# Optimal Software Rejuvenation for Tolerating Soft Failures

András Pfening [a,1], Sachin Garg [b,2], Antonio Puliafito [c], Miklós Telek [a,1] and Kishor S. Trivedi [b]

[a] *Department of Telecommunications, Technical University of Budapest, 1521 Budapest, Hungary*

[b] *Center for Advanced Comp. & Communications, Dept. of Electrical and Computer Engineering, Duke University, Durham, NC 27708, U.S.A.*

[c] *Ist. di Informatica e Telecom., Università di Catania, 95125 Catania, Italy*

**Abstract**

In recent studies, the phenomenon of software "aging" has come to light which causes performance of a software to degrade with time. Software rejuvenation is a fault tolerance technique which counteracts aging. In this paper, we address the problem of determining the optimal time to rejuvenate a server type software which experiences "soft failures" (witnessed in telecommunication systems) because of aging. The service rate of the software gradually decreases with time and settles to a very low value. Since the performability in this state is unacceptable, it is necessary to "renew" the software to its peak performance level. We develop Markov decision models for such a system for two different queuing policies. For each policy, we define the look-ahead-n cost functions and prove results on the convergence of these functions to the optimal minimal cost function. We also prove simple rules to determine optimal times to rejuvenate for a realistic cost criterion. Finally, the results are illustrated numerically and the effectiveness of the MDP model is compared with that of the simple rules.

## 1 Introduction

It has been observed that system failures due to imperfect software behavior are usually more frequent than failures caused by hardware components' faults

[1] Supported in part by Hungarian Research Foundation (OTKA) grant T-16637.
[2] Supported in part by a IBM Fellowhip.

[12]. Recovery blocks [10], N-version programming [2] and N-self checking programming [9] are some of the prominent techniques for tolerating software faults. Based on the principle of design diversity, these techniques are reactive in nature, i.e. they provide means of dealing with a fault after it has resulted in failure. A reactive approach based on data diversity has been proposed in [1].

In recent studies of software field failure data [12,5], it has been observed that a large percentage of failures are transient in nature, i.e. they may not occur again if the program were to be reexecuted. Such failures occur because of an undesirable state reached in the operating environment of the software. Moreover, it is also observed that owing to the presence of intermittent software faults called "Heisenbugs" [6] and interactions for sharing the hardware and operating system resources, such conditions accrue in time causing the software to "age" [7]. Memory bloating and leaks, unreleased file-locks and data corruption etc. are some typical causes of software aging. It may result in a gradual performance degradation of the software and/or a transient crash failure. For example, in telecommunication systems, the software which handles switching starts losing packets as its service rate degrades with time [3]. Although experiencing unacceptable packet loss, it does not crash and continues to be available. This situation is referred to as a "soft failure" as opposed to a "hard failure" when the software crashes and becomes unavailable. In both cases, restoration to a clean (unaged) state is necessary and is accomplished by stopping the software, cleaning its internal state and restarting it. *Huang et. al.* first suggested this technique which is preventive in nature and called it *Software Rejuvenation* [7]. Flushing buffer queues maintained by a server, garbage collection, reinitializing the internal kernel tables, cleaning up file systems are some examples of what cleaning might involve. A commonly known way of restoration is the "reboot" of a computer.

An important issue now is to determine the optimal time to perform this restoration. A continuous time Markov chain model was proposed [7] to determine if rejuvenation is beneficial for systems which experience crash failures. *Garg et. al.* [4] improved upon the model by allowing deterministic rejuvenation time and provided a closed form expression for the optimal rejuvenation interval which maximizes availability. Avritzer and Weyuker, on the other hand, showed how rejuvenation can be used to increase the performability of telecommunications software which experiences soft failures [3]. Here, it involved occasionally stopping the system, cleaning up, and restarting it from its peak performance level. They collected traffic data on an experimental system and proposed heuristics on good times to rejuvenate based on the observed traffic pattern.

In this paper, we study the latter class of systems from a theoretical standpoint. We develop a Markov decision process (MDP) based framework to deal

with the problem of determining optimal times to rejuvenate. In short, this paper consists of the optimal stopping problem as applied to software rejuvenation for tolerating soft-failures. We also consider a realistic cost criterion and prove simple rules which determine the optimal times to rejuvenate. Finally, we numerically compare the results obtained from these rules with those obtained by solving the MDP model. All of the above steps are performed for two different queueing policies. In the first policy (referred to as the *no buffer overflow* case), buffer overflow is not allowed. Whenever the buffer is full and a new packet arrives, the software is stopped and rejuvenated. In the second policy (referred to as the *buffer overflow* case), the buffer may overflow resulting in packet loss during normal operation.

The rest of the paper is organized as follows. We list the system assumptions and formally state the problem in Section 2. Section 3 contains the MDP model for the *no buffer overflow* case. We formulate the model and define a series of look-ahead-n cost functions which approximate the optimal minimal cost function and derive bounds on their convergence to the latter. We also consider a realistic cost criterion and prove simple rules to determine the optimal times to rejuvenate. In Section 4, all of the above steps are repeated for the *buffer overflow* case. We numerically illustrate the usefulness of various results in Section 5 and compare the MDP solution with the proposed rules. Finally, the paper is concluded in Section 6.

## 2 Problem Statement

The system under consideration consists of a software which services arriving packets. The software itself experiences aging, the effect of which is a gradual decrease in its service rate. Eventually, the service rate drops and settles to a low unacceptable value, yet the software continues to be available. In this situation, termed as a soft-failure, the arriving packets keep accumulating and eventually overflow the buffer. Excessive loss of packets makes it necessary to restore the software to its peak service capacity (rate) to achieve the desired performability. The problem is to determine when should the software be stopped for rejuvenation and requires minimizing certain function which captures the cost incurred due to soft failures. For example, in switching software, this cost is measured in terms of average number of packets lost per unit time. We assume that all packets arriving while rejuvenation is in progress as well as all those in the queue when it was initiated are lost.

Further, we assume that packet arrivals follow a Poisson process and the service times are identical, independent and exponentially distributed random variables. The degradation of the system is reflected in the decreasing service rate which is also assumed to be known as a function of time.

Following notation is used in the rest of the paper:

| | |
|---|---|
| $T$ | variable denoting the time until rejuvenation is initiated, |
| $T_R$ | time it takes to perform rejuvenation(constant), |
| $X$ | random variable denoting the number of clients in the queue at time $T$, i.e. when rejuvenation is initiated, |
| $Y$ | random variable denoting number of clients denied service when rejuvenation is in progress, i.e. in $(T, T + T_R)$, |
| $\lambda$ | packet arrival rate, |
| $\mu(t)$ | time dependent service rate, where $\lim_{t\to\infty} \mu(t) = \mu_\infty$, |
| $B$ | buffer length. |

## 3 Optimal Rejuvenation without Buffer Overflow

In this case, if the system is in a state such that the buffer is full and a new packet arrives, we immediately stop and rejuvenate the software thus avoiding buffer overflow. This is the case when it is not desirable to lose packets during normal operation. In other words, the fact that the buffer is full indicates that it is time to rejuvenate the software. The state of the system during normal operation can be fully described by the number of customers in the system and the time spent since last rejuvenation. In each state, we need to decide whether to continue service or to stop and rejuvenate the system.

### *3.1 Markov Decision Process Solution*

The optimization problem can be stated as: Find $T$ that minimizes the average cost of the run, i.e., $\min_T \left\{ E\left[\, \mathcal{C}(X,T,Y)\, \right] \right\}$, if $\lambda$, $\mu(t)$, $T_R$, $B$ are given and $\mathcal{C}(.)$ denotes the cost function. $Y$ is approximated by its expected value $\lambda T_R$.

First, we discretize the time in steps of size $\Delta$. The state of the system can then be represented as a 2-tuple $(i, j)$, where $i$ represents the number of packets in the software queue (including the one being serviced) and $j$ represents the integer number of $\Delta$ time units denoting the time spent since last rejuvenation. Our goal is to find the *optimal stationary policy* $f$, which in each state, dependent only on that state, determines whether to rejuvenate the system or to continue service. The policy is optimal in the sense that it minimizes the expected cost incurred in the process. Since the packet arrival follows a Poisson process and the service time in a state follows negative exponential distribution, we have a Markov Decision Process (MDP) which can be cast

as the optimal stopping problem. The nature of the cost function $C(i,j,a)$, defined as the cost of choosing action $a \in \{cont, rej\}$ ( where *cont* implies continue and *rej* implies stop and rejuvenate ) when the system is in state $(i,j)$, can be summarized as follows:

$$\begin{aligned} C(i,j,rej) &\geq 0,\ 0 \leq i \leq B,\ 0 \leq j, \\ C(i,j,cont) &= 0,\ 0 \leq i < B,\ 0 \leq j. \end{aligned}$$

All the costs are required to be nonnegative. $P_{i,j,k,l}(a)$ is defined as the probability of going from state $(i,j)$ to state $(k,l)$ when action $a$ is chosen. The transition probabilities are given as follows:

$$\begin{aligned}
&(i) \quad P_{\cdot,\cdot,stop,stop}(rej) &&= 1, && \\
&(ii) \quad P_{0,j,1,j+1}(cont) &&= \lambda\Delta + o(\Delta) && j \geq 0, \\
&(iii) \quad P_{0,j,0,j+1}(cont) &&= 1 - \lambda\Delta + o(\Delta) && j \geq 0, \\
&(iv) \quad P_{i,j,i+1,j+1}(cont) &&= \lambda\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0, \\
&(v) \quad P_{i,j,i-1,j+1}(cont) &&= \mu(j)\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0, \\
&(vi) \quad P_{i,j,i,j+1}(cont) &&= 1 - (\lambda + \mu(j))\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0,
\end{aligned}$$

where the state $(stop, stop)$ is when the process gets finished. All the other transition probabilities are irrelevant. $(i)$ describes the case when it is decided to perform rejuvenation. When we decide to continue service, $(ii) - (iii)$ describe the situation when the buffer is empty. In this case, either a new packet arrives or nothing happens during the current time slot. $(iv) - (vi)$ describe the cases when the buffer is not empty, where, in addition to the previous case a packet can leave the system if its service has been completed $((v))$.

If the system started in state $(i,j)$, then for any policy $\pi$, we define the expected cost as:

$$V_\pi(i,j) = E_\pi \left[ \sum_{w=0}^{\infty} C(i_w, j_w, a_w) \mid i_0 = i, j_0 = j \right], \quad 0 \leq i \leq B,\ 0 \leq j,$$

where $(i_w, j_w)$ denotes the system state and $a_w$ is the action taken according to the policy $\pi$ in $t = w\Delta$. Let $V(i,j) = \inf_\pi V_\pi(i,j), \quad 0 \leq i \leq B,\ 0 \leq j$. A policy $\pi^*$ is *optimal* if

$$V_{\pi^*}(i,j) = V(i,j), \quad \forall\, i,j :\ 0 \leq i \leq B,\ 0 \leq j.$$

If $f$ is a stationary policy which chooses actions according to (for $0 \leq i \leq$

$B, 0 \leq j$ )

$$f(i,j) = \arg\min_a \left\{ C(i,j,a) + \sum_{k=0}^{B-1} P_{i,j,k,j+1}(a) V(k,j+1) \right\}, \tag{1}$$

then $V_f(i,j) = V(i,j), \quad 0 \leq i \leq B, 0 \leq j$ and hence $f$ is optimal [11] ($\arg\min_a\{F(a)\}$ denotes the value of $a$ where $F(a)$ is minimal). Thus we have formulated the problem as a Markov Decision Process, for which a stationary optimal policy exists and is determined by Equation 1.

The next step is to derive $V(i,j), \forall(i,j)$, the minimal expected cost when the system started in state $(i,j)$. We shall first define a series of expected cost functions, $\{V_n(i,j)\}$, or look-ahead-$n$ cost functions that are decreasing with $n$ for all the states $(i,j)$ and are an upper bound on $V(i,j)$. Next, we shall show that the cost $C$ is the upper bound on the difference of the optimal and the look-ahead-$n$ cost functions. Therefore when $C$ tends to zero with time, the look-ahead cost function series $V_n$ converges to the minimal cost function $V$. The proofs of the above statements follow the approach given in [11]. Let

$$V_0(i,j) = C(i,j,rej) \quad 0 \leq i \leq B, 0 \leq j,$$

and for $n > 0, \ 0 \leq i \leq B, 0 \leq j,$

$$V_n(i,j) = \min \left\{ C(i,j,rej), \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont) V_{n-1}(k,j+1) \right\}. \tag{2}$$

If the system starts in state $(i,j)$, $V_n(i,j)$ is the minimal expected cost if the process can go at most $n$ stages before stopping and rejuvenating. By our definition of $C(i,j,a)$, the expected cost cannot increase if the process is allowed to continue. Therefore

$$V_n(i,j) \geq V_{n+1}(i,j) \geq V(i,j) \quad 0 \leq i \leq B, 0 \leq j. \tag{3}$$

The process is said to be *stable*, if $\lim_{n\to\infty} V_n(i,j) = V(i,j) \quad 0 \leq i \leq B, 0 \leq j$. Let us also define $C_{max}(j) = \max_i\{C(i,j,rej)\}, \quad 0 \leq j$.

**Theorem 1** *The difference between the look-ahead-n cost function and the minimal expected cost function satisfies the inequality:*

$$V_n(i,j) - V(i,j) \leq C_{max}(n+j) \quad 0 \leq i \leq B, 0 \leq j. \tag{4}$$

**Proof.** The proof of this theorem follows the approach of Theorem 6.13 in [11]. Let $f$ be an optimal policy, and let $T$ be a random variable denoting the time at which $f$ stops. Also, let $f_n$ be a policy which chooses the same actions as $f$ at times $0, 1, \ldots, n-1$, but chooses the action $rej$ at time $n$ (if it had not previously done so). Then,

$$
\begin{aligned}
&V(i,j) = \\
&V_f(i,j) = E_f\left[Z \mid T \leq n\right] P\{T \leq n\} + E_f\left[Z \mid T > n\right] P\{T > n\}, \\
&V_n(i,j) \leq \\
&V_{f_n}(i,j) = E_f\left[Z \mid T \leq n\right] P\{T \leq n\} + E_{f_n}\left[Z \mid T > n\right] P\{T > n\}.
\end{aligned}
$$

where $Z$ denotes the total cost incurred and everything is understood to be conditional on $i_0 = i, j_0 = j$. Thus,

$$
\begin{aligned}
V_n(i,j) - V(i,j) \leq & \left(E_{f_n}\left[Z \mid T > n\right] - E_f\left[Z \mid T > n\right]\right) P\{T > n\} \\
\leq & E_{f_n}\left[Z \mid T > n\right],
\end{aligned}
$$

since $E_f\left[Z \mid T > n\right] \geq 0$ (all the costs are nonnegative) and $P\{T > n\} \leq 1$.

In the case $f_n$ stops only after $n$ stages, then $E_{f_n}\left[Z \mid T > n\right] \leq C_{max}(n + j)$.

In the case $f_n$ stops after $k < n$ stages which happens because doing the remaining $n-k$ steps would be more expensive, i.e., $E_{f_n}\left[Z \mid T > n\right] \leq C_{max}(n + j)$. □

Summarizing, we can define an optimal policy $f$ based on the minimal cost function $V$. $V$ is not known, but can be approximated by the look-ahead cost function series $V_n$. We shall refer to this approximation procedure as the *MDP algorithm* in the sequel. If $C$ converges to zero with time then the approximation is stable. An upper bound on the speed of convergence of the cost function series $V_n$ to $V$ is given by Theorem 1.

This result shows that the MDP algorithm can be used when the conditions of Theorem 1 hold, otherwise the convergence of the algorithm is not guaranteed. However, depending on the time unit ($\Delta$) and the time scales of the queueing process (arrival, service) the algorithm may require a large number of steps to yield the optimal result.

We do not know what $n$ is sufficiently large to get the optimal decision. In other words, when is $V_n$ close enough to $V$ to result in the same policy, i.e., $f_n = f$. In the following theorem, we prove that if certain conditions hold for a

state, then the decision made by the look-ahead policy calculated for a certain depth for that state, is the same as the optimal policy would make.

**Theorem 2** *(i) If* $\exists\, n_0 : f_{n_0}(i,j) = cont$ *then* $\forall n \geq n_0 : f_n(i,j) = cont$ *and* $f(i,j) = cont$*, i.e., the optimal policy will also decide to continue service in this state.*

*(ii) If* $\exists\, n_0 : f_{n_0}(i,j) = rej$ *and*

$$C(i,j,rej) < \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont)(V_{n_0}(k,j+1) - C_{max}(n_0+j+1)) \tag{5}$$

*then* $\forall n \geq n_0 : f_n(i,j) = rej$ *and* $f(i,j) = rej$*, i.e., the optimal policy will decide to stop and rejuvenate in state* $(i,j)$*.*

**Proof.**

(i) Since $f_n$ chooses an action ( for $0 \leq i \leq B$, $0 \leq j$ ) according to

$$f_n(i,j) = \arg\min_a \left\{ C(i,j,a) + \sum_{k=0}^{B-1} P_{i,j,k,j+1}(a) V_n(k,j+1) \right\},$$

and $C(\cdot,\cdot,cont) = 0$ and the process is finished after a $rej$ decision, $f_{n_0}(i,j) = cont$ implies that

$$C(i,j,rej) \geq \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont) V_{n_0}(k,j+1).$$

From Equation (3), the function series $V_n$ is monotone decreasing. Hence

$$\sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont)\ V_{n_0}(k,j+1) \geq \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont)\, V_{n_0+1}(k,j+1),$$

and as a consequence $f_{n_0+1}(i,j) = cont$. From Equation (3) $V_n(i,j) \geq V(i,j)$, therefore the optimal policy will also decide to continue.

(ii) $f_{n_0}(i,j) = rej$ implies that

$$C(i,j,rej) < \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont) V_{n_0}(k,j+1).$$

From Theorem 1 $V_{n_0}(i,j) - C_{max}(n_0+j) \leq V(i,j)$. By simply substituting

in the RHS of (5), we obtain

$$C(i,j,rej) < \sum_{k=0}^{B-1} P_{i,j,k,j+1}(cont)V(k,j+1),$$

i.e., the optimal policy will stop and rejuvenate in state $(i,j)$.

On the other hand, if we suppose that $\exists\, n \geq n_0$ such that $f_n(i,j) = cont$ then by Theorem 2/i, $f(i,j) = cont$ would follow, which is a contradiction. □

As we shall show via a numerical example in Section 5, Theorem 2 can be used to speed up the MDP algorithm and also to determine the sufficient depth up to which it should be run. Note that the transition probabilities of the underlying Markov process have a special structure, namely nonzero probabilities lead only to the next time step. The state space of this system is illustrated in Figure 2, where the small squares refer to the states; the horizontal axis represents the number of time steps, while the vertical axis represents the buffer content. If we follow a sample path of the process, in each time step we move to the next column of squares. So if the conditions of Theorem 2 hold for all the states represented by the first $k$ columns of Figure 2 then for larger $n$ these columns can be ignored since the optimal decision is known for them and they have no effect on the rest of the states. Similarly, if the conditions hold for all the states of interest, the optimal solution is found.

The calculations determined by Theorem 2 are relatively simple with the magnitude of operations given by $O(nBT + n^2B)$. As per our notation, $B$ is the buffer size, $T$ is time range that is studied, and $n$ is the depth of analysis (look-ahead-$n$ is used).

### *3.2 A Realistic Cost Function*

In telecommunication systems, a lost packet either causes retransmissions or is permanently lost. Either ways, a cost is incurred. In this section, we consider the average number of packets lost per unit time as a cost criterion given by

$$C(b,t,rej) = \frac{b + \lambda T_R}{t + T_R}.$$

Note that the time $t$ is no longer discrete. Since $b \leq B$, $\lim_{t\to\infty} C_{max}(t) = 0$, the MDP solution will work according to Theorem 1. However, for this cost function the optimal decision can be derived explicitly for a large range of

states. We also derive an upper limit $n_U$ for the depth of the analysis such that if $n \geq n_U$ then $f_n \equiv f$.

**Theorem 3** *(i) If* $b \geq (\lambda - \mu(t))t - \mu(t)T_R$ *holds for* $1 \leq b \leq B$, *then* $f(b,t) = cont$.
*(ii)* $\forall b, 0 \leq b \leq B : f(b,0) = cont$.

**Proof.** The condition for continuing the service is

$$C(b,t,rej) \geq \sum_{k=0}^{B-1} P_{b,t,k,t+\delta}(cont) V(k,t+\delta),$$

where $\delta$ denotes an arbitrarily small time interval. Since $V(k,t+\delta) \leq C(k,t+\delta,rej)$, if

$$C(b,t,rej) \geq \sum_{k=0}^{B-1} P_{b,t,k,t+\delta}(cont) C(k,t+\delta,rej),$$

the service should be continued. Substituting the cost function, we have the next inequalities:

– if $1 \leq b \leq B$, $\quad \dfrac{b+\lambda T_R}{t+T_R} \geq$

$$\lambda\delta \frac{b+1+\lambda T_R}{t+T_R+\delta} + \mu(t)\delta \frac{b-1+\lambda T_R}{t+T_R+\delta} + (1-(\lambda+\mu(t))\delta)\frac{b+\lambda T_R}{t+T_R+\delta},$$

– if $b = 0$, $\quad \dfrac{\lambda T_R}{t+T_R} \geq$

$$\lambda\delta \frac{1+\lambda T_R}{t+T_R+\delta} + (1-\lambda\delta)\frac{\lambda T_R}{t+T_R+\delta},$$

from which:

– if $1 \leq b \leq B$

$$b \geq (\lambda - \mu(t))t - \mu(t)T_R, \tag{6}$$

– if $b = 0$

$$0 \geq \lambda\delta t, \tag{7}$$

□

In the case of a nonempty buffer, Theorem 3, gives a simple rule to decide about the continuation of service which says if the service intensity is not less than the arrival rate, $\lambda \leq \mu(t)$, the service should be continued independent of the number of packets in the system. For the case of an empty buffer, we did not get a general simple rule. Equation (7) holds only for $t = 0$, i.e., at $t = 0$ we should continue service. For the rest of the states, in which the buffer is empty, the MDP algorithm needs to be run to determine the optimal decision. As we can see if the buffer contains more packets than a certain limit at time $t$, the service should be continued - the more the number of packets in the buffer, the more is the need to continue service.

**Theorem 4** *If $\exists\, t_{limit}$ such that in $t_{limit}$ the software will be stopped and rejuvenated anyway, then if $B \leq (\lambda - \mu(t))t - \mu(t)T_R$ then $f(b,t) = rej$ for $\forall b : 0 \leq b \leq B$.*

**Proof.** Suppose that $f(b, t+\delta) = rej\ \forall b, 0 \leq b \leq B$. The condition for stopping the service in $t$ is

$$C(b,t,rej) \leq \sum_{k=0}^{B-1} P_{b,t,k,t+\delta}(cont)V(k,t+\delta).$$

Since $V(k,t+\delta) = C(k,t+\delta,rej)$, if

$$C(b,t,rej) \leq \sum_{k=0}^{B-1} P_{b,t,k,t+\delta}(cont)C(k,t+\delta,rej)$$

holds, then the service should be stopped. Substituting the cost function, we have

- $1 \leq b \leq B$, $\qquad \dfrac{b+\lambda T_R}{t+T_R} \leq$

$$\lambda\delta\,\frac{b+1+\lambda T_R}{t+T_R+\delta} + \mu(t)\delta\frac{b-1+\lambda T_R}{t+T_R+\delta} + (1-(\lambda+\mu(t))\delta)\frac{b+\lambda T_R}{t+T_R+\delta}$$

- $b = 0$, $\qquad \dfrac{\lambda T_R}{t+T_R} \leq$

$$\lambda\delta\,\frac{1+\lambda T_R}{t+T_R+\delta} + (1-\lambda\delta)\frac{\lambda T_R}{t+T_R+\delta}$$

Simplifying the results we have:

- $1 \leq b \leq B$

$$b \leq (\lambda - \mu(t))t - \mu(t)T_R \tag{8}$$

– $b = 0$

$$0 \leq \lambda \delta t \tag{9}$$

Since $b \leq B$ and (9) holds for all $t \geq 0$, the theorem is proven. □

Since $\mu(t)$ is decreasing such that $\lambda > \mu(t)$ for large $t$, the condition of the statement will be eventually satisfied.

In the above theorem, an upper limit for the time to stop the system has been derived. Together with the result of Theorem 3 it may be enough to determine the optimal policy in practical cases, since we know the optimal decision for $t \geq \frac{B+\mu(t)T_R}{\lambda-\mu(t)}$ and for $t \leq \frac{b+\mu(t)T_R}{\lambda-\mu(t)}$, where $b$ is the buffer content at time $t$. The region where we have no explicit optimal decision is

$$\frac{b+\mu(t)T_R}{\lambda-\mu(t)} \leq t \leq \frac{B+\mu(t)T_R}{\lambda-\mu(t)}.$$

If this region is narrow enough then there is no need to run the MDP algorithm. On the other hand, if we want to know the optimal policy in the region where Theorem 3 and Theorem 4 do not help, we need to run the MDP algorithm. However, we know that if $n \geq n_U = \frac{t_{limit}}{\Delta}$ then $f_n \equiv f$, since the optimal decision in $t \geq t_{limit}$ is known, i.e., Theorem 4 reduces the problem to a finite time problem. The assumption that the system will be stopped at a time $t_{limit}$ does not imply finite time analysis since its value is not assumed to be known.

## 4 Buffer Overflow Case

In this case, we assume that when the buffer is full, any new packet arriving is lost. The system in this state need not be stopped and rejuvenated, To analyze such a case, we need to introduce another variable to describe the system state since the total number of lost packets since last rejuvenation need to be remembered.

### *4.1 MDP Solution*

The optimization problem is slightly modified by introducing a new random variable, $L$, denoting the total number of packets lost at time $T$ when rejuvenation is decided:

Find $T$ that minimizes the average cost of the run, $\min_T \left\{ E\left[\mathcal{C}(X,T,Y,L)\right] \right\}$, if $\lambda$, $\mu(t)$, $T_R$ and $B$ are given. The cost function structure is defined as follows:

$$C(i,j,k,rej) \geq 0, \quad 0 \leq i \leq B,\ 0 \leq j, 0 \leq k \leq j,$$
$$C(i,j,k,cont) = 0, \ \ 0 \leq i < B,\ 0 \leq j, 0 \leq k \leq j,$$

where $i$ and $j$ are as defined in Section 3 and $k$ denotes the number of lost packets until time $t = j\Delta$. The same approximation is used for $Y$. Again, all the costs are required to be nonnegative.

We define $P_{i,j,k,p,q,r}(a)$ as the probability of going from state $(i,j,k)$ to state $(p,q,r)$ when action $a$ is chosen. In our case the transition probabilities are defined as follows:

$$\begin{aligned}
&(i)\ P_{\cdot,\cdot,\cdot,stop,stop,stop}(rej) = 1, \\
&(ii)\ P_{0,j,k,1,j+1,k}(cont) = \lambda\Delta + o(\Delta) && j \geq 0,\ 0 \leq k \leq j, \\
&(iii)\ P_{0,j,k,0,j+1,k}(cont) = 1 - \lambda\Delta + o(\Delta) && j \geq 0,\ 0 \leq k \leq j, \\
&(iv)\ P_{i,j,k,i+1,j+1,k}(cont) = \lambda\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0, \\
& && 0 \leq k \leq j, \\
&(v)\ P_{i,j,k,i-1,j+1,k}(cont) = \mu(j)\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0, \\
& && 0 \leq k \leq j, \\
&(vi)\ P_{i,j,k,i,j+1,k}(cont) = 1 - (\lambda + \mu(j))\Delta + o(\Delta) && 1 \leq i < B,\ j \geq 0, \\
& && 0 \leq k \leq j, \\
&(vii)\ P_{B,j,k,B-1,j+1,k}(cont) = \mu(j)\Delta + o(\Delta) && j \geq 0,\ 0 \leq k \leq j, \\
&(viii)\ P_{B,j,k,B,j+1,k+1}(cont) = \lambda\Delta + o(\Delta) && j \geq 0, 0 \leq k \leq j, \\
&(ix)\ P_{B,j,k,B,j+1,k}(cont) = 1 - (\lambda + \mu(j))\Delta + o(\Delta) && j \geq 0,\ 0 \leq k \leq j,
\end{aligned}$$

where the state $(stop, stop, stop)$ is when the process gets finished. All the other transition probabilities are irrelevant. The above definitions $(i) - (ix)$ follow the same discipline as in Section 3 with the slight difference that additional transition probabilities are defined for the states when the buffer is full and service continuation is chosen $((vii) - (ix))$.

For any policy $\pi$ ( for $0 \leq i \leq B$, $0 \leq j$, $0 \leq k \leq j$ ),

$$V_\pi(i,j,k) = E_\pi \left[ \sum_{w=0}^{\infty} C(i_w, j_w, k_w, a_w) \mid i_0 = i, j_0 = j \right]$$

is defined to be the expected cost if the process started in state $(i,j,k)$. $(i_w, j_w, k_w)$ denotes the process state at $t = w\Delta$, and $a_w$ is the action taken at $t = w\Delta$ according to the policy $\pi$.

Let

$$V(i,j,k) = \inf_{\pi} V_\pi(i,j,k), \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

The policy $\pi^*$ is optimal if

$$V_{\pi^*}(i,j,k) = V(i,j,k), \quad \text{for all } i,j,k:\ 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

If $f$ is a stationary policy which chooses action according to

$$f(i,j,k) = \arg\min_a \left\{ C(i,j,k,a) + \sum_{p=0}^{B-1} \sum_{r=0}^{j+1} P_{i,j,k,p,j+1,r}(a)\ V(p,j+1,r) \right\}, \tag{10}$$

where $0 \le i \le B$, $0 \le j$, $0 \le k \le j$, then

$$V_f(i,j,k) = V(i,j,k), \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j$$

and hence $f$ is optimal [11]. Thus we have formulated the problem as a Markov Decision Process, for which a stationary optimal policy exists, and is determined by Equation 10.

Carrying on in the same way as in Section 3, Let

$$V_0(i,j,k) = C(i,j,k,rej) \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j,$$

and for $n > 0$,

$$V_n(i,j,k) = \min \left\{ C(i,j,k,rej),\ \sum_{p=0}^{B} \sum_{r=0}^{j+1} P_{i,j,k,p,j+1,r}(cont) V_{n-1}(p,j+1,r) \right\},$$

where $0 \le i \le B$, $0 \le j$, $0 \le k \le j$. If we start in state $(i,j,k)$, $V_n(i,j,k)$ is the minimal expected cost if the process can go at most $n$ stages before stopping. By the structure of $C(i,j,k,a)$, the expected cost cannot increase if the process is allowed to continue. Therefore,

$$V_n(i,j,k) \ge V_{n+1}(i,j,k) \ge V(i,j,k) \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

The process is said to be *stable*, if

$$\lim_{n\to\infty} V_n(i,j,k) = V(i,j,k) \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

Let us also define

$$C_{max}(j) = \max_{i,k}\{C(i,j,k,stop)\} \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

**Theorem 5** *The difference of the minimal expected cost function and the look-ahead-n cost function satisfies the next inequality:*

$$V_n(i,j,k) - V(i,j,k) \le C_{max}(n+j) \quad 0 \le i \le B,\ 0 \le j,\ 0 \le k \le j.$$

**Proof.** The proof follows exactly the same arguments as given in Theorem 1, Section 3. □

We can define an optimal policy $f$ based on the minimal cost function $V$. Once again, as shown for the *no-buffer-overflow case*, $V$ is approximated by the look-ahead cost function series $V_n$ (MDP algorithm). If the cost function that gives the cost of stopping in a state converges to zero with time, then the approximation is stable and an upper bound on the speed of convergence is given by Theorem 5.

If the conditions of the following theorem hold for a state when the look-ahead policy is calculated for a certain depth, then we know that the look-ahead-$n$ policy's decision is the same as the optimal policy would make.

**Theorem 6** *(i) If* $\exists\, n_0 : f_{n_0}(i,j,k) = cont$ *then* $\forall n \ge n_0 : f_n(i,j,k) = cont$ *and* $f(i,j,k) = cont$*, i.e., the optimal policy will also decide to continue service in state* $(i,j,k)$.

*(ii) If* $\exists n_0 : f_{n_0}(i,j,k) = rej$ *and*

$$C(i,j,k,rej) < \sum_{p=0}^{B}\sum_{r=0}^{j+1} P_{i,j,k,p,j+1,r}(cont)(V_{n_0}(p,j+1,r) - C_{max}(n_0+j+1))$$

*then* $\forall n \ge n_0 : f_n(i,j,k) = rej$ *and* $f(i,j,k) = rej$*, i.e., the optimal policy will decide to stop and rejuvenate in state* $(i,j,k)$.

**Proof.** Omitted because exactly the same arguments as used in Theorem 2, Section 3 apply. □

The number of operations is higher now due to the additional variable $L$ and is given by $O(nBT^2 + n^2BT + n^3B)$.

### *4.2 A Realistic Cost Function*

In addition to the packets lost due to rejuvenation, the system experiences a loss due to buffer overflow. Since the buffer overflow worsens as the software ages and the service rate decreases, the overall cost criterion must incorporate this loss too. Once again, the average number of packets lost per unit time gives a realistic cost and is defined as:

$$C(b, t, L, stop) = \frac{b + \lambda T_R + L}{t + T_R},$$

where the time is not discretized. For this cost function $\lim_{t\to\infty} C_{max}(t) = 0$ does not hold, therefore Theorem 5 cannot be applied [3] .

As for the *no buffer overflow* case, the optimal decision can be explicitly derived for this cost function also for a certain range of states. However, since the number of lost packets is not bounded from above, an explicit upper limit for the necessary depth of analysis cannot be determined. The results contain the random variable $L$, the total number of lost packets since last rejuvenation due to buffer overflow. The final formulae can be used to make "on-the-fly" decision of whether to continue service or to stop for rejuvenation, if it is possible to keep track of the number of packets lost during operation. In the remainder of this section, we present results which are similar in nature to those obtained for the *no-buffer-overflow* case in Section 3.2.

**Theorem 7** *(i) If* $b \geq (\lambda - \mu(t))t - \mu(t)T_R - L$ *holds for* $1 \leq b \leq B$, *then* $f(b, t) = cont$.
*(ii) If* $L \geq \lambda t$ *then* $f(0, t) = cont$.

**Proof.** The condition for continuing the service is

$$C(b, t, L, stop) \geq \sum_{k=0}^{B} \sum_{L=0}^{\infty} P_{b,t,L,k,t+\delta,l}(cont)V(k, t + \delta, l).$$

[3] However, if the cost function is modified to

$$C(b, t, L, stop) = \frac{b + \lambda T_R + L}{t^{1+\varepsilon} + T_R},$$

where $\varepsilon > 0$, $C_{max}(t)$ tends to zero with $t$, the condition of Theorem 5 holds.

Since $V(k, t+\delta, l) \leq C(k, t+\delta, l, rej)$, if

$$C(b, t, l, rej) \geq \sum_{k=0}^{B} \sum_{L=0}^{\infty} P_{b,t,L,k,t+\delta,l}(continue) C(k, t+\delta, l, rej).$$

holds, then the service should be continued. Substituting the cost function and simplifying the results we have:

- $b = B$

  $$B \geq (\lambda - \mu(t))t - \mu(t)T_R - L$$

- $1 \leq b \leq B - 1$

  $$b \geq (\lambda - \mu(t))t - \mu(t)T_R - L$$

- $b = 0$

  $$L \geq \lambda t$$

□

It is unlikely that the last rule derived for the empty buffer case will hold for $t > 0$. As a check on our results, notice that the derived decision rule for $b = B$ and $1 \leq b \leq B$ cases is the same. Moreover, if we substitute $L = 0$ in the final results, the expressions match exactly with those obtained in Section 3.2.

**Theorem 8** *If $\exists\, t_{limit}$ such that in $t_{limit}$ the system will be stopped and rejuvenated anyway, then if $B + L \leq (\lambda - \mu(t))t - \mu(t)T_R$ then $f(b, t) = rej$ for $\forall b : 0 \leq b \leq B$.*

**Proof.** Suppose that $f(b, t+\delta) = stop$ for $\forall b : 0 \leq b \leq B$. The condition for stopping the service in $t$ is

$$C(b, t, L, rej) \geq \sum_{k=0}^{B} \sum_{L=0}^{\infty} P_{b,t,L,k,t+\delta,l}(cont) V(k, t+\delta, l).$$

Since $V(k, t+\delta, l) = C(k, t+\delta, l, rej)$, if

$$C(b, t, l, rej) \leq \sum_{k=0}^{B} \sum_{L=0}^{\infty} P_{b,t,L,k,t+\delta,l}(cont) C(k, t+\delta, l, rej)$$

holds, then the service should be continued. Substituting the cost function and simplifying the results we have:

– if $b = B$

$$B \leq (\lambda - \mu(t))t - \mu(t)T_R - L,$$

– if $1 \leq b \leq B - 1$

$$b \leq (\lambda - \mu(t))t - \mu(t)T_R - L, \tag{11}$$

– if $b = 0$

$$L \leq \lambda t. \tag{12}$$

Since $b \leq B$ and (11) implies (12), the theorem is proven. □

The assumption that the system will be stopped and rejuvenated once is justified in this case as well. However, we can not claim that the condition of this theorem will always be eventually fulfilled. Therefore, as opposed to the *no buffer overflow* case, it is not possible to reduce the *overflow* case to a finite time problem.

The above theorem provides an optimal decision for $t \geq \frac{B+L+\mu(t)T_R}{\lambda-\mu(t)}$ and for $t \leq \frac{b+L+\mu(t)T_R}{\lambda-\mu(t)}$, where $b$ is the buffer content at time $t$, and $L$ is the number of lost customers in $(0, t)$ and can be used to make "on-the-fly" decisions, when $L$ is known. However, we can not determine the optimal decision when

$$\frac{b + L + \mu(t)T_R}{\lambda - \mu(t)} \leq t \leq \frac{B + L + \mu(t)T_R}{\lambda - \mu(t)}.$$

## 5 Numerical Example

In this section, we evaluate a simple system to demonstrate the applicability of the discussed methods for the *non-overflow* case, using the cost function discussed in Section 3.2. The buffer length is assumed to be 8, and the analysis included the first 26 time steps where $\Delta = 0.05$ and $T_R = 2\Delta$. We note that the values do not represent any real application and are chosen arbitrarily to illustrate the usefulness of the various results. The arrival rate and the service rate are shown in Figure 1 as functions of time. The decision map is illustrated in Figure 2. The black area refers to the states where Theorem 3 yields "continue" decision. On the other hand, using the result of Theorem 4

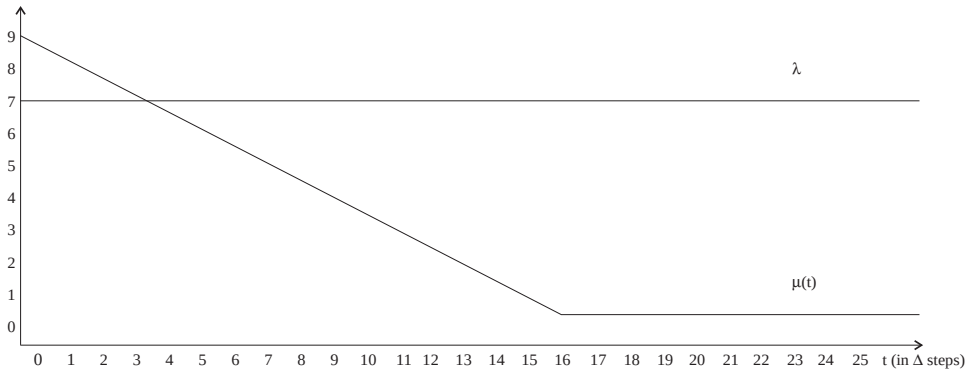


Fig. 1. Arrival rate ($\lambda$) and service rate ($\mu(t)$) of the analyzed system

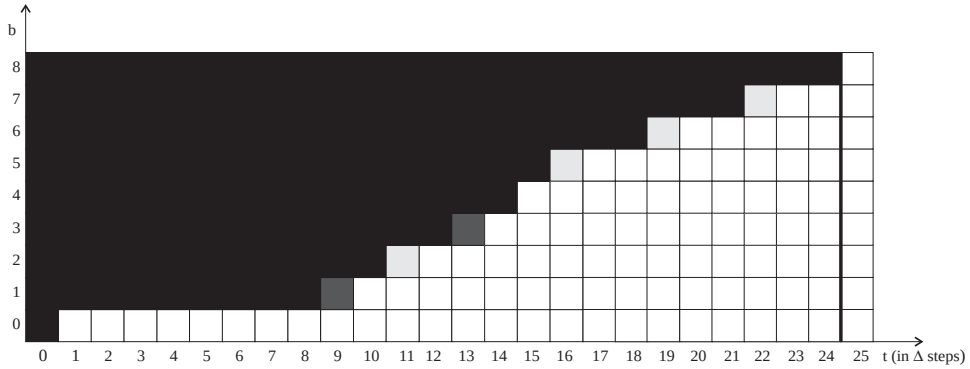


Fig. 2. Decision map of the analyzed system

we can predict the time limit of the "continue" decisions. Suppose that this limit will be where $\mu(t) = \mu = 0.5$ (see Figure 1):

$$t \geq \frac{B + \mu T_R}{\lambda - \mu} = \frac{8 + 0.0025}{7 - 0.5} \approx 1.23115 \approx 24.6\Delta,$$

i.e., we expect no "continue" decision beyond $24\Delta$, which is represented by the thick vertical line in the decision map. By applying Theorem 4 and Theorem 3 we know that the uncertain region is between the black area and the vertical line and the optimal policy is not predicted for these states.

We also run the MDP algorithm for the same cost function which verifies the above results. The MDP method has been programmed in *Mathematica 2.1* and it was run for the above system with several look-ahead depths. The light grey area (three states) refers to the states where (in addition to the black area) the MDP algorithm with depth 1 yielded "continue" decision, and the dark grey area (two states) refers to the states where (in addition to the black and light grey area) the MDP algorithm with depth 3 yielded "continue" decision. The algorithm was run with look-ahead-25 policy as well, but the decision map did not differ from the look-ahead-3 map. We know from Theorem 4 that there is no point in running the algorithm for higher depths. Unfortunately,

we could not make use of Theorem 2/ii since the condition of the statement was not fulfilled in any of the cases.

## 6 Conclusion

The problem of determining the optimal time to rejuvenate a server type software is studied in the paper from a theoretical standpoint. The software while serving incoming packets experiences soft failures due to aging whereby its service rate keeps decreasing with time eventually settling to a low unacceptable value. We developed MDP models for two queuing policies. In the first policy, buffer overflow is not allowed during normal operation by forcing the software to rejuvenate whenever the buffer is full. In the second policy the system may experience packet loss during normal operation due to buffer overflow. Each policy was modeled as the optimal stopping problem and results on the optimal decision of whether to continue service or to stop were derived. The MDP algorithm to find the optimal policy was shown to work if the cost function tends to zero with time. Moreover, results were derived to make the MDP algorithm converge faster.

We also evaluated the expected number of packets lost per unit time during a rejuvenation interval as a realistic cost function for each queuing policy. For the case when no buffer overflow is allowed, simple explicit rules are derived determining the optimal policy for most of the states. For the case when buffer overflow is allowed the rules are not explicit since they contain the number of lost packets as a variable.

The results for the no buffer overflow case were demonstrated via a simple numerical example. The simple rules provided an optimal decision for most of the states. The MDP algorithm confirmed the results obtained by applying the rules and provided the optimal decisions for the states not covered by the rules.

Further research directions include the application of more advanced queueing processes (like Semi-Markov Process or Markov Regenerative Process), and validating the model in practical applications. Another interesting aspect is to include customer waiting times in the cost function.

## Acknowledgement

The authors wish to thank S. Janakiram (University of North Carolina at Chapel Hill, Department of Operations Research) for his valuable suggestions.

## References


[1] P. E. Ammann and J. C. Knight, "Data-diversity: an approach to software fault-tolerance", *Proc. of 17th Intnl. Symp. on Fault Tolerant Computing*, pp. 122-126, June 1987.

[2] A. Avizienis, "The n-verion approach to fault-tolerant software", *IEEE Trans. on Software Engg.*, Vol. SE-11, No. 12, pp. 1491-1501, December 1985.

[3] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability", AT&T Bell Laboratories internal technical memorandum.

[4] S. Garg, A. Puliafito, M. Telek and K.S. Trivedi, "Analysis of software rejuvenation using Markov regenerative stochastic Petri net", To appear in *Proc. of Sixth Intnl. Symposium on Software Reliability Engineering*, Toulouse, France, October 24-27, 1995.

[5] J. Gray, "A census of tandem system availability between 1985 and 1990", *IEEE Trans. on Reliability*, Vol. 39, pp. 409-418, Oct. 1990.

[6] J. Gray, "Why do computers stop and what can be done about it?", *Proc. of 5th Symp. on Reliability in Distributed Software and Database Systems*, pp. 3-12, January 1986.

[7] Y. Huang, C. Kintala, N. Koletis, N. D. Fulton, "Software Rejuvenation- design, implementation and analysis", Proc. of *Fault-tolerant Computing Symposium*, Pasadena, CA, June 1995.

[8] P. Jalote, Y. Huang and C. Kintala, "A framework for understanding and handling transient failures", In *Proc. of 2nd ISSAT Intnl. Conf. on Reliability and Quality in Design*, March 8-10, 1995, Orlando, Florida, pp.231-237.

[9] J-C. Laprie, J. Arlat, C. Béounes and K. Kanoun, "Architectural issues in software fault-tolerance", *Software Fault Tolerance*, Ed. M. R. Lyu, John, Wiley & sons. ltd., pp. 47-80, 1995.

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

[11] S. M. Ross, *Applied Probability Models with Optimization Applications*. Dover Publications, Inc., New York, 1992.

[12] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability - A study of field failures in operating systems", in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 2-9, 1991.