e \in [0 \dots 1]$, taking steps of 0.01. The results are plotted in Figure 8(a). The solid line is Θ, the upper throughput bound of the original system. The dotted line is Θ_1, while dot-dashed is Θ_2 and dashed line is Θ_3.

[Figure 8 about here.]

The findings show that Θ_2 is a bit lower than the original upper throughput bound for low probabilities of error. This holds until the probability of error reaches a value near to 0.14. From that point, Θ_3 becomes the new upper throughput bound, which besides exponentially decreases. It is remarkable that $\mathbf{y}'_3$, i.e., the p-semiflow associated to Θ_3, is even faster than the others for low probabilities of error ($r_e < 0.06$). Lastly, when probability of error reaches a value near to 0.8, the throughput of all minimal p-semiflows quickly decreases and tends to zero.

4.3 Resource Assignment

This section introduces an iterative strategy that computes the number of resources needed to maintain a given upper throughput bound in a degradable system where our proposed FT models are added.

Such a strategy is presented in Algorithm 1. As input, it needs the description of the PN model with the FT techniques added to it with the initial marking and the vector of service times of transitions, $\langle \mathcal{N}, \mathbf{m_0}, \delta \rangle$; the upper throughput bound Θ before adding the FT techniques; and the set $\mathbf{Y}^{FT}$ of minimal p-semiflows that are modified after adding the FT techniques. As output, it returns the initial marking $\mathbf{m'_0}$ such that the upper throughput bound Θ' of the FT system is greater than or equal than Θ.

Algorithm 1 works as follows. It iterates in the content of the set $\mathbf{Y}^{FT}$ of minimal p-semiflows that have been modified when adding a proposed FT model. For each minimal p-semiflow $\mathbf{y}_i \in$

Algorithm 1 An iterative algorithm to compute initial marking needed to maintain a certain upper throughput bound with a probability of error.

Input: $\langle \mathcal{N}, \mathbf{m_0}, \delta \rangle$, Θ, $\mathbf{Y}^{FT}$
Output: $\mathbf{m'_0}$
$\mathbf{m'_0} = \mathbf{m_0}$
for each $\mathbf{y}_i \in \mathbf{Y}^{FT}$ **do**
  $\mathbf{m'_0}(r_i) = maximum(\mathbf{m_0}(r_i), \lceil (\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}) \cdot \Theta \rceil)$
end for each

$\mathbf{Y}^{FT}$, the value of the initial marking for associated resource r_i is computed as the maximum of the previous initial marking of the resource (i.e., $\mathbf{m_0}(r_i)$) or the $\lceil (\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}) \cdot \Theta \rceil$. The latter equation comes from solving $\Theta = \dfrac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}$. The ceiling is needed because $\mathbf{m'_0}(r_i) \in \mathbb{N}$.

Let us apply the Algorithm 1 in the running example. The previous upper throughput bound is $\Theta = 0.470588$, and the set of minimal p-semiflows that are modified after adding isolation FT is $\mathbf{Y}^{FT} = \{\mathbf{y}'_1, \mathbf{y}'_2, \mathbf{y}'_3\}$. For a given initial marking $\mathbf{m_0}(p_0) = 10, \mathbf{m_0}(p_2) = 2, \mathbf{m_0}(p_7) = 2$, Algorithm 1 returns as solution: $\mathbf{m'_0}(p_0) = \mathbf{m_0}(p_0) = 10, \mathbf{m'_0}(p_2) = 3, \mathbf{m'_0}(p_7) = 4$. That is, it is needed another thread and two more filtering-threads to compensate a 20% of errors (and a 5% of them deriving in solid faults) using reconfiguration as FT technique.

We have plotted in Figure 8(b) the initial marking needed to support the given throughput of $\Theta = 0.470588$ varying the probability of error $r_e, r_e \in [0 \ldots 1]$, taking steps of 0.01. The dotted line is the initial number of tokens of p_0 (packets, nP), the solid line corresponds to the initial number of tokens of p_2 (threads, nT) and the dashed line is the initial number of tokens of p_7 (filtering-threads, nS). The results show that the number of packets and threads remain more or less equal, i.e., there is no need to increment too much units to be able to maintain the given throughput, even with high probability of errors. However, the number of filtering-threads needed increases rapidly with respect to the probability of error.

4.4 Minimising Cost of Compensating Throughput Degradation

In this section, we present an Integer-Linear Programming Problem (ILPP) that minimises the cost of compensating throughput degradation caused by the presence of errors.

We are able to compute the initial marking needed to maintain a given throughput with the previous Algorithm 1. However, the increment of items of resources can have a cost in real systems and we may not be able to increment as much as it is desired. Recall that equation $\dfrac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}$ relates not only the number of items of resources ($\mathbf{m_0}(r_i)$) but also activity timings and error (and solid faults) probabilities ($\mathbf{D}$). If we consider a given error probability r_e and solid faults probability r_s, a compensation may be done in two ways: either the number of resources in the system can be incremented, or the timing of FT activities (detection, compensation and recovery phases) can be decremented. Both ways can have some cost associated.

Let us assume that FT phases are abstracted in single timed transition, i.e., a FT technique j adds to the system three timed transition: T^j_{detect} (detection phase), T^j_c (compensation phase) and T^j_{rec}/T^j_{MTTR} (recovery/maintenance phase). Let c^r_i the cost of an increment of one unit of the resource r_i, and c^d_j the cost of a decrement of one unit of time of detection phase of FT technique j, while $c^c_j(c^{rm}_j)$ is the cost of a decrement of one unit of time of compensation(recovery/maintenance) phase.

We can build an Integer-Linear Programming Problem (ILPP) to compute the minimum cost that guarantees a compensation of the throughput system after adding a number m of FT techniques as follows:

$$
\begin{array}{rcl}
minimum & & \left(\sum_{i=1}^{n} c^r_i \cdot \alpha_i + \sum_{j=1}^{m} \left(c^d_j \cdot \beta^d_j + c^c_j \cdot \beta^c_j + c^{rm}_j \cdot \beta^{rm}_j \right) \right) subject\ to \\
\mathbf{m_0}(r_i) + \alpha_i & \geq & \Theta \cdot \mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}' \\
\delta'(T^j_{detect}) & = & \delta(T^j_{detect}) - \beta^d_j \\
\delta'(T^j_c) & = & \delta(T^j_c) - \beta^c_j \\
\delta'(T^j_{rec}) & = & \delta(T^j_{rec}) - \beta^{rm}_j \\
\delta'(t) & \geq & \delta_{min}(t), \forall t \in T \\
\alpha_i, \beta^d_j, \beta^c_j, \beta^{rm}_j & \geq & 0, \alpha_i \in \mathbb{N}, \forall i \in [1 \ldots n], \forall j \in [1 \ldots m]
\end{array}
\tag{9}
$$

where n p-semiflows have been modified by the addition of m FT techniques to the original system; $\mathbf{D}'(t) = \delta'(t) \cdot \mathbf{v}(t), \forall t \in T$; and $\delta_{min}(t)$ is a lower bound for the service time of transition t (that is, we impose a minimum service time for transitions). The new number of resources and firing of transitions will be given by the values of $\alpha_i, \beta_j^d, \beta_j^c, \beta_j^{rm}$, respectively.

This ILPP is applied to the case study in the next section.

5 Case Study: a Secure Database System

This section introduces a case study to test our approach. We have considered the design of a Secure Database System (SDBS) deployed as a Web Service that stores confidential data and keeps traceability of all operations made over the data. Examples of this kind of system are a medical insurance company (that keeps customer's medical data), or a bank company (that keeps customer's balance accounts).

The UML-Sequence Diagram in Figure 9 models how SDBS works when a user requests an operation on its stored data (for instance, a bank customer asks for all operations made on its bank accounts). When a new request arrives at the system (attended by `WS-Requester`), it asks for a security token that is provided by `WS-SecurityToken`. Once it is provided, the request is accordingly encrypted and set to `WS-PolicyService`, where it is validated, decrypted and transmitted to `WS-Coordinator`. Finally, `WS-Coordinator` unpacks the request and sends it to the `WS-Application`, which accesses the database through `WS-DBApplication` service via a secure intranet. An acknowledgement is sent back through the system to the origin of the request, reporting the results to the user. Note that the result also needs a security token to be securely transmitted back to the user.

[Figure 9 about here.]

Figure 10 depicts the Petri net (PN) corresponding to the behaviour of the SDBS system described in Figure 9. The transformation from UML to PN is documented in [28], and can be carried out by several tools, such as ArgoPN, ArgoPerformance [28] or ArgoSPE [29]. Each resource is represented by a dark grey place in the PN: p_2 (`WS-Requester`), p_5 (`WS-PolicyService`), p_{13} (`WS-SecurityToken`), p_{24} (`WS-Coordinator`), p_{29} (`WS-Application`) and p_{32} (`WS-DBApplication`); while user's requests are represented by the process-idle place p_0 (depicted in light grey). As the running example, we consider that there is a place p_0' with the same initial marking that p_0, thus it becomes implicit and it is not considered for the analysis (indeed, we omitted it in the Figure 10). The number of instances of each resource is summarised in Table 3(b), and they will be represented by tokens in the respective place. Due to the state explosion problem the computation of the number of states with this configuration using different tools (e.g. `PeabraiN` [30] or GreatSPN [31]) has not been possible in reasonable time in a Intel Pentium IV 3.6GHz with 3GiB RAM DDR2 533MHz host machine.

The acquire (release) of a resource is represented by an immediate transition with an input (output) arc. For example, transition t_2 represents the reception of the request by the `WS-Requester` service, while t_7 represents the release of such a resource.

[Figure 10 about here.]

[Table 3 about here.]

Consider that transition that represents an operation on data after reading the DB, T_{31}, may fail with a probability of 0.15. We decide to add a reinitialisation FT technique FT^1, without compensation phase and with a concurrent error detection that takes, on average, $\delta(T^1_{detect}) = 0.5ms$. The recovery time, i.e., the time needed for reconfiguring DB service takes, on average, $\delta(T^1_{rec}) = 20ms$. Lastly, place p_{36} (the one before faulty transition T_{31}) is labelled as $p_{36}|_{rtn}$.

The upper throughput bound of the system is, before adding the FT technique, $\Theta = 1.481481$, and it is associated to the minimal p-semiflow of p_{32} – i.e., `WS-DBApplication`. When adding the FT technique described, the minimal p-semiflows that are modified correspond to the ones that use T_{31}, i.e., $\mathbf{y}_{p_0}, \mathbf{y}_{p_2}, \mathbf{y}_{p_{29}}$ and $\mathbf{y}_{p_{32}}$, and the upper throughput bound decreases near to a 133.98%, that is, $\Theta' = 0.633147$ and it is related as well to `WS-DBApplication`.

Let us apply now Algorithm 1 to compute the initial marking needed to compensate the throughput degradation. The minimal p-semiflows under study here are: $\mathbf{y}'_{p_0} = \mathbf{y}_{p_0} \cup \{^\bullet T_{31}, T^\bullet_{31}, p^1_4\}, \mathbf{y}'_{p_2} = \mathbf{y}_{p_2} \cup \{^\bullet T_{31}, T^\bullet_{31}, p^1_4\}, \mathbf{y}'_{p_{29}} = \mathbf{y}_{p_{29}} \cup \{^\bullet T_{31}, T^\bullet_{31}, p^1_4\}, \mathbf{y}'_{p_{31}} = \mathbf{y}_{p_{31}} \cup \{^\bullet T_{31}, T^\bullet_{31}, p^1_4\}$ (the other p-semiflows $\mathbf{y}''_{p_0}, \mathbf{y}''_{p_2}, \mathbf{y}''_{p_{29}}, \mathbf{y}''_{p_{31}}$ are not of interest due to $\delta_{detect} <= \delta_{31}$). The computation of value of $\mathbf{y}^1_{p_i} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is, respectively, $41.9520, 41.6557, 10.3965, 9.3594$. Thus, the solution of Algorithm 1 is $\mathbf{m'_0}(p_0) = 100, \mathbf{m'_0}(p_2) = 50, \mathbf{m'_0}(p_{29}) = 11, \mathbf{m'_0}(p_{31}) = 10$. That is, the number of `WS-Application` (p_{29}) and `WS-DBApplication` (p_{31}) must be incremented to 11 and 10 units, respectively, to maintain the given throughput of $\Theta = 1.481481$ and a probability of error of 0.15. If resources are incremented as it is given by the solution of this algorithm, the new upper throughput bound has a value of $\Theta' = 1.567476$.

Let us consider that the addition of new resources has some associated cost, more precisely, the cost of adding new instances of any host service is \$350 each (for instance, because new licenses for deploying more virtual servers must be purchased). In the case of recovery method, it can be improved having a cost, on average, of \$250 per each millisecond, and the minimum required time for recovering is $5ms$ (i.e., $\delta_{min}(T_{rec}) = 5ms$).

With this configuration, we apply now the proposal ILPP (10) for computing the minimal cost that compensate a probability of error of 0.15. The result of applying ILPP (10) is that 4 more resources of `WS-Application` (p_{29}), 5 more resources of `WS-DBApplication` (p_{32}) and recovery time must be decremented in $2ms$. The cost associated to these actions is \$3, 650. After applying these changes, the upper throughput bound is $\Theta'' = 1.500441$, which represents an improvement near to 1.28% of the previous upper throughput bound Θ.

Note that as the number of resources and the timing must be natural numbers, we will always obtain an upper throughput bound in the FT system where results of ILPP (10) are applied (slightly) better than in the original system model.

In summary, the solution of Algorithm (10) has an associated cost of \$3, 850, because 11 more resources must be added, whilst the solution giving by minimising cost through ILPP (10) costs \$3, 650.

6 Conclusions

Software systems are usually subject to faults that may lead to the existence of error and failures. Normally, Fault-Tolerant (FT) techniques are incorporated to these systems (then called FT systems) to mitigate the impact of activations of faults. FT systems can be naturally modelled as Discrete Event Systems (DES) where sharing resources are used.

In this paper, firstly we have provided compositional models for FT techniques that allow us to make performability (i.e., performance under failure conditions) analysis easier when FT parameters change. Thus, these FT models can be useful for evaluating different FT approaches in the same system model. Secondly, we have presented an iterative algorithm that computes the initial marking needed to maintain a given upper throughput bound in a system model within our proposed FT models. Thirdly, we present an Integer-Linear Programming Problem (ILPP) that minimises the cost of compensating throughput degradation caused by the presence of faults and errors). The use of linear programming techniques guarantees its efficiency and scalability to large models. Both algorithms are applied to a process Petri net modelling a Secure Database System.

This paper provides upper throughput bounds, which are usually closer to the real system throughput [24, 32]. As future work, we aim at analysing lower throughput bounds following the same methodology. The lower throughput bounds would enhance the throughput analysis under failure, as an interval for the throughput would be provided.

References

[1] Meyer JF. Closed-Form Solutions of Performability. IEEE Trans Comput. 1982 Jul;31(7):648–657. Available from: http://dx.doi.org/10.1109/TC.1982.1676062.

[2] Avizienis A, Laprie JC, Randell B, Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing. 2004 jan-march;1(1):11–33.

[3] Avizienis A. Toward Systematic Design of Fault-Tolerant Systems. Computer. 1997 apr;30(4):51–58.

[4] Colom J. The Resource Allocation Problem in Flexible Manufacturing Systems. In: van der Aalst W, Best E, editors. Applications and Theory of Petri Nets. vol. 2679 of LNCS. Springer Berlin / Heidelberg; 2003. p. 23–35.

[5] Tricas F. Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems. Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza; 2003. Available from: http://webdiis.unizar.es/~ftricas/Articulos/FernandoTricasFinal.pdf.gz.

[6] Goŝeva-Popstojanova K, Trivedi KS. Architecture-based approach to reliability assessment of software systems. Performance Evaluation. 2001;45(2–3):179–204. Available from: http://www.sciencedirect.com/science/article/pii/S0166531601000347.

[7] Gokhale SS, Wong WE, Horgan JR, Trivedi KS. An analytical approach to architecture-based software performance and reliability prediction. Performance Evaluation. 2004;58(4):391–412.

[8] Sanders WH, Meyer JF. A Unified Approach for Specifying Measures of Performance, Dependability, and Performability. Dependable Computing and Fault-Tolerant Systems: Dependable Computing for Critical Applications. 1991;4:215–237.

[9] Bobbio A. Petri Nets Generating Markov Reward Models for Performance/Reliability Analysis of Degradable Systems. In: Potier D, Puigjaner B, editors. Proceedings of the Fourth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation. New York, NY, USA: Plenum; 1989. p. 353–365.

[10] Raiteri DC, Franceschinis G, Iacono M, Vittorini V. Repairable Fault Tree for the automatic evaluation of repair policies. In: International Conference on Dependable Systems and Networks. IEEE; 2004. p. 659–668.

[11] Reussner RH, Schmidt HW, Poernomo IH. Reliability prediction for component-based software architectures. J Syst Softw. 2003 June;66(3):241–252. Available from: `http://dx.doi.org/10.1016/S0164-1212(02)00080-8`.

[12] Abdelmoez W, Nassar DM, Shereshevsky M, Gradetsky N, Gunnalan R, Ammar HH, et al. Error Propagation in Software Architectures. In: Proceedings of the 10th International Symposium on Software Metrics (ISSME); 2004. p. 384–393.

[13] Cortellessa V, Grassi V. A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems. In: Schmidt H, Crnkovic I, Heineman G, Stafford J, editors. Proceedings of the 10th International Conference on Component-Based Software Engineering. vol. 4608 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin / Heidelberg; 2007. p. 140–156. Available from: `http://dl.acm.org/citation.cfm?id=1770657.1770670`.

[14] Li J, Fan Y, Zhou M. Performance Modeling and Analysis of Workflow. IEEE T Syst Man Cy A. 2004 March;34(2):229–242.

[15] Wang H, Zeng Q. Modeling and Analysis for Workflow Constrained by Resources and Nondetermined Time: An Approach Based on Petri Nets. IEEE T Syst Man Cy A. 2008 July;38(4):802–817.

[16] Hee KV, Reijers H, Verbeek E, Zerguini L. On the Optimal Allocation of Resources In Stochastic Workflow Nets. In: Djemame K, Kara M, editors. Proceedings of the 7th UK Performance Engineering Workshop. University of Leeds, Leeds, UK; 2001. p. 23–34.

[17] Chen YL, Hsu PY, Chang YB. A Petri Net Approach to Support Resource Assignment in Project Management. IEEE T Syst Man Cy A. 2008 May;38(3):564–574.

[18] Murata T. Petri Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE. vol. 77; 1989. p. 541–580.

[19] Ramchandani C. Analysis of Asynchronous Concurrent Systems by Petri Nets. Dept. of Electrical Engineering, Massachusetts Institute of Technology. Cambridge, MA, USA; 1974.

[20] Florin G, Natkin S. Les réseaux de Petri stochastiques. Technique et Science Informatique. 1985;4:143–160.

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

[22] Ross SM. Stochastic Processes. Wiley series in mathematical statistics. Probability and mathematical statistics. Wiley; 1983. Available from: `http://books.google.es/books?id=Hj7bAAAAMAAJ`.

[23] Florin G, Natkin S. Necessary and Sufficient Ergodicity Condition for Open Synchronized Queueing Networks. IEEE T Software Eng. 1989;15(4):367–380.

[24] Campos J, Silva M. Structural Techniques and Performance Bounds of Stochastic Petri Net Models. Lecture Notes in Computer Science. 1992;609:352–391.

[25] Donatelli S, Franceschinis G. The PSR Methodology: Integrating Hardware and Software Models. In: Proceedings of the 17th International Conference of Application and Theory of Petri Nets (ICATPN); 1996. p. 133–152.

[26] Bernardi S, Donatelli S, Horváth A. Implementing Compositionality for Stochastic Petri Nets. Journal of Software Tools for Technology Transfer. 2001;3:417–430.

[27] Sousa P, Bessani AN, Correia M, Neves NF, Verissimo P. Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery. IEEE Transactions on Parallel and Distributed Systems. 2010 april;21(4):452–465.

[28] Distefano S, Scarpa M, Puliafito A. From UML to Petri Nets: The PCM-Based Methodology. IEEE Transactions on Software Engineering. 2011 jan-feb;37(1):65–79.

[29] Gómez-Martínez E, Merseguer J. ArgoSPE: Model-Based Software Performance Engineering. In: International Conference of Application and Theory of Petri Nets; 2006. p. 401–410.

[30] Rodríguez RJ, Júlvez J, Merseguer J. 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. p. 142–147. Available from: http://webdiis.unizar.es/~ricardo/files/papers/RJM-ACSD-12.pdf.

[31] Baarir S, Beccuti M, Cerotti D, De Pierro M, Donatelli S, Franceschinis G. The GreatSPN tool: recent enhancements. SIGMETRICS Perform Eval Rev. 2009;36(4):4–9.

[32] Rodríguez RJ, Júlvez J. Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing. In: Proceedings of the 7th European Performance Engineering Workshop (EPEW). vol. 6342 of LNCS. Springer; 2010. p. 175–190. Available from: http://webdiis.unizar.es/~ricardo/files/papers/RJ-EPEW-10.pdf.

List of Figures

List of Figures

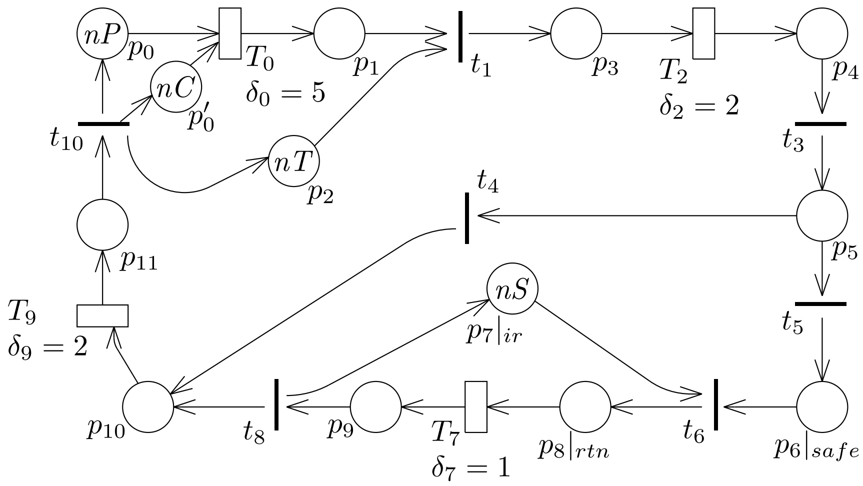


Figure 1: Petri net representation of a packet-routing algorithm.

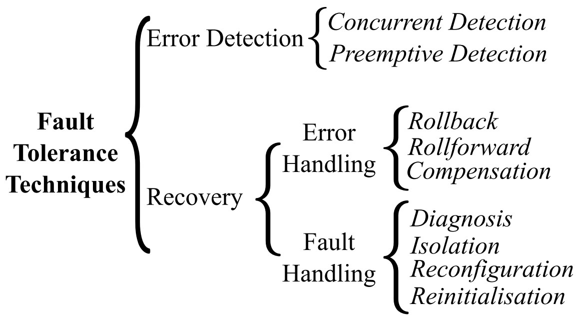
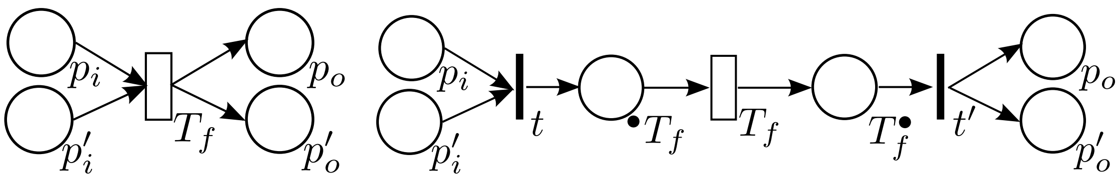


Figure 2: Phases involved on a Fault-Tolerant technique (adapted from [2]).



(a) Original model (b) Transformed model

Figure 3: Transformation rule $\mathcal{TR}$ of a transition t_f subject to fail (faulty transition).

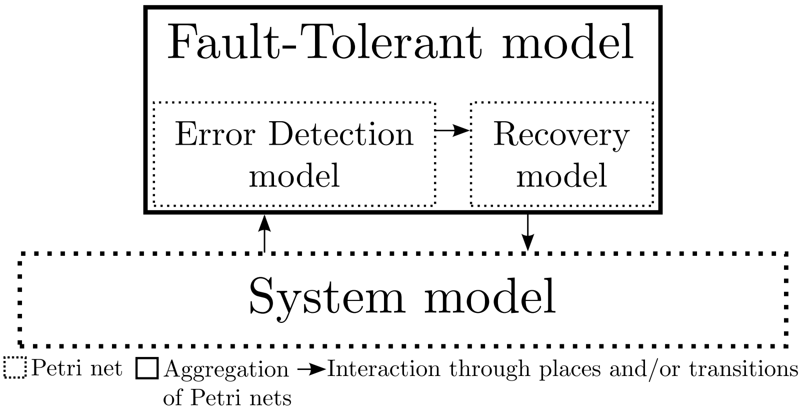
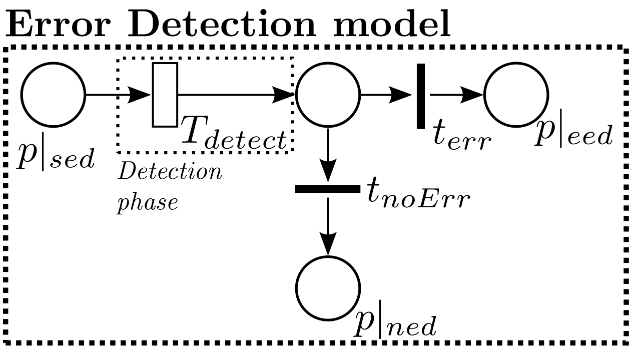
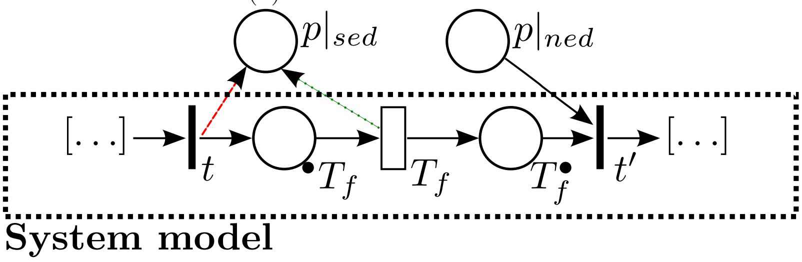


Figure 4: Integration between a PN-based system model and a PN-based FT technique.

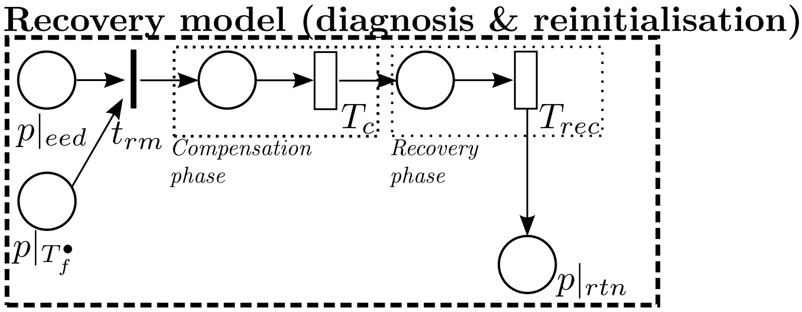


(a) Error Detection model

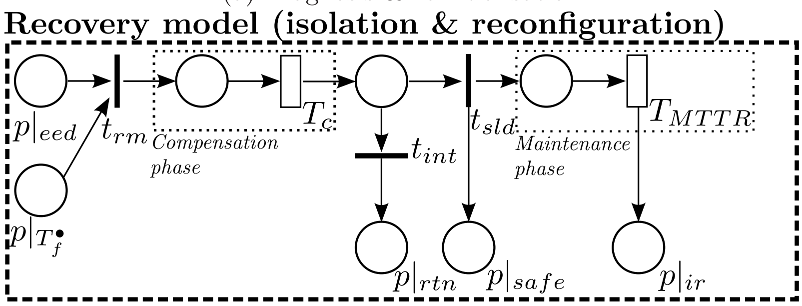


(b) Places $p|_{sed}, p|_{ned}$ added to the system model

Figure 5: PN-based model of Error Detection and faulty activity inside the system.



(a) Diagnosis & reinitialisation



(b) Isolation & reconfiguration

Figure 6: PN-based models of Recovery model: (a) and (b) isolation & reconfiguration.

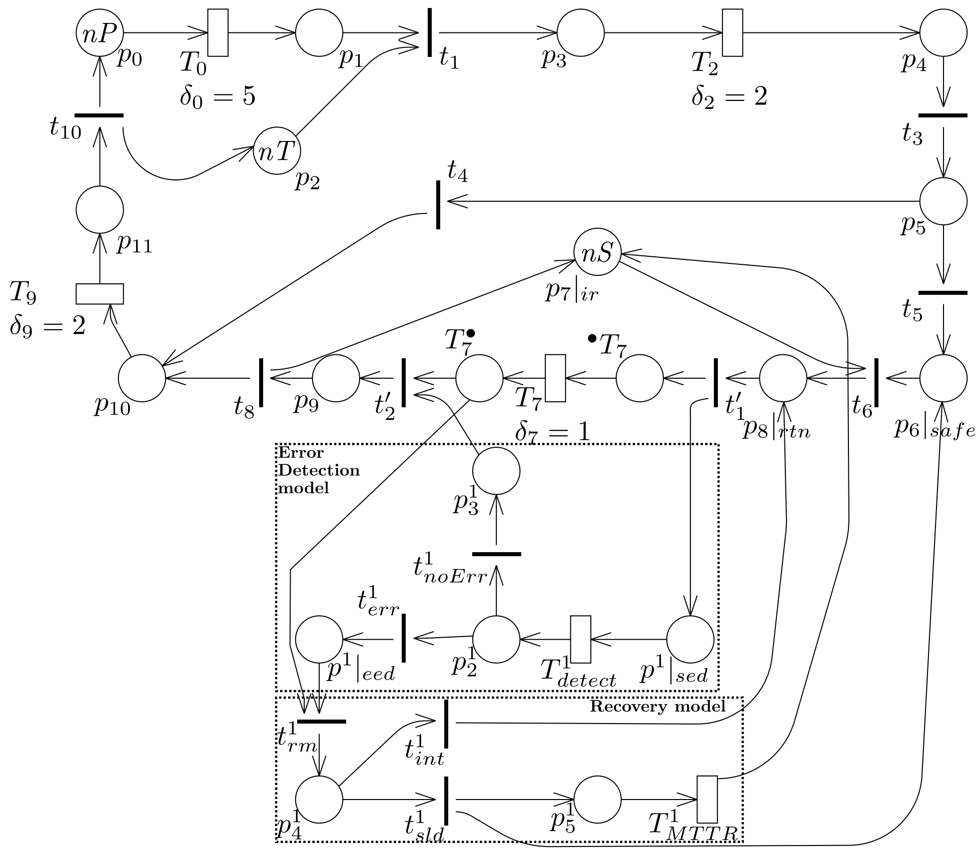
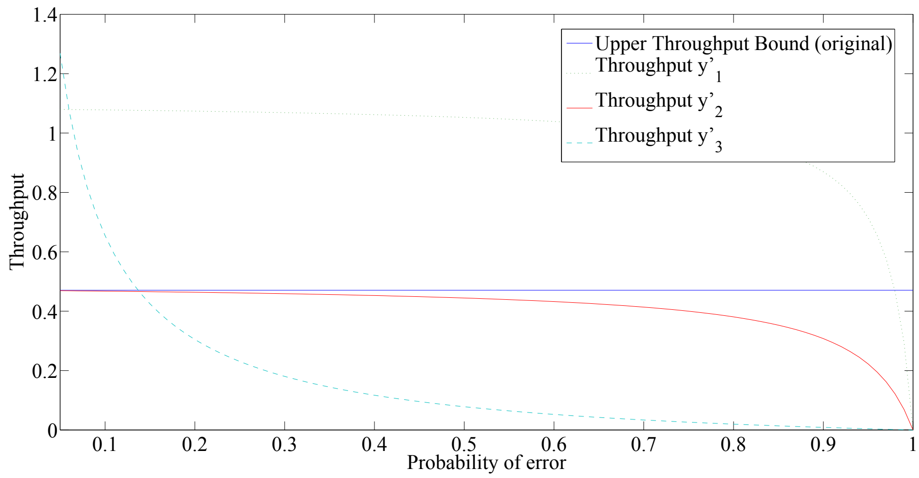
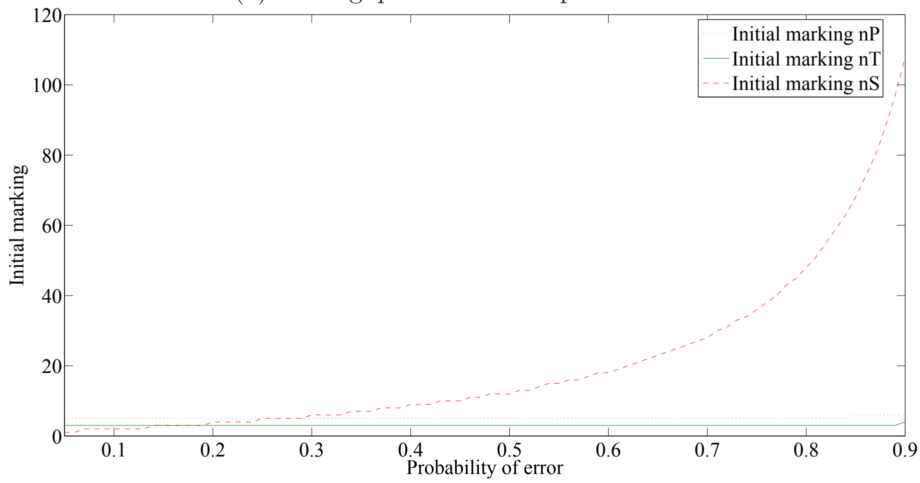


Figure 7: Petri net representation of the packet-routing algorithm depicted in Figure 1 extended with a FT technique.



(a) Throughput of minimal p-semiflows



(b) Initial marking of resource places

Figure 8: Results of (a) throughput values and (b) initial marking with respect to probability of error.

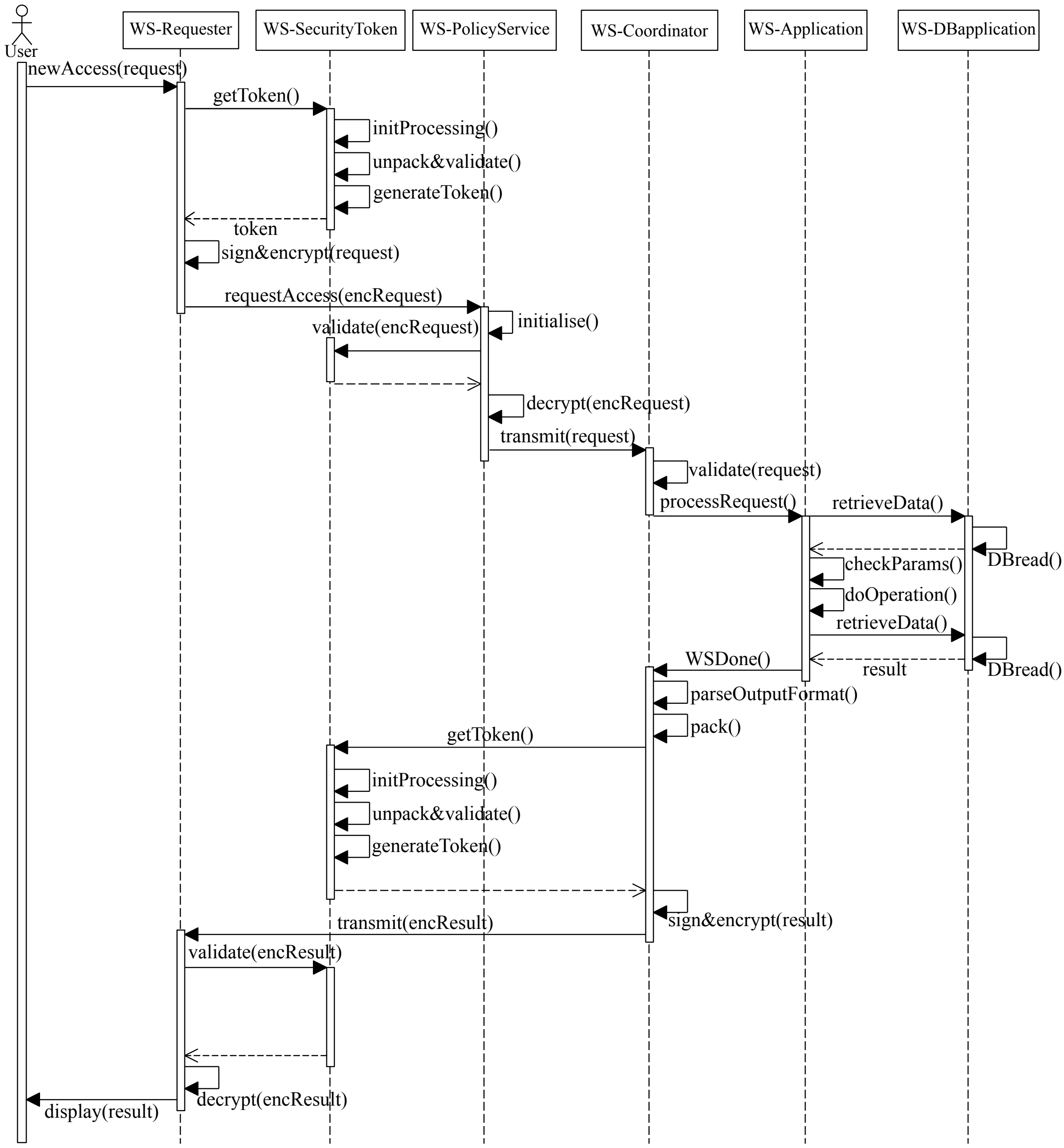


Figure 9: SDBS Request Customer's Data scenario.

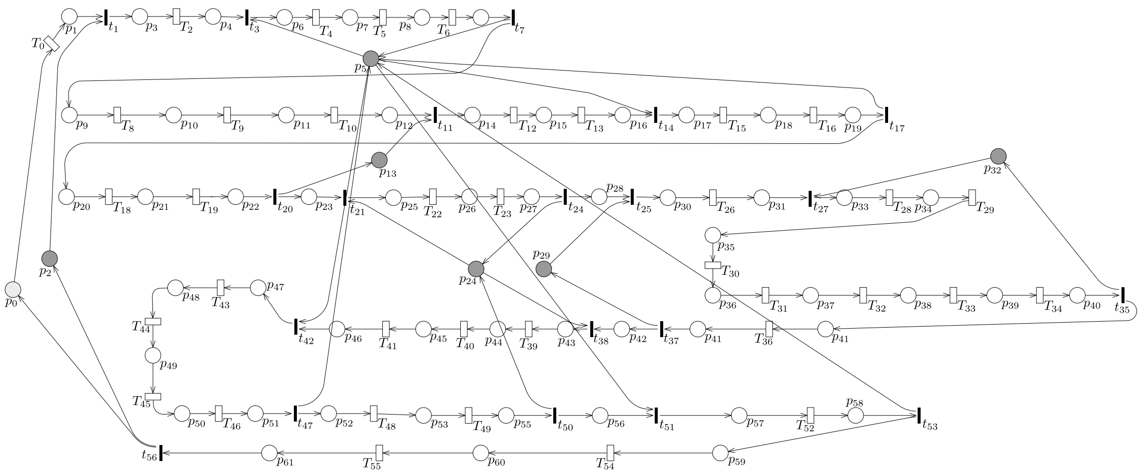


Figure 10: Petri net of the SDBS. Resource places are depicted in dark grey, whilst process-idle place in light grey.

List of Tables

	Rollforward (& compensation)*	**Rollbackward (& compensation)***
Diagnosis	$\surd$	$\surd$
Isolation	$\surd$	$\surd$
Reconfiguration	X	$\surd$
Reinitialisation	X	$\surd$

Table 1: Valid combinations of error handling and fault handling techniques. The symbol * means optional.

$$\begin{array}{l}
\mathbf{y}_1' = \mathbf{y}_1 \cup \{{}^{\bullet}T_7, T_7^{\bullet}, p_4^1\} \\
\mathbf{y}_1'' = \mathbf{y}_1 \cup \{p^1|_{sed}, p_2^1, p_3^1, p^1|_{eed}, p_4^1\} \\
\mathbf{y}_2' = \mathbf{y}_2 \cup \{{}^{\bullet}T_7, T_7^{\bullet}, p_4^1\} \\
\mathbf{y}_2'' = \mathbf{y}_2 \cup \{p^1|_{sed}, p_2^1, p_3^1, p^1|_{eed}, p_4^1\} \\
\mathbf{y}_3' = \mathbf{y}_3 \cup \{{}^{\bullet}T_7, T_7^{\bullet}, p_4^1, p_5^1\} \\
\mathbf{y}_3'' = \mathbf{y}_3 \cup \{p^1|_{sed}, p_2^1, p_3^1, p^1|_{eed}, p_4^1, p_5^1\}
\end{array}$$

Table 2: New p-semiflows of the PN in Figure 7.

Transition	**Method**	**Value(s)**
T_0	`newAccess()`	$0.2ms$
T_2, T_8, T_{10}, T_{49}	*$delayNet*	$2.5ms$
$T_{13}, T_{16}, T_{19}, T_{23}, T_{36}, T_{41}, T_{46}$	*$intranetLag*	$0.2ms$
$T_{26}, T_{29}, T_{32}, T_{34}$	*$secIntraLag*	$0.5ms$
T_4, T_{43}	`initProcessing()`	$1ms$
T_5, T_{44}	`unpack&validate()`	$0.1ms$
T_6, T_{45}	`generateToken()`	$0.5ms$
T_9, T_{48}	`sign&encrypt()`	$0.8ms$
T_{12}	`initialise()`	$0.3ms$
T_{15}, T_{22}, T_{52}	`validate()`	$0.3ms$
T_{18}, T_{54}	`decrypt()`	$1ms$
T_{28}, T_{33}	`DBread()`	$0.2ms$
T_{30}	`checkParams()`	$0.6ms$
T_{31}	`doOperation()`	$0.2ms$
T_{39}	`parseOutputFormat()`	$0.3ms$
T_{40}	`pack()`	$0.1ms$
T_{55}	`display()`	$1.5ms$

(a) Activity times

Place	**Meaning**	**Value(s)**
p_0	No. users	100
p_2	No. request capacity	50
p_5	No. security hosts	25
p_{13}	No. policy hosts	10
p_{24}	No. coordinator hosts	10
p_{29}	No. application hosts	6
p_{32}	No. DB hosts	4

(b) Initial number (no.) of resources

Table 3: Experiments parameters.