Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Net

Sachin Garg
Center for Advanced Comp. & Comm.
Dept. of Electrical Engineering
Duke University
Durham, NC 27708

Antonio Puliafito
Ist. di Informatica e Telecom.
Università di Catania, Viale A. Doria, 6
95125 Catania, Italy

Miklós Telek
Dept. of Telecommunications
Technical University of Budapest
1521 Budapest, Hungary

Kishor S. Trivedi
Center for Advanced Comp. & Comm.
Dept. of Electrical Engineering
Duke University
Durham, NC 27708

Abstract

In a client-server type system, the server software is required to run continuously for very long periods. Due to repeated and potentially faulty usage by many clients, such software "ages" with time and eventually fails. Huang et. al. proposed a technique called "software rejuvenation" [9] in which the software is periodically stopped and then restarted in a "robust" state after proper maintenance. This "renewal" of software prevents (or at least postpones) the crash failure. As the time lost (or the cost incurred) due to the software failure is typically more than the time lost (or the cost incurred) due to rejuvenation, the technique reduces the expected unavailability of the software. In this paper, we present a quantitative analysis of software rejuvenation. The behavior of the system is represented through a Markov Regenerative Stochastic Petri Net (MRSPN) model which is solved both for steady state as well as transient conditions. We provide a closed-form analytical solution for the steady state expected down time (and the expected cost incurred) due to system unavailability. We also evaluate the optimal rejuvenation interval which minimizes the expected unavailability of the software.

1 Introduction

In fault tolerant systems, preventive maintenance is considered as one of the key strategies to increase system availability and to reduce costs due to the system failure. It is a widely researched field, especially in the operations research community. The reader is referred to [17] for a survey. In general, preventive maintenance consists of periodically stopping the system and restarting it after doing proper maintenance. This reduces the probability of "unexpected" failure of the system, which would have eventually happened without any maintenance. Since the system is unavailable for normal use during maintenance, some cost is involved in doing so. A typical research problem is to find the optimal maintenance policy, i.e., the one which minimizes a certain cost function defined on the system unavailability. While preventive maintenance concepts have been usually applied to mechanical systems, they can also be effectively applied to the field of software reliability.

With constant and rapid reduction in hardware failure rates due to fast-paced technological improvements, importance of software reliability in overall systems' availability is being highlighted. System failures due to imperfect software behavior are usually more frequent than failures caused by hardware components' faults [2]. These failures result from either inherent design defects in the software or from improper usage by clients [14]. Thus fault tolerant software has become an effective alternative to virtually impossible fault-free software. A wide literature exists in this field where the software has the ability to recover from a *transient* fault [1, 10, 11, 16]. Most of the approaches, for example, the N-version programming [1] approach and the recovery block [16] approach are corrective in nature, i.e. only after a failure occurs, the recovery process is started. The overhead incurred by such recovery strategies remains high and much research has gone into reducing it.

Huang et. al. have suggested a complimentary technique which is preventive in nature. It involves periodic maintenance of the software so as to prevent unexpected failures. They call it *software rejuvenation* [9] and define it as the *periodic preemptive rollback of continuously running applications to prevent failures.*

While monitoring real applications, it was observed that software typically "ages" as it is run. Potential fault conditions slowly accumulate since the beginning of the software activity. Consider, for example, a

server module interacting with many client modules. Memory bloating, unreleased file-locks, data corruption are the typical causes of slow degradation which, if not taken care of, lead to crash failure. Software rejuvenation involves periodically stopping the system, cleaning up, and restarting it from a clean internal state. This "renewal" of software prevents (or in the least postpones) a crash failure. Since the down time caused by this planned shutdown is typically less than the down time resulting from a crash failure, this strategy increases the system availability. For further motivation and practical examples, the reader is referred to [9].

In this paper, we present a quantitative analysis of software rejuvenation. To deal with deterministic interval between successive rejuvenations, behavior of the system is represented through a Markov regenerative stochastic Petri net (MRSPN) model which is subsequently solved for steady state as well as transient conditions using Markov renewal theory. We provide a closed-form analytical solution for the steady state expected down time (and the expected cost incurred due to software unavailability). Earlier work on quantitative analysis by Huang et. al. was based on a continuous time Markov chain (CTMC) model. Intuitively, we expect that there would be a trade-off involved between the down time caused due to crash failures and down time due to rejuvenation depending on how often it is performed. We demonstrate the effect of the rejuvenation interval defined as the *time to perform next rejuvenation starting in the robust state* on the steady state expected down time and cost. We also evaluate the optimal value of this interval which minimizes the software unavailability for a given set of system parameters.

The rest of this paper is organized as follows. In Section 2, we give a brief introduction to the theory of MRSPNs, their evaluation technique and the fundamental equations for the steady state and the transient probabilities. Description of the system, assumptions and the MRSPN model that captures the system behavior is given in Section 3. Reachability graph of the MRSPN model is also constructed in this section. In Section 4, we derive the matrices which describe the system mathematically. Next, we solve for the transient and the steady state expected down time and expected cost incurred due to unavailability of the software using equations from Section 2. Section 5 contains an illustrative numerical example and interpretation of the results. Finally, in Section 6, we conclude with pointers to further research.

2 Introduction to MRSPN

One difficulty in modeling a stochastic system such as software with rejuvenation arises because of the deterministic rejuvenation interval, which renders the system *"non-Markovian"* and standard modeling method using the theory of continuous time Markov chains can not be applied. In this case, the approach is to study the underlying stochastic process of such non-Markovian systems. Although a general stochastic process may not be analytically tractable, in many cases this process can be shown to be a Markov regenerative one (MRGP also known as semi-regenerative process) and therefore Markov renewal theory can be applied for its long-run as well as transient behavior [6, 3, 4].

A complementary issue is to specify the system behavior in a concise way from which the underlying stochastic process can be extracted and analyzed. Petri nets with their remarkable flexibility and potential for capturing concurrency, contention and synchronization in a system have been widely used for qualitative modeling [15]. To study a system quantitatively, stochastic Petri nets (SPNs) can be used as the high-level specification tool. Each transition in an SPN can be one of the following three types[1].

- *Type I:* Immediate (i.e. they fire in zero time)
- *Type II:* Timed with exponentially distributed firing time
- *Type III:* Timed with generally distributed firing time

If the SPN contains only type I and type II transitions, the system is Markovian, i.e., at any instant, the future evolution depends only on the current state and not on the past history. It is then standard to automatically generate the underlying continuous time Markov chain [5, 19] and numerically solve it for reliability and performance measures.

If however, the SPN model contains at least one type III transition, the above mentioned memoryless property does not hold in general. For analyzing such a non-Markovian SPN, we need to identify certain time points embedded in the underlying stochastic process at which it is possible to forget the past history. These points, indicated as *regeneration points*, are such that the future evolution of the stochastic process only depends on the present state entered when a regeneration time point occurs. The underlying stochastic process is determined by a marking process $\{\mathcal{M}(t), t > 0\}$, obtained by constructing the reachability graph for the net. Once the *Reachability Set* ($\mathcal{RS}$) of the net is identified, namely the set of all possible states (markings) of the system, the *reachability graph* ($\mathcal{RG}$) can be obtained by connecting a marking M_i to a marking M_j with a directed arc if the marking M_j can result from the firing of some transition enabled in M_i. From the given initial marking M_0, a unique reachability graph is obtained. A marking is a *tangible* marking if no Type I (immediate) transition is enabled in that marking, otherwise it is a *vanishing* marking.

A single realization of the marking process $\mathcal{M}(t)$ can be written as:

$$\mathcal{R} = \{(\tau_0, M_0); (\tau_1, M_1); \ldots; (\tau_i, M_i); \ldots\}$$

[1] The classification is valid only when the transition follows the so called prd (preemptive repeat different) policy. Since the discussion and analysis of different firing policies [18] is orthogonal to this paper, we do not elaborate on this aspect.

where M_{i+1} is a marking immediately reachable from M_i, and $\tau_{i+1} - \tau_i$ is the sojourn time in marking M_i. With the above notation, $\mathcal{M}(t) = M_i$ for $\tau_i \leq t < \tau_{i+1}$. We now give a formal definition of MRSPN.

Definition 1 - *A regeneration time point τ_n^* in the marking process $\mathcal{M}(t)$ is the epoch of entrance in a tangible marking M_n in which the Markov property holds.*

Definition 2 - *An SPN, for which an embedded sequence of regeneration time points and associated state (τ_n^*, M_n) behaving as a Markov renewal process (or Markov renewal sequence) can be found, is an MRSPN [3].*

Choi et. al. in [3] showed that if at any time, at most one type *III* (generally distributed) transition is enabled, then it is always possible to find an embedded sequence (τ_n^*, M_n) i.e. the non-Markovian SPN is guaranteed to belong to the class of MRSPN.

Let Ω represent the state space of the underlying MRGP. It is given by the tangible subset of the reachability graph given an initial marking. Thus, $\Omega = \mathcal{RS}(M_0)$. also let n be the cardinality of Ω. Let the set of possible states at regeneration time points be given by Ω'. Thus $\Omega' = \{M_n : (\tau_n^*, M_n)$ is the embedded sequence $\}$. Clearly, $\Omega' \subseteq \Omega$ and $m = |\Omega'| \leq n$. To provide an analytical formulation of the stochastic process underlying an MRSPN, according to [3, 6], we define $\mathbf{V}(t) = [V_{ij}(t)]$, $\mathbf{K}(t) = [K_{ij}(t)]$ and $\mathbf{E}(t) = [E_{ij}(t)]$ as the following matrix valued functions.

$$\begin{aligned} V_{ij}(t) &= Pr\{\mathcal{M}(t) = j \,|\, \mathcal{M}(\tau_0^*) = i\} \\ K_{ij}(t) &= Pr\{\mathcal{M}(\tau_1^*) = j\ ,\ \tau_1^* \leq t | \,\mathcal{M}(\tau_0^*) = i\} \\ E_{ij}(t) &= Pr\{\mathcal{M}(t) = j\ ,\ \tau_1^* > t | \,\mathcal{M}(\tau_0^*) = i\} \end{aligned} \quad (1)$$

$\mathbf{V}(t)$ is an $m \times n$ transition probability matrix and gives the probability that the stochastic process $\mathcal{M}(t)$ is in marking j at time t given that it was in marking i at $t = 0$. Thus $\mathbf{V}(t)$ captures the transient behavior of the process. The $m \times m$ matrix $\mathbf{K}(t)$ is called the *global kernel* and provides the probability of the event that the next regeneration time point is τ_1^* and the next regeneration marking is j given that the marking is i at $\tau_0^* = 0$. Finally, the $m \times n$ matrix $\mathbf{E}(t)$ is called the *local kernel* since it describes the behavior of the marking process $\mathcal{M}(t)$ inside two consecutive regeneration time points. The element $E_{ij}(t)$ is the probability that the process is in marking j at time t starting from marking i at $\tau_0^* = 0$ before the next regeneration time point. From the above definitions:

$$\sum_{j \in \Omega'} K_{ij}(t) + \sum_{j \in \Omega} E_{ij}(t) = 1 \quad (2)$$

The transient behavior of the *MRSPN* can be evaluated by solving the following generalized Markov renewal equation (in matrix form) [6, 3]:

$$\mathbf{V}(t) = \mathbf{E}(t) + \mathbf{K} * \mathbf{V}(t) \quad (3)$$

where $\mathbf{K} * \mathbf{V}(t)$ is a convolution matrix, whose (i, j)-th entry is:

$$[\mathbf{K} * \mathbf{V}(t)]_{ij} = \sum_k \int_0^t dK_{ik}(y)\, V_{kj}(t - y).$$

Next, we outline the solution of the above equation for steady state and transient cases.

2.1 Steady-state solution

If the embedded discrete time Markov chain (DTMC) defined at regeneration points $(\{\mathcal{M}(\tau_n^*), n \geq 0\})$ is finite and irreducible then its steady-state probability vector ν given by the solution of the linear system

$$\nu = \nu \mathbf{K}(\infty) \quad (4)$$

under the condition $\sum_{i \in \Omega'} \nu_i = 1$ can be evaluated. Let α_{ij} denote the integral $\int_0^\infty E_{ij}(t)dt$. Then it can be shown [13] that the steady-state probabilities π_j of the MRGP can be obtained in closed form by:

$$\pi_j = \frac{\sum_{k \in \Omega} \nu_k \alpha_{kj}}{\sum_{k \in \Omega} \nu_k \sum_{l \in \Omega} \alpha_{kl}} \quad (5)$$

2.2 Transient solution

Coupled integral Equations (3) describing the behavior of an MRGP[2] can be numerically solved by two different approaches:

- Direct solution in time domain. Equation (3) represents coupled Voltera equation of the second kind, for which the numerical solution methods are discussed in [7].
- Numerical solution from transform domain. In this paper, we follow this approach as described below.

If we take the Laplace-Stieljes transform (LST) on both sides of (3) we obtain,

$$\tilde{\mathbf{V}}(s) = \tilde{\mathbf{E}}(s) + \tilde{\mathbf{K}}(s)\tilde{\mathbf{V}}(s),$$

from which the transient probabilities in Laplace transform (LT) domain (as LST of a function is s times its LT) are obtained as

$$\overset{*}{\mathbf{V}}(s) = \left[\mathbf{I} - s\overset{*}{\mathbf{K}}(s)\right]^{-1} \overset{*}{\mathbf{E}}(s) \quad (6)$$

[2] An alternative formulation for transient solution of MRGPs is possible using partial differential equations [8].

Symbolic manipulators like "Mathematica" can be used to automate evaluation of matrix inversion and obtain expressions for V_{ij} in the s domain. These expressions are then inverted numerically to obtain the solution in time domain. For this purpose we use the Jagerman's method [12].

To summarize the procedure, modeling with MRSPNs consists of the following steps.

1. Specify the system behavior by a concise SPN and verify that it falls in the MRSPN class.
2. Obtain the reachability graph of the SPN and determine the state space of the underlying MRGP.
3. Derive the global kernel ($\mathbf{K}$) and the local kernel ($\mathbf{E}$) from the reachability graph in time domain as well as Laplace domain.
4. Use Equation (6) and numerical inversion to obtain the transient measures
5. Calculate $\mathbf{K}(\infty)$ and solve for the steady state probabilities of the embedded DTMC.
6. Evaluate α_{ij}s from the local kernel and use Equation (5) to obtain the steady state measures.

We now proceed to follow the above steps for analyzing software rejuvenation.

3 The system

The software starts up in a "robust" state in which the probability of failure is zero. As it is used, it ages with time and if no rejuvenation is done eventually transits to another state. In this state, it provides normal service but can fail (crash) with a non-zero probability. Once it crashes, it takes a random amount of time to bring it up again to the clean state and restart it. Rejuvenation is performed at a fixed interval from the start (or restart) of the software in the robust state. At the time of rejuvenation, if the software has not already crashed, it is either in the clean or the failure probable state. It is then stopped, cleaned and restarted; all of which takes a random amount of time. We assume that the time for which software remains clean and the time to fail from the failure probable state are both exponentially distributed. Thus the time to failure for the software starting in the robust state has a hypo-exponential distribution. We further assume that the times to restart both from rejuvenation and crash failures are both exponentially distributed. The rejuvenation interval, however, is deterministic. Even though our assumption about exponentiality of restart times is not substantiated, they clearly suffice to demonstrate the tradeoffs involved in rejuvenation. Showing the existance of an optimality condition and illustrating the trade-offs is the primary objective of this paper. Furthermore, it is relatively straightforward to solve the same problem with general distributions using the same model.

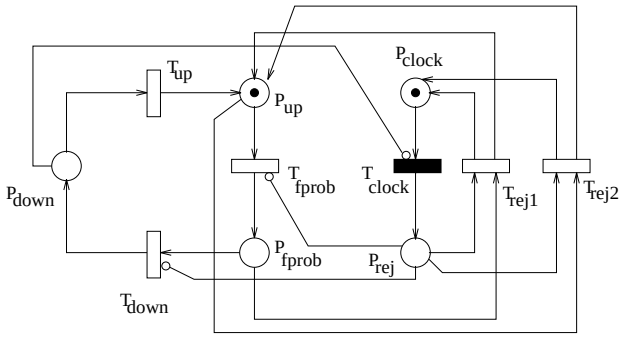

Figure 1: MRSPN Model of Software Rejuvenation

3.1 Petri net model

Figure 1 shows the Petri net model of the above system. The circles represent places with dots inside representing the tokens held inside that place. Unshades rectangles represent transitions with exponentially distributed firing time while the shaded rectangle represents a transition with a constant firing time.

The robust state is modeled by the place P_{up}. Transition T_{fprob} models the aging of the software. When this transition fires, i.e., a token reaches place P_{fprob}) the software enters the failure probable state. The transition T_{down} models crash failure of the software. During the software restart (while the transition T_{up} is enabled), every other activity is suspended; the inhibitor arc from place P_{down} to transition T_{clock} is used to model this fact.

The transition T_{clock} models the rejuvenation period. It is competitively enabled with T_{fprob} and fires when the clock expires if T_{fprob} has not fired by that time. Once it fires, a token moves in place P_{rej} and the activity related with software rejuvenation (transition T_{rej}) starts.

During the rejuvenation phase, every other activity in the system is suspended. This is modeled by inhibitor arcs from place P_{rej} to transitions T_{fprob} and T_{down}. Upon rejuvenation, the net has to be re-initialized into a condition with one token in place P_{up} and one in place P_{clock}, and all the other places empty. If the software was in the robust state when T_{clock} fired, then after rejuvenation is complete, T_{rej2} fires to re-initialize the net. If the software had reached the failure probable state (token in place P_{fprob}), then T_{rej1} fires to complete the rejuvenation and re-initializes the net.

As there is only one deterministic transition in the net, the condition for at most one generally distributed transition enabled at any time is automatically satisfied. Thus our SPN model of the system belongs to the MRSPN class.

3.2 Reachability graph

Since there are no immediate transitions in the net, all the markings are tangible. Let the 5-

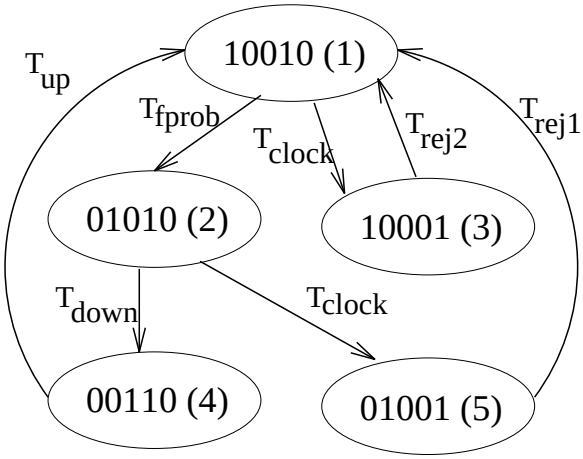

Figure 2: Reachability Graph for the MRSPN Model

tuple $(P_{up}, P_{fprob}, P_{down}, P_{clock}, P_{rej})$ denote a marking with $P_x = 1$, if a token is present in place P_x, and zero otherwise. From the SPN description, it is clear that only five markings are possible viz (10010), (01010), (10001), (00110) and (01001).

Figure 2 shows the reachability graph with ovals representing the markings and arcs representing possible transitions between the markings. The five markings mentioned above are labeled one through five respectively. An arc from a marking i to another marking j is labeled with the name of the transition whose firing brought about the change. Let λ_1, λ_2, λ_3, λ_4 and λ_5 be the transition rates associated with T_{fprob}, T_{down}, T_{rej1}, T_{up} and T_{rej2} respectively. Also, let δ be the firing time associated with T_{clock}.

4 Model solution

The state space of the underlying MRGP consists of five states, i.e., $\Omega = \{1, 2, 3, 4, 5\}$. We define the regeneration time points as the instances when either of the following events occur:

- The deterministic transition gets enabled.
- The deterministic transition gets disabled or fires.
- An exponentially distributed transition which is not competitively enabled with the deterministic transition fires.

It can be seen from the net behavior and above conditions that the regeneration times exactly correspond to times of entering either of states 1, 3, 4 or 5. In all these states, at the time of entering that state, future evolution of the process can be fully characterized by just that state. In contrast, when a process transits from state 1 to state 2 due to firing of T_{fprob}, the deterministic transition T_{clock} that was enabled when process entered state 1, remains enabled. Thus, in order to probabilistically determine future state of the process, the time already expired in T_{clock} also needs to be known apart from the knowledge that the process is in state 2.

Thus possible set of states at regeneration instants is given by $\Omega' = \{1, 3, 4, 5\}$. We now proceed to define the $\mathbf{E}(t)$ and the $\mathbf{K}(t)$ matrices. Since cardinality of Ω' is four, $\mathbf{K}(t)$ is a 4×4 matrix given as following:

$$\mathbf{K}(t) = \begin{pmatrix} 0 & K_{13}(t) & K_{14}(t) & K_{15}(t) \\ K_{31}(t) & 0 & 0 & 0 \\ K_{41}(t) & 0 & 0 & 0 \\ K_{51}(t) & 0 & 0 & 0 \end{pmatrix}$$

Note that the subscripts ij on $K_{ij}(t)$ denote the actual state labels according to the reachability graph (and not the indices of rows or columns). The diagonal entries are zero as at a regeneration instance, the process must change state. As no transition is possible from states 3, 4 and 5 to each other, corresponding entries are zero.

From Equation (1), $K_{13}(t)$ is given by the probability that the process enters state 3 by firing of T_{clock}. This equals the probability that T_{fprob} does not fire in the interval $[0, \delta)$ and is given as

$$K_{13}(t) = e^{-\lambda_1 \delta} u(t - \delta) \tag{7}$$

where $u(t)$ denotes the unit-step function. $K_{14}(t)$ is given by the joint probability that the state at next regeneration instant is 4 and the time to regenerate is less than or equal to t, given that the state is 1 at time zero. As the process has to pass through state 2, this quantity is obtained by conditioning on the time to reach 2 and then un-conditioning in the interval $[0, \delta]$ as

$$K_{14}(t) = \begin{cases} 1 - \frac{\lambda_1}{\lambda_1 - \lambda_2} e^{-\lambda_2 t} + \frac{\lambda_2}{\lambda_1 - \lambda_2} e^{-\lambda_1 t}, & 0 \le t < \delta \\ 1 - \frac{\lambda_1}{\lambda_1 - \lambda_2} e^{-\lambda_2 \delta} + \frac{\lambda_2}{\lambda_1 - \lambda_2} e^{-\lambda_1 \delta}, & t \ge \delta \end{cases} \tag{8}$$

$K_{15}(t)$ is given by the probability that T_{down} does not fire up to time δ, given that the process is in state 1 at time zero.

$$K_{15}(t) = \frac{\lambda_1}{\lambda_1 - \lambda_2} \left[e^{-\lambda_2 \delta} - e^{-\lambda_1 \delta} \right] u(t - \delta) \tag{9}$$

In states 3, 4, and 5, only a single exponential transition is enabled. So $K_{i1}(t), i = 3, 4, 5$ is the probability that the corresponding transition fires in time t.

$$K_{i1}(t) = 1 - e^{-\lambda_i t}, i = 3, 4, 5 \tag{10}$$

$\mathbf{E}(t)$ describes the process at times between two consecutive regeneration points starting from a state at regeneration and hence is a 4×5 matrix given as following:

$$\mathbf{E}(t) = \begin{pmatrix} E_{11}(t) & E_{12}(t) & 0 & 0 & 0 \\ 0 & 0 & E_{33}(t) & 0 & 0 \\ 0 & 0 & 0 & E_{44}(t) & 0 \\ 0 & 0 & 0 & 0 & E_{55}(t) \end{pmatrix}$$

Again, the subscripts ij represent the actual state labels and not the row or column indices. Zero entries in the matrix at ijth location indicate that either a transition from i to j is not possible or it results in a regeneration. $E_{11}(t)$ is simply the probability that the process starting in state 1, stays in it till time t. Here t lies in the interval $[0, \delta)$ as at time δ, the process is sure to transit to state 3 resulting in a regeneration. Thus

$$E_{11}(t) = e^{-\lambda_1 t}\left(u(t) - u(t-\delta)\right) \tag{11}$$

$E_{12}(t)$ is the probability that at time t the process is in state 2 given that at time 0, it was in state 1. Note that t takes values in $(0, \delta)$. Conditioning on time to enter state 2 and un-conditioning, we obtain

$$E_{12}(t) = \frac{\lambda_1}{\lambda_1 - \lambda_2}\left[e^{-\lambda_2 t} - e^{-\lambda_1 t}\right]\left(u(t) - u(t-\delta)\right) \tag{12}$$

$E_{ii}(t)$, for $i = 3, 4, 5$ is the probability that the corresponding enabled exponential transition does not fire by time t. Therefore

$$E_{ii}(t) = e^{-\lambda_i t}, i = 3, 4, 5 \tag{13}$$

4.1 Steady state solution

Using the notation defined in Section 2, we evaluate the following quantities:

$$\alpha_{11} = \int_0^\infty E_{11}(t)dt \tag{14}$$

$$= \frac{1}{\lambda_1}\left[1 - e^{-\lambda_1 \delta}\right] \tag{15}$$

$$\alpha_{12} = \int_0^\infty E_{12}(t)dt$$

$$= \frac{1}{\lambda_2} - \frac{\lambda_1}{\lambda_1 - \lambda_2}\left[\frac{e^{-\lambda_2 \delta}}{\lambda_2} - \frac{e^{-\lambda_1 \delta}}{\lambda_1}\right] \tag{16}$$

$$\alpha_{ii} = \int_0^\infty E_{ii}(t)dt \tag{17}$$

$$= \frac{1}{\lambda_i}, i = 3, 4, 5 \tag{18}$$

The steady state probabilities for the embedded DTMC are obtained by using Equation (4) as following:

$$\nu_1 = 1/2 \tag{19}$$

$$\nu_3 = \frac{1}{2}e^{-\lambda_1 \delta} \tag{20}$$

$$\nu_4 = \frac{1}{2} - \frac{\lambda_1}{2(\lambda_1 - \lambda_2)}e^{-\lambda_2 \delta} - \frac{\lambda_2}{2(\lambda_1 - \lambda_2)}e^{-\lambda_1 \delta} \tag{21}$$

$$\nu_5 = \frac{\lambda_1}{2(\lambda_1 - \lambda_2)}\left[e^{-\lambda_2 \delta} - e^{-\lambda_1 \delta}\right] \tag{22}$$

Next, the steady state probabilities of the MRGP being in state $i, 1 \leq i \leq 5$, are obtained in *closed form* by plugging in values from Equations (14) through (17) and Equations (19) through (22) in Equation (5). The steady state probability of the software being unavailable is given by the probability that the process is in state 3, 4 or 5. Therefore

$$\pi_{down} = \pi_3 + \pi_4 + \pi_5. \tag{23}$$

Let T_d be a random variable denoting the down time of software, then the expected down time in the interval $[0, T]$ would be

$$E[T_d] = \pi_{down} T \tag{24}$$

Since rejuvenation is typically scheduled for times when the costs incurred will be less, it may be appropriate to measure the performance in terms of costs incurred rather than in terms of down time. Let C_r be the fixed cost per unit time when the software is in the rejuvenation phase and C_f be the fixed cost per unit time when it has failed. If C is a random variable denoting the cost incurred, then, the expected total cost incurred in the interval $[0, T]$ is given by

$$E[C] = ((\pi_3 + \pi_5)C_r + \pi_4 C_f)T \tag{25}$$

We have obtained closed-form expressions for the expected down time and expected down costs which, for a given T are functions of $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ and δ. For each, differentiating with respect to δ and equating to zero gives us the optimal value of δ that minimizes the respective quantity.

5 Results

Values of λ_1 through λ_5 are fixed for all the results and are taken from page 15 of [9]. Table 1 lists these values.

Parameter	*Value*
λ_1^{-1}	240 hrs.
λ_2^{-1}	2160 hrs.
λ_3	6 /hr.
λ_4	2 /hr.
λ_5	6 /hr.

Table 1: Parameter Values

Transient results:
We chose the value of δ to be 336 hours(also from [9]). To obtain the results we evaluated $\overset{*}{\mathbf{K}}(s)$ and $\overset{*}{\mathbf{E}}(s)$ and programmed Equation (6) in Mathematica to obtain $\overset{*}{\mathbf{V}}(s)$. Then we used Jagerman's method to numerically invert $\overset{*}{V}_{13}(s) + \overset{*}{V}_{14}(s) + \overset{*}{V}_{15}(s)$. Figure 3 shows the transient probability of the software being unavailable (due to rejuvenation as well as failure).

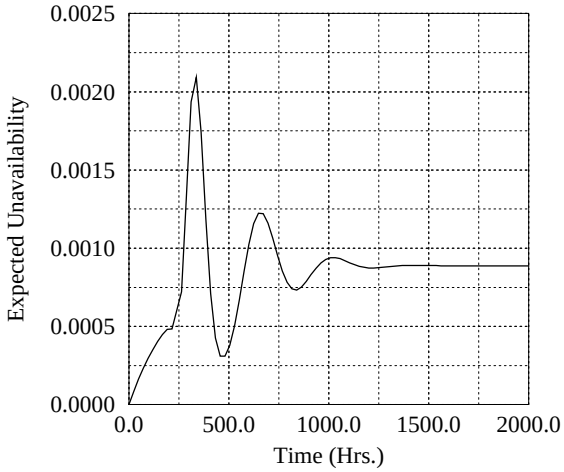

Figure 3: Transient expected unavailability of software

From the plot, we can see that the expected unavailability ripples as time increases finally settling down to the steady state value. Also, the local maxima of the first ripple occurs at 336 hrs. The maxima of ith ripple for $i \geq 2$ occurs at $(i+1)336 + i\frac{1}{6}$. Note that $\frac{1}{6}$ is the time it takes to complete the rejuvenation. Also note that there are minor "kinks" in the graph which result from a loss of precision in the numerical inversion of Laplace transform.

Steady state results:
We are not only interested in the steady state probability of the software being unavailable given a rejuvenation interval, but also on the effect of the interval on expected unavailability. Figure 4 shows the effect on expected down time by change in the rejuvenation interval(δ). In the extreme case, if $\delta = 0$, i.e., the software is always rejuvenating, it would also be always unavailable. As δ is increased, expected unavailability decreases. As it is performed less and less frequently, the unavailability caused by rejuvenation is overshadowed by the down time due to failure. Thus expected unavailability achieves a minimum and then increases again. It was found that the optimal rejuvenation interval is 33 days. As $\delta \to \infty$, expected unavailability approaches the value when there is no rejuvenation.

Figure 5 shows the change in expected costs as δ is varied. For illustration purpose, we assume that the cost incurred when software fails (i.e., C_f) is fixed at \$5000 /hr. The results are plotted for different values of C_f/C_r. As with the down time, expected cost is a function with the optimum value of δ at its minimum. As the ratio C_f/C_r is increased, which means keeping C_f fixed at 5000\$/hr. and decreasing C_r, we see that the optimal value of δ also decreases. Thus higher the cost of failure relative to the cost of maintenance, more often would we perform rejuvenation. In the limit $\delta \to \infty$, i.e., no rejuvenation, the expected cost is a function of C_f only and hence all graphs approach the same value. Also, in the limit $\delta \to 0$, the cost incurred is infinite, as the software always remains in the rejuvenation state.

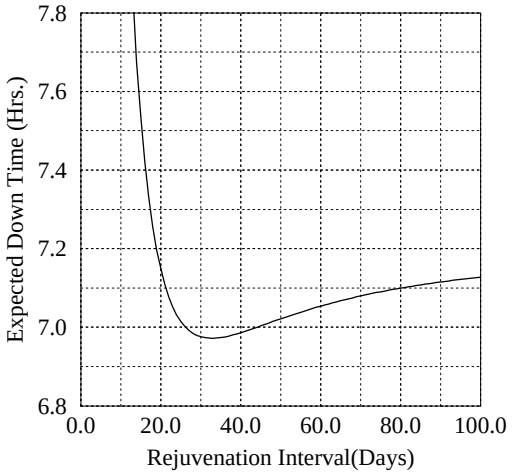

Figure 4: Steady state expected down time versus rejuvenation interval

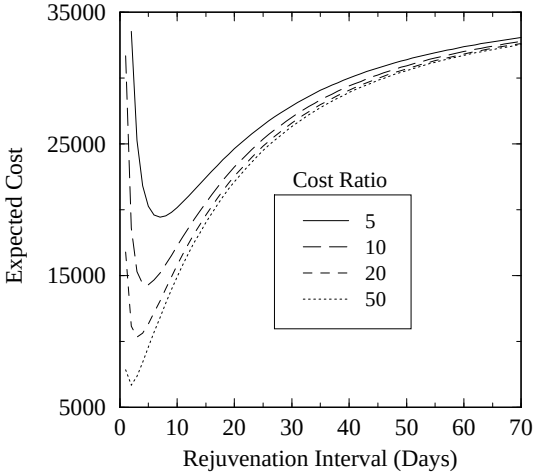

Figure 5: Steady state expected cost versus rejuvenation interval

6 Conclusion

We have presented an MRSPN based model of software rejuvenation which illustrates the change in expected down time and expected total cost with respect to the rejuvenation interval. The model can be used to find the optimum rejuvenation interval given the system parameters. The model provides the probability that the software is unavailable in both the transient and steady state cases. In the presented model, rejuvenation is solely time based. Since the cost incurred is a function of the load on the service provided by the software, rejuvenation may also be a function of load. We are currently investigating rejuvenation policies which are purely load dependent as well as policies which are both time and load dependent.

References

[1] A. Avizienis, "The n-version approach to fault

tolerant software", *IEEE Trans. on Software Engg.*, Vol. SE-11, No. 12, pp. 1491-1501, December 1985.

[2] R. Chillarege, S. Biyani and J. Rosenthal, "Measurements of failure rate in commercial software", To appear in *Proc. of 25th Symposium on Fault Tolerant Computing*, June, 1995.

[3] H. Choi, V.G. Kulkarni, and K. Trivedi, "Markov regenerative stochastic petri nets, *In G. Iazeolla and S.S. Lavenberg, eds. PERFORMANCE-93*, pp. 339-356, 1993.

[4] G. Ciardo, R. German and C. Lindmann, "A characterization of the stochastic process underlying a stochastic Petri net", *IEEE Transaction on Software Engineering*, Vol. 20, pp. 506-515, 1994.

[5] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala and K.S. Trivedi. Automated generation and analysis of Markov reward models using stochastic reward nets. In *Linear Algebra, Markov Chains, and Queueing Models*, Carl Meyer and R.J. Plemmons (eds.), IMA Volumes in Mathematics & its Applications, Vol. 48, pp 145-191, Springer Verlag, Heidelberg, 1993.

[6] E. Cinlar, *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs, 1975.

[7] R. Geist, M. Smotherman, K.S. Trivedi and J.B. Dugan, "The reliability of life-critical computer systems", *Acta Informatica, Vol. 23*, pp:621-642, 1986.

[8] R. German and C. Lindemann, "Analysis of stochastic Petri nets by the method of supplementary variables", *Performance Evaluation*, Vol. 20, Nos. 1-3, 317-335, 1994.

[9] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, "Software rejuvenation: Analysis, Module and Applications", To appear in *Proc. of 25th Symposium on Fault Tolerant Computing*, June, 1995.

[10] Y. Huang and C.M.R. Kintala, "Software implemented fault tolerance: Technologies and experience", *In Proc. of 23rd Intl. Symposium on Fault-Tolerant Computing*, Toulouse, France, pp. 2-9, June 1993.

[11] P. Jalote, Y. Huang and C. Kintala, "A framework for understanding and handling transient software failures", *In Proc. of 2nd ISSAT Intnl. Conf. on Reliability and Quality in Design*, Orlando, Florida, 1995

[12] D.L. Jagerman, "An inversion technique for the Laplace transform", *The Bell System Technical Journal*, 61:1995-2002, October 1982.

[13] V.G. Kulkarni, *Modeling and analysis of stochastic systems*, Chapman Hall, 1995.

[14] P.A. Lee, "Software-faults: The remaining problem in fault tolerant systems?", *In Eds. M. Banatre and P.A. Lee, Hardware and Software Architectures for Fault Tolerance: Experiences and Perspectives*, Lecture Notes in Computer Sc., Springer Verlag, Vol. 774, pp 171-181, 1994

[15] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, U.S.A., 1981.

[16] B. Randell, "System structure for software fault tolerance" , *IEEE Trans. on Software Engg.*, Vol. SE-1, pp. 220-232, June 1975.

[17] Y.S. Sherif and M.L. Smith, "Optimal maintenance models for systems subject to failure-a review", *Naval Research Logistics Quarterly*, vol 28, 47-74 (1981)

[18] M. Telek, "Some advanced reliability modeling techniques", *Ph.D. Thesis*, Hungarian Academy of Science, Budapest, Hungary, 1994.

[19] K.S. Trivedi, G. Ciardo, M. Malhotra and S.Garg, "Dependability and performability analysis using stochastic Petri nets", in *Proc. of 11th Intnl. Conf. on Analysis and Optimization of Systems-DES*, Sophia-Antipolis, France, June 1994.