2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) | 978-1-6654-7679-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISSREW55968.2022.00060

# A Markov Regenerative Model of Software Rejuvenation Beyond the Enabling Restriction

Laura Carnevali
*Dept. of Information Engineering*
*University of Florence*
Florence, Italy
laura.carnevali@unifi.it

Marco Paolieri
*Dept. of Computer Science*
*University of Southern California*
Los Angeles, USA
paolieri@usc.edu

Riccardo Reali
*Dept. of Information Engineering*
*University of Florence*
Florence, Italy
riccardo.reali@unifi.it

Leonardo Scommegna
*Dept. of Information Engineering*
*University of Florence*
Florence, Italy
leonardo.scommegna@unifi.it

Enrico Vicario
*Dept. of Information Engineering*
*University of Florence*
Florence, Italy
enrico.vicario@unifi.it

***Abstract*—Software rejuvenation is a proactive maintenance technique that counteracts software aging by restarting a system or some of its components. We present a non-Markovian model of software rejuvenation where the underlying stochastic process is a Markov Regenerative Process (MRGP) beyond the enabling restriction, i.e., beyond the restriction of having at most one general (GEN, i.e., non-exponential) timer enabled in each state. The use of multiple concurrent GEN timers allows more accurate fitting of duration distributions from observed statistics (e.g., mean and variance), as well as better model expressiveness, enabling the formulation of mixed rejuvenation strategies that combine time-triggered and event-triggered rejuvenation. We leverage the functions for regenerative analysis based on stochastic state classes of the ORIS tool (through its SIRIO library) to evaluate this class of models and to select the rejuvenation period achieving an optimal tradeoff between two steady-state metrics, availability and undetected failure probability. We also show that, when GEN timers are replaced by exponential timers with the same mean (to satisfy enabling restriction), transient and steady-state are affected, resulting in inaccurate rejuvenation policies.**

***Index Terms*—Software rejuvenation, Markov regenerative processes, enabling restriction, bounded regeneration restriction, stochastic state classes, stochastic time Petri nets.**

## I. Introduction

Software aging increases the failure rate and reduces the performance of software systems [1]–[7] due to *Aging-Related Bugs* (ARBs). This class of software faults manifests its effects over time, affecting different types of software systems including cloud infrastructures, operating systems, database management systems, web servers, and web applications [1], [2], [5], [8], [9]. As time passes and service requests are completed, ARBs can be activated and propagate, resulting in a chain of threats [10] that lead the system towards error states with increasing failure probability, until an aging-related failure occurs (e.g., a process crash due to memory allocation failure caused by a memory leak that progressively reduced available resources).

ARBs often manifest themselves as Mandelbugs [11], which are hard to remove before deployment and operation due to the limited increase of detection probability with respect to testing efforts, and which are often hidden in interactions with the execution environment or with third-party integrated libraries [2], [12]–[14]. To avoid or mitigate disruptive failures, unnecessary resource consumption, and other aging effects, a viable alternative to fault removal is represented by a preventive and proactive maintenance technique termed *software rejuvenation*, which, in its basic form, consists in repeatedly stopping the system or some selected system component, cleaning its internal state, and restarting it [15]. This approach improves system reliability by mitigating error accumulation and propagation, but it normally reduces availability due to downtimes for cleaning the system state. The selection of rejuvenation times thus subtends a crucial trade-off between software qualities of reliability and performance efficiency [16].

A large variety of quantitative models of Software Aging and Rejuvenation (SAR) have been proposed to derive an optimal rejuvenation schedule, for different types of software systems and rejuvenation strategies, resulting in different classes of underlying stochastic process [17].

In several SAR models, the system behavior is abstracted as a *Continuous Time Markov Chain* (CTMC), either defined by direct identification of states [15], [18] or specified through a high level modeling formalism [19], often in the class of Stochastic Petri Nets.

Models in the class of *Semi Markov Processes* (SMPs) [20] have been proposed to improve validity by representing behaviors where the future evolution depends not only on the current logical state (as in a CTMC) but also on the time spent in the state. These models are usually expressed by direct identification of a set of states, with sojourn durations associated with general (GEN, i.e., non-exponential) distributions fitting experimental data [21]–[25].

Expressive power is further improved by models identifying

978-1-6654-7679-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ISSREW55968.2022.00060

an underlying *Markov Regenerative Process* (MRGP, often abbreviated as MRP) [20]. While SMPs impose that the model satisfies the Markov condition (clearing memory of the past history) at each transition, MRGPs only require the Markov condition to be eventually satisfied with probability 1 at some time called *regeneration point*. The evolution of the process can be represented through behaviors where the system depends only on the history accumulated since the last regeneration point. In models of SAR, this expressiveness becomes crucial to represent aging phases during the interval between subsequent rejuvenations, as in the seminal model of [26] specified as a Markov Regenerative Stochastic Petri Net (MRSPN), and in various subsequent works addressing virtualized [12], [13] and clustered systems [27], [28]. As a common trait, in all these models, the expressive power of MRGPs is limited by the so-called *enabling restriction*, which requires at most one GEN timer to be present in the model (or in each state). Specifically, a deterministic (DET) transition is used to represent the rejuvenation period, and exponential (EXP) timers are used for all the remaining durations.

In this paper, we explore the potential of SAR models that break the limit of the enabling restriction and permit representation of concurrent timers with GEN distributions. In this case, numerical solution can be performed using the method of stochastic state classes [29] implemented in the SIRIO library [30] of the ORIS tool [31]. This method supports transient and steady-state analysis of models that satisfy a so-called *bounded regeneration restriction* [32], requiring that a regeneration is always reached within a bounded number of discrete events, and that GEN durations are either DET or expressed as Exponomial (often called Expolynomial) distributions, with a possibly bounded support.

To this end, we extend the SAR model of [26] by using exponomial GEN distributions with bounded support to fit the mean values of repair and rejuvenation activities reported in [26]. We show how the representation of durations with GEN distributions impacts the evaluation of the optimal rejuvenation period. We then show how the extended expressive power allows us to analyze a model of rejuvenation policies combining time-triggered and event-based strategies.

The rest of the paper is organized as follows. In Section II, we present an MRGP model of software rejuvenation beyond the enabling restriction, and we perform its analysis to derive the optimal rejuvenation period. In Section III, analysis results are compared with those obtained for a model variant satisfying the enabling restriction. In Section IV, the expressive power of models beyond enabling restriction is put to work to combine time-triggered and event-based rejuvenation strategies. Finally, we draw our conclusions in Section V.

## II. Optimal Periodic Rejuvenation Beyond Enabling Restriction

In this section, we extend a rejuvenation model from the literature by using multiple concurrent GEN timers, which result in an underlying stochastic process from a more general class of MRGPs (Section II-A). Then, we show how the ORIS tool can be used during system design to identify the rejuvenation period that optimizes the trade-off between metrics of reliability, which are evaluated as model rewards (Section II-B).

| Transition | Expected Value (hours) |
|---|---|
| rejFromErr | 0.1666 |
| rejFromUp | 0.1666 |
| error | 240 |
| fail | 2160 |
| detect + repair | 0.5 |

TABLE I: Expected values of timers used in the MRGP under enabling restriction of [26]. Stochastic parameters of timers in Fig. 1b are selected to obtain the same expected values.

### A. Software aging and rejuvenation models

A well-known model of software aging and rejuvenation was defined in [26]. In this work, we extend the structure of such model to introduce a state where a failure has occurred but has not been detected; the resulting model is presented in Fig. 1a. The model is defined as an STPN describing the system aging, failure, and repair process (blue section), and the related rejuvenation process (green section). Initially, the system is in a safe state (`Up`); due to ARBs, the system transitions to an error state (`Err`), which eventually leads to a state of failure (`Down`); then, the failure is detected (`Detected`) and the system is repaired, returning to the safe state. Concurrently, the rejuvenation process waits in the initial state (`Clock`) for an amount of time equal to the rejuvenation period; after this time, rejuvenation is ready to start (`Rej`): error or failure events are inhibited, and rejuvenation is initiated in safe or error states of the system (through transitions `rejFromUp` and `rejFromErr`, respectively). After rejuvenation, the system goes back to a safe state and the rejuvenation process restarts.

In Fig. 1a, similarly to [26], the rejuvenation period is modeled with a DET timer (gray bar) while all other activities are modeled with EXP timers (white bars), so that the model is under the enabling restriction. In this case, the transient behavior of the system can be evaluated in closed form, by inversion of the matrix form of the generalized Markov renewal equations, where global and local kernels are derived in the Laplace-Stieltjes domain. Transient probabilities of this model can also be computed through regenerative transient analysis based on the method of stochastic state classes [29], [33]. This method removes the enabling restriction, allowing the analysis of the model in Fig. 1b, where multiple GEN timers (thick black bars) can be enabled concurrently. We assign the following GEN distributions in Fig. 1b:

- `error` and `fail` are modeled with 4-phase Erlang distributions fitting the expected values in Table I;
- `detect` is modeled with a uniform distribution over $[0, 0.3]$, with mean value $0.15$;

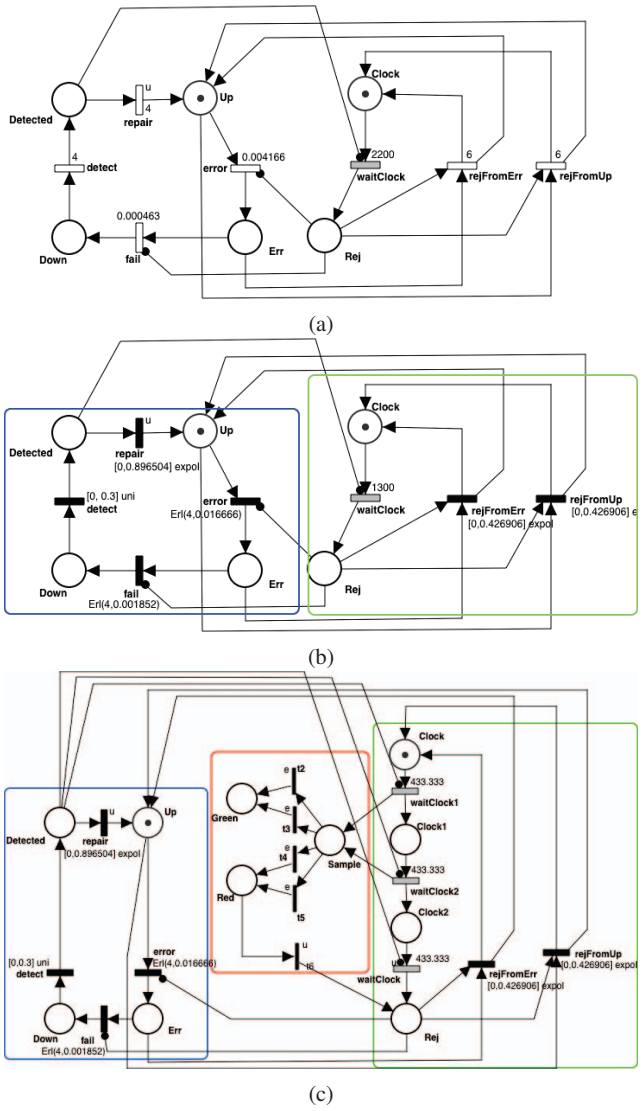



Fig. 1: Software aging and rejuvenation models: (a) extends [26] to distinguish undetected failures and has an underlying MRGP under the enabling restriction; (b) and (c) use multiple GEN timers and have underlying MRGPs beyond the enabling restriction; (b) has a periodic rejuvenation schedule, while (c) introduces a diagnostic event-driven rejuvenation policy discussed in Section IV.

- The other timers are modeled with truncated exponential distributions, with density $f(x) = 2.04\exp(-1.51x)$ over $[0, 0.90]$ for `repair` and $f(x) = 4.28\exp(-3.17x)$ over $[0, 0.43]$ for `rejFromErr` and `rejFromUp`. The upper bounds and rates of these distributions were selected to fit the expected values in Table I and a coefficient of variation equal to $1/\sqrt{2}$ [34]; `repair` has mean value $0.35$ to match the mean value of $0.5$ used in Table I for the sum of the duration of `repair` and `detect`.

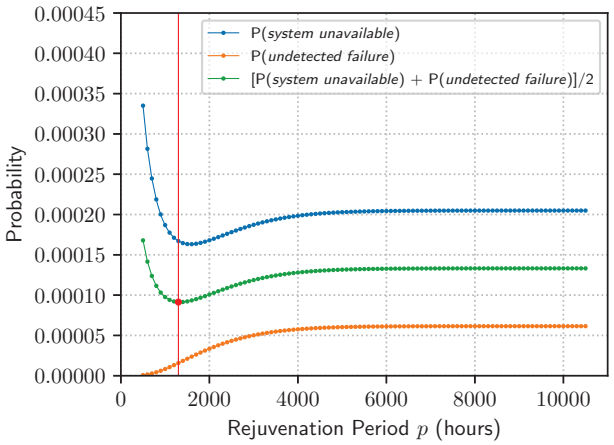



Fig. 2: Steady-state unavailability, undetected failure probability, and their average as a function of the rejuvenation period for the model in Fig. 1b (multiple GEN timers).

### B. Optimal rejuvenation period

In the context of software aging and rejuvenation, steady-state *unavailability* is a metric of interest quantifying the occurrence of states in which the system is not able to provide service. We consider an additional metric, the steady-state probability of an *undetected failure*, which can be of particular concern for system designers. Both metrics can be evaluated using the ORIS GUI and the SIRIO library using the *rewards* "`If(Down+Detected>0||Rej>0,1,0)`" and "`Down`", respectively. We are interested in selecting the rejuvenation period $p^*$ that results in the best trade-off between steady-state unavailability $\bar{a}(p)$ and probability of undetected failure $\bar{r}(p)$. Specifically, we minimize the average of the two metrics:

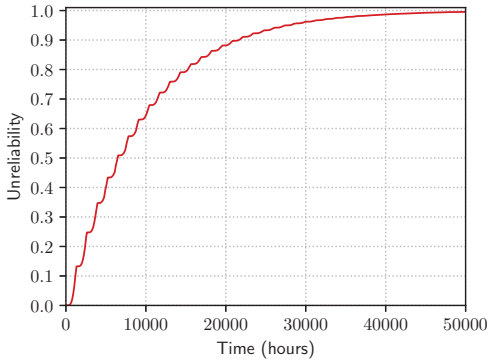
$$p^* = \arg\min_p \left[\frac{\bar{a}(p) + \bar{r}(p)}{2}\right]. \quad (1)$$

The ORIS tool and the SIRIO library provide manual and automated modeling capabilities, respectively; we use the SIRIO library to evaluate $[\bar{a}(p) + \bar{r}(p)]/2$ for 100 variants of the model in Fig. 1b, with rejuvenation period $p$ ranging from $500$ to $10400$ hours, with increments of $100$ hours.

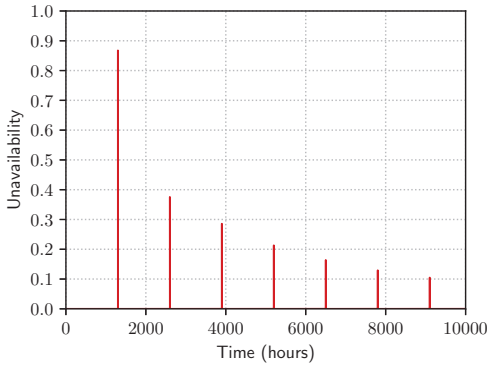
The results, presented in Fig. 2, show that, as expected, the steady-state probability of an undetected failure $\bar{r}(p)$ increases with the rejuvenation period $p$, converging to $0.000061$. On the other hand, the steady-state unavailability $\bar{a}(p)$ initially decreases with respect to the rejuvenation period $p$, reaching the minimum value of $0.000163$ for a period of $1600$ hours, and then increases, converging to the value $0.000205$. The rejuvenation period minimizing the average of these metrics in Eq. (1) is $p^* = 1300$, achieving the objective $0.000091$.

### C. Transient reliability metrics

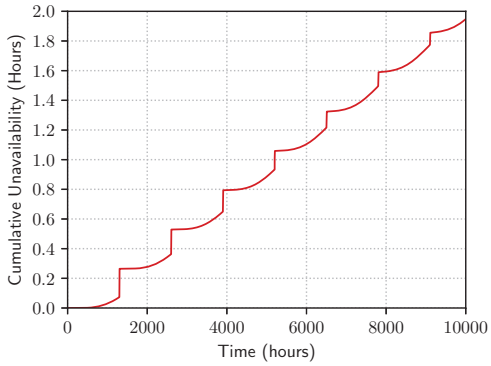
After identifying the optimal rejuvenation period, we use the ORIS tool to evaluate three transient metrics: *unreliability*, *unavailability*, and *cumulative unavailability*. We evaluate




(a)




(b)




(c)

Fig. 3: Transient evaluation of (a) unreliability, (b) unavailability, and (c) cumulative unavailability for the model in Fig. 1b.

the expected steady-state unreliability by using the reward "`Down`" and the stop condition "`Down==1`" (i.e., the system has encountered at least one failure); unavailability is evaluated using the same reward used in the previous section, "`If(Down+Detected>0||Rej>0,1,0)`"; cumulative unavailability at time $t$ is obtained in ORIS by integrating the instantaneous unavailability in $[0, t]$. The results show that system unreliability (Fig. 3a) converges to 1 after $t = 50000$ hours; at multiples of the rejuvenation period $p = 1300$, the increase in unreliability is reduced, confirming the importance of the rejuvenation processes. In contrast, unavailability (Fig. 3b) shows sharp peaks near multiples of the rejuvenation period, since the system is not available during rejuvenation; as time advances, due to the randomness of repair times (after which the rejuvenation clock is restarted), the unavailability peaks are in fact distributed over longer periods of time, with lower maximum probability values. Correspondingly, we observe a sharp increase in the expected cumulative unavailability (Fig. 3c) near multiples of the rejuvenation period; as time advances, cumulative unavailability becomes smoother.

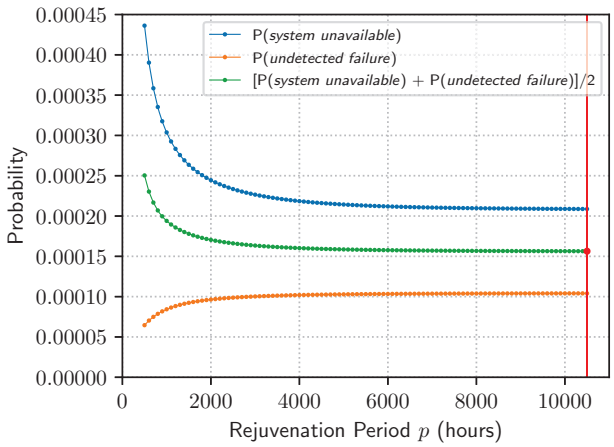



Fig. 4: Steady-state unavailability, undetected failure probability, and their average as a function of the rejuvenation period for the model in Fig. 1a (under enabling restriction).

## III. Comparing Enabling Restriction with Bounded Regeneration Restriction

In this section, we repeat the selection of an optimal rejuvenation period presented in Section II-B for a model under the enabling restriction (Section III-A). Then, we evaluate the impact of the enabling restriction on the transient metrics presented in Section II-C, illustrating that transition distributions greatly impact the choice of the rejuvenation period, despite having the same mean values (Section III-B).

### A. Optimal rejuvenation period

We repeat the analysis of Section II-B for the model presented in Fig. 1a, which includes a DET timer (`waitClock`) for the rejuvenation period and only EXP timers (instead of GEN timers) for the other activities, with rates that result in the same expected values. The underlying stochastic process is an MRGP under enabling restriction, since at most one non-exponential timer can be enabled in each state.

The results, presented in Fig. 4, illustrate that the steady-state values of both rewards have a monotonic trend, which is decreasing for unavailability and increasing for the probability of undetected failure. In this case, by applying again the criterion of Eq. (1) to find the optimal rejuvenation period,

the largest rejuvenation period is selected; similarly, after extending the range used for the rejuvenation period, the best strategy is to perform rejuvenation as rarely as possible, suggesting that rejuvenation is not beneficial for the metrics in Eq. (1). However, this is in contrast with the literature on software aging and rejuvenation: in fact, the considered MRGP model under enabling restriction is characterized by stochastic parameters that, despite having the same mean values, are not sufficiently accurate to identify an optimal rejuvenation period. Since the model does not allow us to select an optimal rejuvenation period, we select $p^* = 2200$ in order to anticipate the mean time of error detection (2300 hours) by a few hours.

### B. Transient reliability metrics

We also investigate the impact on transient behavior when GEN transitions are replaced in the model of Fig. 1b with EXP transitions with the same mean values (model of Fig. 1a). To this end, we analyze the two models for rejuvenation periods equal to either $p = 1300$ or $p = 2200$, respectively, and we evaluate cumulative unavailability over $t = 6000$ hours and unreliability over $t = 50000$ hours.

The cumulative unavailability graphs (Fig. 5) show that the model under enabling restriction converges to the steady-state more rapidly than the model with bounded regeneration (i.e., beyond enabling restriction), a consequence of replacing GEN transitions (with bounded supports) with EXP transitions (with unbounded supports). Moreover, for the optimal rejuvenation period $p = 1300$ (Fig. 5a), unavailability of the model under enabling restriction is always greater than that of the model with bounded regeneration. This suggests that, although the two models have transitions characterized by distributions with the same expected value, the different nature of their analytical forms produces significantly different transient behaviors. Notably, the transient metrics of the model under enabling restriction are closer to those of the model with bounded regeneration for the larger rejuvenation period $p = 2200$.

Fig. 6 illustrates that the model with bounded regenerations (beyond enabling restriction) achieves lower unreliability, i.e., the system is less likely to fail during the same time interval. This behavior is more pronounced when the optimal rejuvenation period $p = 1300$ is considered: for the model with bounded regenerations, unreliability converges to 1 only after $t = 50000$ hours of operation, while $t = 20000$ hours are sufficient for the model under enabling restriction.

This comparison explains the difficulty of optimizing system parameters using a model under enabling restriction, where only mean values of observed activity durations can be accurately represented, resulting in notably different performance metrics. In contrast, models with bounded regeneration can include GEN timers to fit multiple statistics (e.g., mean and variance) of collected data, as well as multiple DET timers; the increased expressive power allows modeling of more advanced rejuvenation strategies, such as those presented in the next section.

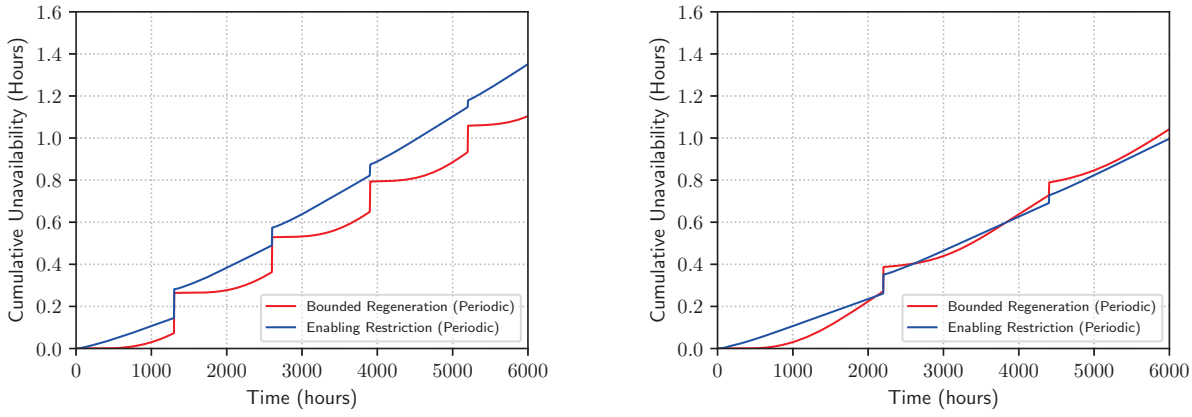
## IV. Event-driven rejuvenation policy

In this section, we illustrate how the expressive power of models beyond enabling restriction allows the study of new rejuvenation strategies combining periodic rejuvenation with event-driven rejuvenation based on system diagnostics (Section IV-A). Then, we compare steady-state and transient metrics of this model with those of the model using only periodic rejuvenation and analyzed in Section II (Section IV-B).

### A. Modeling event-driven rejuvenation policies

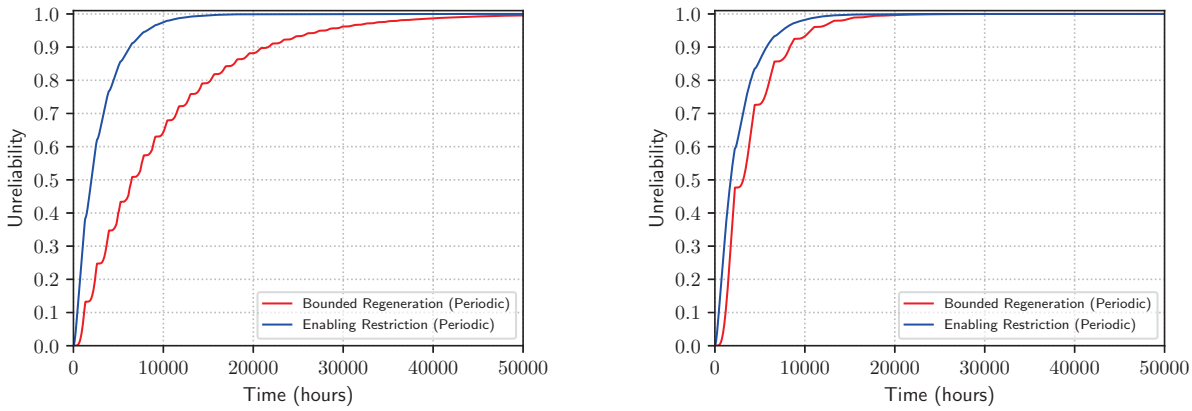
The model illustrated in Fig. 1b describes a system where rejuvenation is triggered periodically. In many concrete cases, the rejuvenation process can be guided by system diagnostics, which provide statistical insight into the safety status of the system. System diagnostics can be used to estimate the probability that the system is in an error state, determining whether to start rejuvenation. Combining the periodic rejuvenation approach with an event-based approach can result in improved management of an aging system, since rejuvenation can be anticipated when system diagnostics indicate high probability of an error state, while periodic rejuvenation can improve reliability when systems diagnostics are inaccurate.

In Fig. 1c, we propose a model combining the two approaches. The process consists of three steps (deterministic transitions in the green section): after the first two steps (`waitClock1` and `waitClock2`) diagnostic tests (red section) are started to determine whether to start rejuvenation immediately; if none of the diagnostic tests report a failure, then rejuvenation is started according to a time-triggered policy (`waitClock`). In the STPN model, when the diagnostic tests start, a token is put in place `Sample`, which is the initial state of the tests. If the diagnostic tests report a safe state, a token is put in place `Green`. Otherwise, an error or failure are likely to be present in the system; in this case, a token is put in place `Red`, vacant tokens are removed from rejuvenation and diagnostic processes (i.e., from places `Clock1`, `Clock2` and `Green`), and a token is put in place `Rej`. The diagnostic process can correctly or incorrectly predict whether the system state is safe. This requires modeling four different situations: the system is safe and no error is detected, the system is safe but an error is detected, the system is unsafe and an error is detected, the system is unsafe but no error is detected. The first case is modeled through transition `t2`, which is enabled when at least one token is in place `Up`. The undetected error state is modeled through transition `t3`: in this case, the transition is enabled when there is no token in place `Up`. Valid error or failure detection are modeled through transition `t4`, which is enabled when no tokens are in place `Up`. Invalid error or failure detection are modeled through transition `t5`, which is enabled when at least one token is in place `Up`. The probability of predicting a correct state is 0.9, both in the case of fault detection and in the case of safe state detection. The rest of the rejuvenation process (green section) and the aging and repairing process (blue section) are modeled and operate as described in Section II-A.




(a) (b)

Fig. 5: Cumulative transient unavailability of models in Figs. 1a and 1b with rejuvenation periods 1300 (a) and 2200 (b).




(a) (b)

Fig. 6: Transient unreliability of models in Figs. 1a and 1b with rejuvenation period equal to 1300 (a) and 2200 (b).

We also consider a variant of this model that triggers the rejuvenation (i.e., puts a token in place `Rej` before `waitClock` eventually triggers) only when system safety verification has produced *two warnings in a row*. To model this case using STPNs, it is sufficient to add the enabling function "`Red==2`" to transition `t6`, which will be enabled only when two tokens are present in place `Red`.

This model could be extended in multiple ways, for example by varying the number of diagnostic stations, or by using a different sub-model for each station. Although not possible in STPN models, the system aging process could be emulated in SIRIO by varying the probabilities for the execution of transitions `t1`, `t2`, `t3`, and `t4` based on the current system state; moreover, different probabilities of correct prediction could be used for fault detection and safe state detection.

### *B. Steady-state and transient metrics comparison*

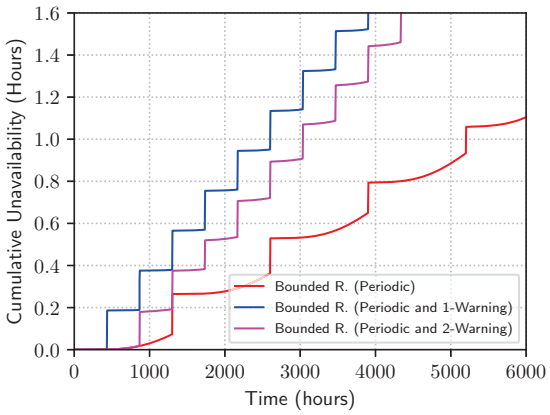
We consider the model of Section II-A (periodic rejuvenation) and the models defined in Section IV-A (rejuvenating periodically, or after 1 or 2 warnings), and we compare their steady-state unavailability and undetected failure probability, as well as transient unreliability and transient cumulative unavailability.

Steady-state metrics, presented in Table II, show that the probability of undetected failures can be reduced by an order of magnitude (from 0.0000157 to 0.0000017) by using diagnostics information to trigger rejuvenation on demand; the model that requires two consecutive diagnostic warnings results in greater steady-state probability of undetected failure (0.0000021). In contrast, additional rejuvenation processes triggered by diagnostics can result in worse steady-state availability, since the system is unavailable during rejuvenation. We observe that unavailability is lower (0.0001670) in the model with periodic rejuvenation; in the models with diagnostic stations and event-based rejuvenation, unavailability is lower (0.0003255) when two warnings are necessary to trigger rejuvenation instead of one (0.0003324).

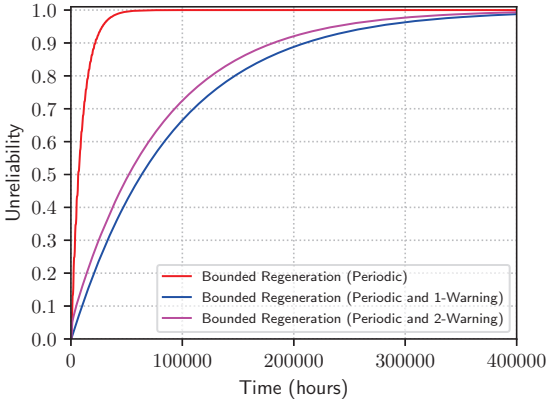
| Metric | Rejuvenation Model (Bounded Regeneration) | | |
|---|---|---|---|
| | Periodic | Periodic and 1-Warning | Periodic and 2-Warning |
| P(*Undetected Failure*) | 0.0000157 | 0.0000017 | 0.0000021 |
| P(*System Unavailable*) | 0.0001670 | 0.0003324 | 0.0003255 |

TABLE II: Steady-state metrics for the periodic model of Fig. 1b, and for variants of Fig. 1c using 1 or 2 warnings.

Fig. 7 illustrates the results obtained for transient metrics. The transient cumulative unavailability confirms our observations regarding steady-state metrics. The classical rejuvenation model presents lower values of unavailability, because it reduces the amount of time in which the system is under rejuvenation; between the two event-driven rejuvenation models, the model requiring two consecutive failure warnings has lower unavailability than the model requiring only one warning. On the other hand, transient unreliability shows that models with event-driven rejuvenation policies obtain higher reliability. In fact, while at $t = 50000$ hours the model with periodic rejuvenation policy is unreliable with probability close to 1, the event-driven models are unreliable with probability lower than 0.5. In addition, the unreliability of the model requiring two consecutive fault warnings is higher than the unreliability of the model requiring only one warning; this is expected, since system diagnostics are highly accurate in our model (the state of the system is correctly diagnosed with probability 0.9).




(a)




(b)

Fig. 7: Cumulative transient unavailability (a) and transient unreliability (b) of models in Figs. 1b and 1c.

## V. Conclusions

We addressed the problem of modeling and evaluating software rejuvenation approaches aimed at counteracting the software aging phenomenon. To this end, we presented a non-Markovian model with an underlying stochastic process from the class of MRGPs beyond the limit of the enabling restriction, i.e., with multiple concurrent GEN timers enabled in each state. The expressive power of this class of models enables the derivation of stochastic parameters that fit multiple statistics of observed, for example to preserve not only the sample mean value but also the sample variance. We evaluated reliability metrics for the proposed model and for a variant obtained by replacing GEN distribution with EXP distributions that fit only the mean value, in order to restrict the underlying MRGP with the enabling restriction. Experimental results show significant differences both in transient and steady-state behavior, with non-negligible impact on the selection of a rejuvenation period that achieves a trade-off between system availability and probability of undetected failures.

The increased expressive power of MRGPs beyond enabling restriction also allowed us to formulate and evaluate software rejuvenation models that combine the usual time-triggered rejuvenation policy with an event-triggered policy where warnings emitted by a diagnostic mechanism are used to trigger early rejuvenation. Experimental results show that the two event-triggered policies under consideration improve system reliability, at the cost of reducing its availability. Nevertheless, this comprises an acceptable cost, given that repairing a system after a failure is typically more expensive than performing a rejuvenation.

The results also show that the ORIS tool and the SIRIO library can be effectively used to design and evaluate quantitative models with underlying MRGPs beyond the enabling restriction, supporting parametric studies to select optimal stochastic parameters. More importantly, our analysis suggests that software aging and rejuvenation models beyond enabling restriction allow the evaluation of dynamic policies that improve system reliability, motivating the adoption of this class of models in the context of SAR.

## REFERENCES


[1] J. Araujo, R. Matos, V. Alves, P. Maciel, F. V. d. Souza, R. M. Jr, and K. S. Trivedi, "Software aging in the Eucalyptus cloud computing infrastructure: characterization and rejuvenation," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, pp. 1–22, 2014.

[2] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," *Performance Evaluation*, vol. 70, no. 3, pp. 163–178, 2013.

[3] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*. IEEE, 1998, pp. 283–292.

[4] Y. Bao, X. Sun, and K. S. Trivedi, "A workload-based analysis of software aging, and rejuvenation," *IEEE Transactions on Reliability*, vol. 54, no. 3, pp. 541–548, 2005.

[5] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *IEEE Transactions on reliability*, vol. 55, no. 3, pp. 411–420, 2006.

[6] T. Dohi, K. S. Trivedi, and A. Avritzer, *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*. World Scientific, 2020.

[7] J. Xiang, C. Weng, D. Zhao, A. Andrzejak, S. Xiong, L. Li, and J. Tian, "Software aging and rejuvenation in Android: new models and metrics," *Software Quality Journal*, vol. 28, no. 1, pp. 85–106, 2020.

[8] J. Parri, S. Sampietro, L. Scommegna, and E. Vicario, "Evaluation of software aging in component-based web applications subject to soft errors over time," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2021, pp. 25–32.

[9] R. Pietrantuono and S. Russo, "A survey on software aging and rejuvenation in the cloud," *Software Quality Journal*, vol. 28, no. 1, pp. 7–38, 2020.

[10] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[11] M. Grottke and K. S. Trivedi, "Fighting bugs: Remove, retry, replicate, and rejuvenate," *Computer*, vol. 40, no. 2, pp. 107–109, 2007.

[12] F. Machida, D. S. Kim, and K. S. Trivedi, "Modeling and analysis of software rejuvenation in a server virtualized system," in *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*. IEEE, 2010, pp. 1–6.

[13] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE, 2009, pp. 365–371.

[14] C. Weng, J. Xiang, S. Xiong, D. Zhao, and C. Yang, "Analysis of software aging in android," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016, pp. 78–83.

[15] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*. IEEE, 1995, pp. 381–390.

[16] I. O. for Standardization/International Electrotechnical Commission *et al.*, "ISO/IEC 25010: Systems and software engineering-systems and software quality requirements and evaluation (square)-system and software quality models," *Authors, Switzerland*, 2011.

[17] G. Ciardo, R. German, and C. Lindemann, "A characterization of the stochastic process underlying a stochastic Petri net," *IEEE Transactions on software engineering*, vol. 20, no. 7, pp. 506–515, 1994.

[18] G. Ning, K. S. Trivedi, H. Hu, and K.-Y. Cai, "Multi-granularity software rejuvenation policy based on continuous time Markov chain," in *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*. IEEE, 2011, pp. 32–37.

[19] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 311–332, 2001.

[20] V. G. Kulkarni, *Modeling and analysis of stochastic systems*. Chapman and Hall/CRC, 2016.

[21] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule," in *Proceedings. 2000 Pacific Rim International Symposium on Dependable Computing*. IEEE, 2000, pp. 77–84.

[22] W. Xie, Y. Hong, and K. Trivedi, "Analysis of a two-level software rejuvenation policy," *Reliability Engineering & System Safety*, vol. 87, no. 1, pp. 13–22, 2005.

[23] D. Chen and K. S. Trivedi, "Analysis of periodic preventive maintenance with general system failure distribution," in *Proceedings 2001 pacific rim international symposium on dependable computing*. IEEE, 2001, pp. 103–107.

[24] J. Zhao, Y. Wang, G. Ning, K. S. Trivedi, R. Matias Jr, and K.-Y. Cai, "A comprehensive approach to optimal software rejuvenation," *Performance Evaluation*, vol. 70, no. 11, pp. 917–933, 2013.

[25] F. Machida, V. F. Nicola, and K. S. Trivedi, "Job completion time on a virtualized server with software rejuvenation," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, pp. 1–26, 2014.

[26] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic Petri net," in *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*. IEEE, 1995, pp. 180–187.

[27] D. Wang, W. Xie, and K. S. Trivedi, "Performability analysis of clustered systems with rejuvenation under varying workload," *Performance Evaluation*, vol. 64, no. 3, pp. 247–265, 2007.

[28] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi, "Analysis and implementation of software rejuvenation in cluster systems," in *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2001, pp. 62–71.

[29] A. Horváth, M. Paolieri, L. Ridi, and E. Vicario, "Transient analysis of non-Markovian models using stochastic state classes," *Performance Evaluation*, vol. 69, no. 7-8, pp. 315–335, 2012.

[30] Sirio, "The Sirio Library for the Analysis of Stochastic Time Petri Nets." [Online]. Available: https://github.com/oris-tool/sirio

[31] M. Paolieri, M. Biagi, L. Carnevali, and E. Vicario, "The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems," *IEEE Trans. Software Eng.*, vol. 47, no. 6, pp. 1211–1225, 2021.

[32] M. Biagi, L. Carnevali, M. Paolieri, T. Papini, and E. Vicario, "Exploiting non-deterministic analysis in the integration of transient solution techniques for Markov regenerative processes," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2017, pp. 20–35.

[33] E. Vicario, L. Sassoli, and L. Carnevali, "Using stochastic state classes in quantitative evaluation of dense-time reactive systems," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 703–719, 2009.

[34] L. Carnevali, R. Reali, and E. Vicario, "Compositional evaluation of stochastic workflows for response time analysis of composite web services," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 2021, pp. 177–188.