



Article

A Comprehensive Model of Android Software Aging and Rejuvenation Considering Battery Saving

Vitaliy Yakovyna ^{1,2,*} , Bohdan Uhrynovskiy ² and Natalya Shakhovska ³ 

¹ Faculty of Mathematics and Computer Science, University of Warmia and Mazury in Olsztyn, ul. Oczapowskiego 2, 10-719 Olsztyn, Poland

² Software Department, Lviv Polytechnic National University, 12 S. Bandery St., 79013 Lviv, Ukraine

³ Artificial Intelligence Department, Lviv Polytechnic National University, 12 S. Bandery St., 79013 Lviv, Ukraine

* Correspondence: yakovyna@matman.uwm.edu.pl

Abstract: The more complex the software system, the greater the number of possible combinations of defects that can cause errors, resulting in software defects that are difficult to isolate and expensive to correct in the development stage. An essential feature of such defects is a gradual deterioration in software performance finishing with software failure—software aging. Mobile devices are particularly vulnerable to software aging. Thus, there is a constant need for methods and tools to eliminate the effects of aging in mobile systems based on modeling the aging process, including the study of metrics and aging factors and the development of more reliable and adequate aging and rejuvenation models. This paper summarizes the previously developed Android software aging and rejuvenation models and presents a comprehensive model of aging and rejuvenation for the Android operating system. The comprehensive model is based on continuous-time Markov Chains and considers different aging levels, mobile device activity, and battery status. The aging and rejuvenation model can be used to assess the software quality, allows obtaining expressions for indicators of software rejuvenation efficiency, and can be used to design and select parameters of the software rejuvenation method considering battery saving.



Citation: Yakovyna, V.; Uhrynovskiy, B.; Shakhovska, N. A Comprehensive Model of Android Software Aging and Rejuvenation Considering Battery Saving. *Electronics* **2023**, *12*, 1600. <https://doi.org/10.3390/electronics12071600>

Academic Editors: Seonah Lee, Jinhyun Kim, Suwon Lee and Iouliia Skliarova

Received: 4 March 2023

Revised: 24 March 2023

Accepted: 27 March 2023

Published: 29 March 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: software aging; software rejuvenation; mobile system; battery saving; continuous-time Markov chain

1. Introduction

Today, software has become one of the essential elements in ensuring the work of companies and organizations in their business environment. Software is used in everyday life by ordinary people on PCs, mobile devices, watches, etc. The needs of software users and their applications are often critical to human life and health, such as banking, health and treatment, and public services. Thus, software quality characteristics such as reliability and performance are critical features of most modern technology and mobile systems in particular.

The rapid development of technology contributes to a significant increase in the complexity of the software, which in turn increases the likelihood of errors in the software. Any software is a system set of interacting elements in a particular environment. Each system element may contain various defects that can lead to software errors. The more complex the system, the greater the number of possible combinations of defects that can cause errors. There is a category of software defects in such conditions that are difficult to isolate and expensive to correct at the development stage. An essential feature of defects that are difficult to isolate is that they do not cause failures immediately but accumulate in the system's state from the moment the system starts, which causes a gradual deterioration in software performance only after a specific random interval leads to software failure.

The described process of accumulating defects and errors in software, which leads to deterioration of performance and reliability, is called the aging process. The aging phenomenon in real systems manifests as memory leaks, unblocked file locks, data corruption, and numerous buildups of errors, slowly degrading system performance and causing software failure.

A recognized cost-effective tool to combat aging is the implementation of the so-called software rejuvenation procedure, which consists of cleaning the system from accumulated errors. For example, manually or automatically rebooting the device allows returning the system to a state with higher performance and a lower probability of aging failure. An important task is to develop effective mechanisms and strategies for rejuvenation and plan rejuvenation at the optimal time, which will maximize the time the system is in a state of high performance and reliability and minimize the likelihood of downtime.

Mobile devices are particularly vulnerable to the aging process. Users often use their phone for a long time without rebooting it, and their specifications are limited compared to PCs or server systems. The Android mobile OS is currently being actively studied in the context of aging software.

The main contribution of this paper is the development of a comprehensive model of aging and rejuvenation for the Android operating system, which is based on continuous-time Markov chains and considers different aging levels, mobile device activity, and battery status. This aging and rejuvenation model can be used to assess the software quality, allows obtaining expressions for indicators of software rejuvenation efficiency, and can be used to design and select parameters of the software rejuvenation method considering battery saving.

2. Problem Statement

Markov and semi-Markov processes are the basis for many models of software aging [1–7]. The first work [1] on software aging modeled software aging using continuous-time Markov chains (CTMC), which are the basic model for describing the phenomenon. This basic model was extended in many ways; for example, [2] uses inhomogeneous CTMC, where the residence time in each state is non-exponentially distributed, and [3] uses semi-Markov processes, where the transition intensities from one state to another depend on the current state.

Although most studies use classical Markov and semi-Markov processes, there are other types of modeling. For example, the authors of [2,4,5] used the Markov regenerative process (MRGP) in combination with stochastic Petri net (SPN) to build a simple but general model for estimating the optimal rejuvenation schedule in a software system. MRGP is the generalization of Markov and semi-Markov processes that can record the behavior of real systems with deterministic and exponentially time-distributed events.

The problem of finding the optimal rejuvenation schedule can also be formulated as the Markov decision process (MDP), where time is discretized. The process represents one dimension of the state description; decisions in each state determine whether the system should be rejuvenated or not. The solution is to find the optimal strategy to minimize the cost function. An example is presented in [6], where the authors adapted the MDP to build a model of software rejuvenation in a telecommunications system that includes buffer overflow. In [7], a partially observable Markov decision process (POMDP) was also used to model the phenomenon.

Software rejuvenation [1,8] is an active technique to prevent and delay software aging. This approach involves regularly or irregularly resetting the system's internal state in such a way as to clear the accumulated errors of software aging. For example, restarting the OS can restore the "pure" initial state. Alternatively, only the aging-affected application can be restarted to perform the rejuvenation procedure. Other techniques, such as garbage collection or memory deallocation techniques, can be applied as well. Software rejuvenation is a form of software maintenance that differs from installing updates to correct errors [9] or reengineering software to address aging issues [10].

In [1,11,12], software rejuvenation has been identified as a cost-effective solution to eliminate the effects of aging and avoid aging-related failures that do not require knowledge of error sites related to aging or even the very fact of their existence.

The most important problem of software rejuvenation is planning its execution during the OS operation to improve reliability and reduce the system's cost of time and resources. Paper [1] presents a simple general model of aging and rejuvenation based on CTMC to analyze software rejuvenation (Figure 1). In this model, after starting, the system remains in the so-called "very reliable state" S_0 , in which the probability of aging failure is negligible. After some time with intensity a_{0P} , the system passes into the S_P state, characterized by a high likelihood of aging failure. In this state, the failure of aging can occur with the intensity of a_{PF} , and there will be a transition to the state of failure (S_F), followed by the recovery to the state of S_0 with the intensity of a_{F0} . If the system performs rejuvenation, it moves from the state S_P to S_R with the intensity a_{PR} and then to the reliable state S_0 with the intensity a_{R0} . This model calculates the expected downtime and its costs, which, in turn, allows us to analyze in which conditions the rejuvenation of the software is beneficial to ensure reliability.

Software aging is manifested not only in server or PC software but also in mobile applications such as Android [13]. Mobile devices are used continuously by users for a long time without rebooting, and their hardware resources are limited compared to PCs and servers. Thus, mobile devices are particularly vulnerable to software aging [12]. In addition, mobile devices have some features (such as battery dependence) that distinguish them from other software and hardware systems and should be considered in studies of software aging.

Previous research on the aging phenomenon was mainly focused on Linux, and research on Android is at an early stage. Although Android is a Linux-based system, it is impossible to replicate Linux research schemes completely [14]. Various methodologies and approaches have been developed for the Android platform to study software aging and its metrics and factors. Paper [15] proposed an approach to studying Android software aging, which uses existing tools to monitor and detect memory leaks in Android applications. The Monkey tool [16] was used to create a stress load to accelerate software aging. The results of [15] indicate the effectiveness of this approach and allow for detecting the effects of software aging, such as memory leaks.

In [13,17], a detailed and refined experimental methodology for analyzing software aging in the Android OS was presented. The methodology uses statistical methods to determine which factors (workloads and device configurations) exacerbate performance and resource degradation. In addition, the methodology analyzes the relationship between software aging and resource utilization indicators to determine which subsystems affect aging and ensure the development of software rejuvenation strategies.

Domenico Cotroneo et al., in 2016 [13], identified two groups of metrics that allow generalizing metrics to study aging in general and for Android: user-perceived metrics and system metrics.

In addition to studying Android aging, as well as its metrics and factors, there are various approaches to reduce the effects of aging and the use of software rejuvenation techniques. Thus, [18] proposed and implemented an aging detection and rejuvenation tool for Android (ADARTA), which monitors system processes and trends of increasing load on the system, determines the state of aging, evaluates time to aging failure (TTAF), and plans and performs software rejuvenation by restarting basic system services.

In addition to performing the software rejuvenation procedure, anti-aging mechanisms within the applications are studied by reducing the workload and avoiding aging errors. For example, the algorithm for unloading calculations [19] aims to divide the program into local (calculations are performed on a mobile device) and remote (complex calculations are performed on a remote server) parts to reduce the total execution time of the program locally and thus avoid overload, the accumulation of aging errors, and battery consumption. Such unloading is helpful if remote execution performs better than on-device execution or if

the cost of transmitting data to a remote server is less than the time savings or power usage during local computing. The partial unloading scheme is claimed to effectively reduce program execution time and energy consumption [19].

A CTMC Android aging and rejuvenation model was built in [20], which considers the user's phone activity to be able to predict user behavior at different times during the day. Using SPN and the Oris tool [21], a CTMC model combined a user activity model and an aging model with software rejuvenation. Thanks to the obtained CTMC model, it is possible to perform a preliminary assessment of rejuvenation strategies considering the device's intensity of use and the Android aging state.

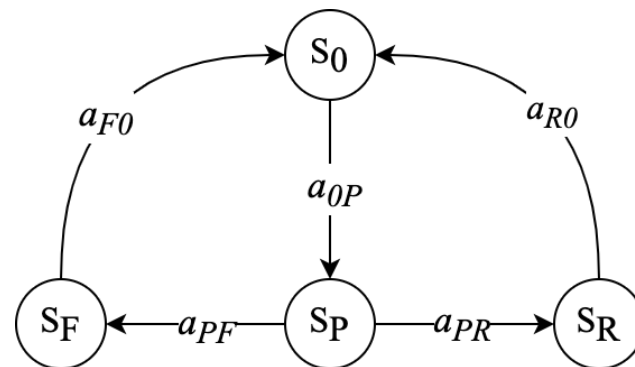


Figure 1. Graph of states and transitions of the primary software aging and rejuvenation model [22].

Thus, there is a constant need for methods and tools to eliminate the effects of aging in mobile systems, including Android, based on an empirical study of the aging process, including the study of metrics and aging factors, creating more reliable and adequate aging and rejuvenation models. This work aims to increase the level of reliability of mobile systems' software by identifying factors influencing the aging process and developing software aging and rejuvenation models considering battery saving, which is crucial for mobile devices such as smartphones, tablets, or embedded Android-based devices.

3. Materials and Methods

The aging process is characterized by a gradual deterioration in productivity, which leads to the failure of aging. Given this factor in the construction of models, it is essential to distinguish between the system's aging levels, which will allow for the building of different software rejuvenation strategies and apply appropriate rejuvenation mechanisms for a particular level. For example, with a critically low level of performance and a high probability of aging failure, restarting a mobile device is needed; with a moderate level of performance, restarting individual applications or services can be performed, and if there is no aging, there is no need for rejuvenation. In addition, considering the aging level allows us to take advantage of detecting aging methods based on measurements and metric thresholds, which provides more accurate information about the system's state under specific usage conditions.

It is proposed to extend the aging model [17,20] with early rejuvenation so that the rejuvenation procedure is not performed in the young state of the system. In this case, the general model of aging and rejuvenation with different levels of aging can be represented by a set of the five possible states of the system: "Young", "Aging", "Old", and "Rejuvenation", and "Failure". The "Young", "Aging," and "Old" states reflect the gradual deterioration of system performance in the context of aging software. Thus, the general extended model can be represented as a graph (Figure 2), where its states have the following characteristics:

- "Young" is a system state characterized by a high level of productivity and a low level of utilization of system resources; for example, the measured aging metrics do not exceed certain thresholds (see Section 4).

- “Aging” is a state of the system where there is a deterioration in performance and increased use of system resources. Still, this process does not yet significantly impact the user experience, and the measured aging metrics are within the threshold values of the transition to “Old”.
- “Old” is a state of the system that occurs when the user experiences significant delays in the graphical interface; there is a depletion of system resources, which can lead to a failure of user applications.
- “Rejuvenation” is a state of the system in which the rejuvenation procedure is performed. The system may be temporarily unavailable when performing a “cold” rejuvenation procedure.
- “Failure” is a state of failure due to aging in which the system is restored to the “Young” state. Recovery, in this case, can be achieved by rebooting the mobile device manually or automatically.

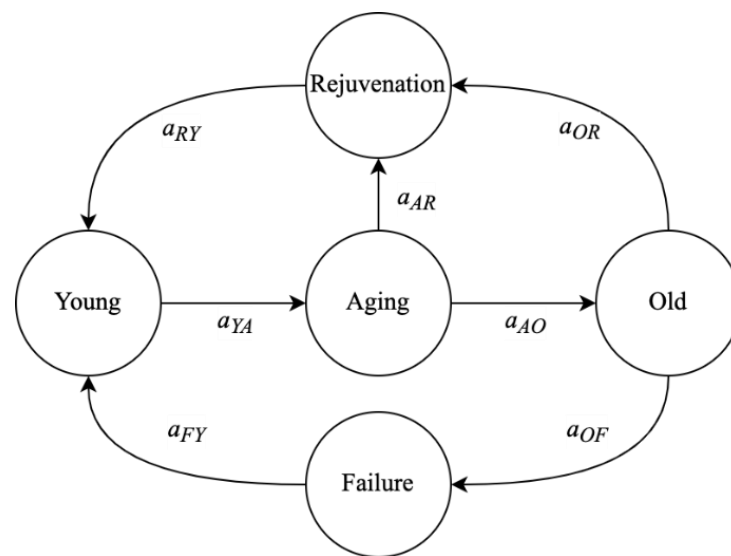


Figure 2. Graph of states and transitions of the general aging and rejuvenation model, which considers different levels of aging.

To calculate the transition intensities a_{ij} , as well as the time of transitions between states T_{ij} , it is important to consider the presence or absence of a constant trend of deteriorating metrics in states such as “Young,” “Aging,” and “Old”. For example, the system may not show an increase in frame draw time (FDT) trends in the “Young” or “Aging” states, so the transition to the next “Aging” and “Old” states may not occur. In this case, implementing the rejuvenation procedure and the need to predict it may be impractical.

Thus, in contrast to the original model [20], the additional “Aging” state allows us to clearly identify the aging process in the system, which can lead to the “Old” state. In this case, the extended model allows rejuvenation in “Aging” and “Old”, provided by transitions with the intensities a_{AR} and a_{OR} , respectively.

Combining the model of aging and rejuvenation, which considers different levels of aging, with the model of mobile device activity, the user can obtain an improved Android software aging model [22], which is presented as a graph of states and transitions in Figure 3. Possible states of the model system, which considers the different levels of aging and user activity, are as follows:

- AmY is a state of active device usage with a high-performance level.
- AmA is a state of active device usage in which the aging process is observed.
- AmO is a state of active device usage with a low-performance level.
- AmR is the state of the rejuvenation procedure during active device usage.
- AmF is the state of failure due to aging during active device usage.
- SmY is the standby state of a device with a high performance level.

- SmA is the state of waiting of a device in which the aging process is observed.
- SmO is the standby state of a device with a low performance level.
- SmR is the state of the rejuvenation procedure while waiting for the device.
- SmF is the state of aging failure while waiting for the device.

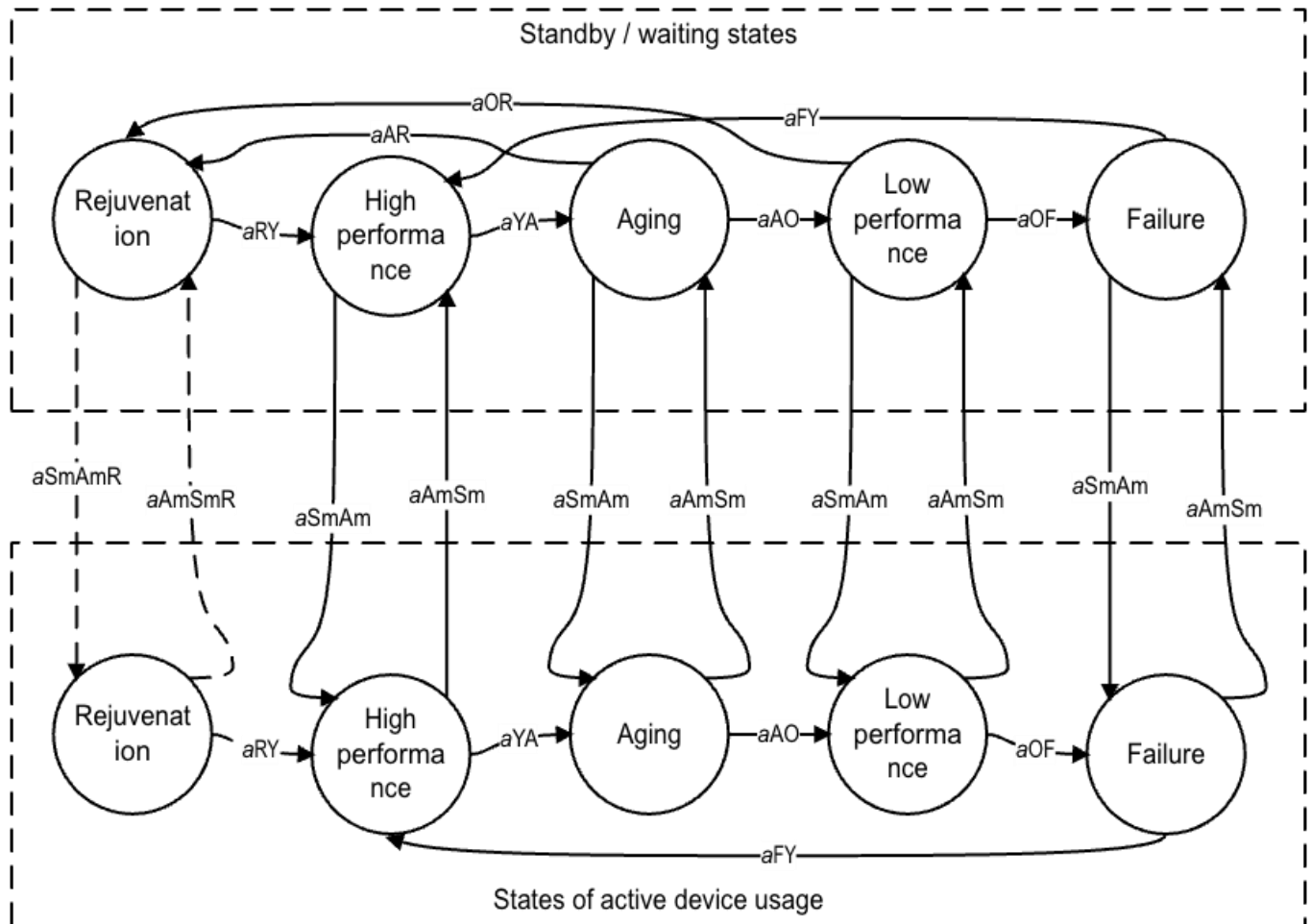


Figure 3. Graph of states and transitions of the aging and rejuvenation model, which considers different levels of aging and user activity of the mobile device [22].

Aging effects can be determined by monitoring and analytically modeling resource consumption at runtime. Furthermore, aging-related metrics (see Section 4 of this paper) can be used to determine the system's aging at a particular time, viz., if a system in a high-performing state or aging process is observed [23].

Software rejuvenation planning involves choosing the optimal time to run it. From the user's point of view, the optimal time is those states of the system when the mobile device is not actively used, namely, S_mY , S_mA , and S_mO . In turn, the optimal time from the point of view of the mobile device is the state of the system when there is a deterioration in productivity; the rejuvenation procedure is performed before aging failure, and rejuvenation does not interrupt high-priority processes, i.e., in S_mA and S_mO states. Thus, the optimal rejuvenation time for the user and the mobile device is the S_mA and S_mO states.

The described model allows us to consider three main strategies for rejuvenation planning, namely [22]:

1. Simultaneous rejuvenation from the "Aging" and "Old" states, i.e., when $a_{AR} > 0$ and $a_{OR} > 0$.
2. Rejuvenation only from the "Aging" state, i.e., when $a_{AR} > 0$ and $a_{OR} = 0$.
3. Rejuvenation only from the "Old" state, i.e., when $a_{AR} = 0$ and $a_{OR} > 0$.

The rejuvenation procedure can be divided into “cold” and “warm” procedures [24]. “Cold” rejuvenation occurs by rebooting the entire device, which gives the best result of improving system performance but takes the most time and can interrupt the user’s work. In turn, rejuvenation by restarting the application (“warm” rejuvenation) takes the least time, but the effectiveness of improving performance is not as high, and, also, it is possible to interfere with the user’s activities or the performance of important tasks. Given the existence of different mechanisms of rejuvenation, namely “cold” and “warm” rejuvenation, the model introduced the transition intensities between the states of A_mR and S_mR . These parameters allow for determining the type of rejuvenation procedure. When $a_{SmAmR} = 0$ and $a_{AmSmR} = 0$, “cold” rejuvenation is performed, i.e., the system is idle. Thus, considering the user activity, an important task is to choose the optimal rejuvenation mechanism that will ensure a high level of system performance.

4. Results

4.1. Model of Aging and Rejuvenation of Software Considering the Level of Battery Charge

A feature of mobile devices that distinguishes them from other systems is their dependence on a battery charge. This factor is essential for the user and for planning and implementing software rejuvenation. The user needs to increase the device’s operating time from a battery charge, which can be achieved by reducing the load on the system’s resources, including a rejuvenation procedure. In turn, to rejuvenate the software, it is essential to consider the possibility of its implementation at the scheduled time and the effectiveness of this procedure with a long-term effect. Without considering the battery charge, the planned rejuvenation of the software may not be performed when the device is completely discharged. It is also worth noting that rejuvenating the software in a low-charge device may not have a long-term effect. On the contrary, it increases the system’s downtime, negatively affecting the user experience. Thus, in the model of aging and rejuvenation, it is proposed to consider the battery charge factor [25], which would avoid rejuvenation in a state with a low charge or fully discharged device.

The possible states of the system in the model [25] are:

- Active Mobile + Stable Power + Young—the user actively uses a mobile device with a high charge level and performance.
- Active Mobile + Stable Power + Old—the user actively uses a mobile device with a high charge and a low-performance level, likely leading to an aging failure.
- Active Mobile + Stable Power + Failure—aging failure state during active use of a mobile device with a high battery, such that the user is forced to reboot the mobile device.
- Active Mobile + Stable Power + Rejuvenation—the rejuvenation procedure is performed during active mobile device usage with a high battery charge.
- Sleep Mobile + Stable Power + Young—the mobile device is in standby mode with high charge and performance.
- Sleep Mobile + Stable Power + Old—a mobile device is in standby mode with a high charge and low performance level, likely leading to an aging failure in the future.
- Sleep Mobile + Stable Power + Failure—state of an aging failure in the standby mode of the mobile device with a high battery charge; the system automatically reboots the mobile device.
- Sleep Mobile + Stable Power + Rejuvenation—the rejuvenation procedure is performed in the standby mode of a mobile device with a high battery charge.
- Active Mobile + Low Power + Young—the user actively uses a mobile device with low charge and high performance.
- Active Mobile + Low Power + Old—the user actively uses a mobile device with a low charge and low-performance level, likely leading to an aging failure.
- Active Mobile + Low Power + Failure—the state of an aging failure during the active use of a mobile device with a low battery; the user is forced to restart the mobile device.
- Active Mobile + Low Power + Rejuvenation—the rejuvenation procedure when the user actively uses a mobile device with a low battery.

- Sleep Mobile + Low Power + Young—a mobile device with low charge and high productivity is in standby mode.
- Sleep Mobile + Low Power + Rejuvenation—the rejuvenation procedure is performed in the standby mode of a mobile device with a low battery.
- Sleep Mobile + Low Power + Failure—the state of an aging failure in standby mode of a mobile device with a low battery, the system automatically reboots the mobile device.
- Sleep Mobile + Low Power + Old—a mobile device is in standby mode with a low charge and low-performance level, likely leading to an aging failure.
- Off Power—the state of complete battery discharge.

A graphic model of aging and rejuvenation, considering the activity of the mobile device and the battery factor, is presented in Figure 4 [25]. An essential feature of this model is that software rejuvenation can be performed only in the “Sleep”, “Stable Power”, and “Young” states. This approach simulates the aging and rejuvenation process, where rejuvenation does not interrupt the user’s active use of the device and precedes the transition of the battery to a low charge state or complete shutdown.

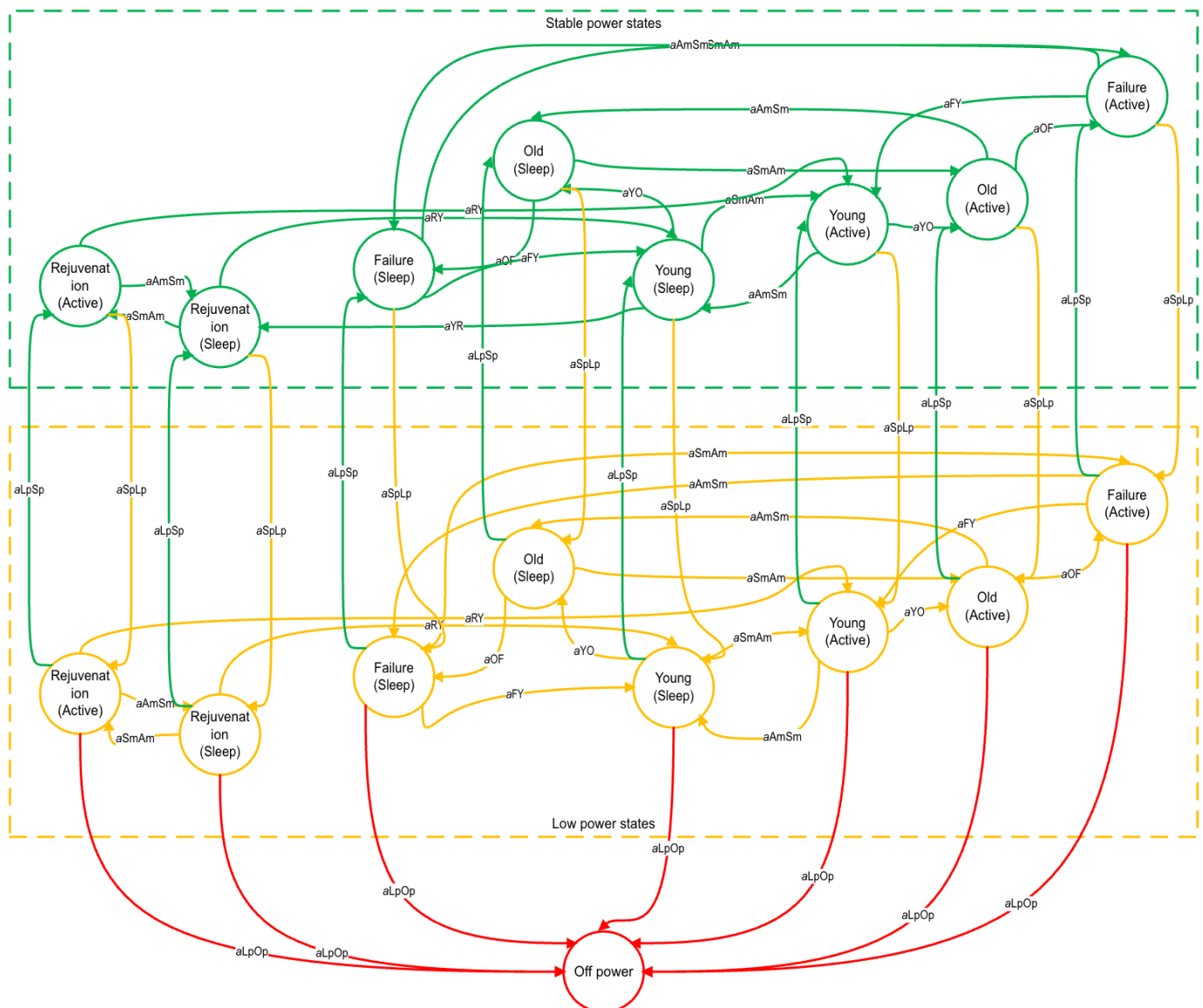


Figure 4. Graph of states and transitions of the software aging and rejuvenation model considering the battery charge level [25].

By solving the system of differential equations [25] for different a_{YR} values, the optimal rejuvenation time can be determined when the probability of the system being in the “Active” and “Low Power” states is the lowest, i.e., the rejuvenation procedure is least likely to be performed when using a mobile device and will be ahead of battery discharge.

For verification and analysis of the model considering the battery charge, the solutions for the system of differential equations for the graph depicted in Figure 4 and the original model [20] were obtained. A comparison of $P_{AmR}(t) + P_{AmF}(t)$ and $P_{AmLpR}(t) + P_{AmLpF}(t)$ values for different $T_{YR}(1/a_{YR})$ for both models allows for analyzing the influence of the battery charge factor on the determination of the optimal rejuvenation time.

The transition intensities between the states were selected to compare the simulation results with those presented in [20]. This paper uses one minute as a unit of time, and user activity is determined for one day. The following parameters used in simulations are typical for average mobile device use and obviously may vary in practice for different scenarios. However, this does not limit the conclusions of the paper to the studied set of parameters. The a_{AS} and a_{SA} transition intensities are assumed to be 0.1 min^{-1} (the average time to use the mobile device before entering the standby mode is 10 min) and 0.017 min^{-1} (the average time spent in the standby mode is 60 min), respectively. Transitions from the “Young” state to the “Old” state, as well as from the “Old” state to the “Failure” state, take 24 h, so $a_{YO} = a_{OF} = 0.00069 \text{ min}^{-1}$. The rejuvenation and recovery after aging failure are considered the process of rebooting the mobile device, which takes about two minutes, so $a_{FY} = a_{RY} = 0.5 \text{ min}^{-1}$. In the case of the battery-based model, it is assumed that the battery discharge time is 15 h, and then the time to complete shutdown is one hour, i.e., $a_{SpLp} = 0.0010 \text{ min}^{-1}$ and $a_{LpOp} = 0.017 \text{ min}^{-1}$. The time required to restore the battery charge from low to stable is 3 h, i.e., $a_{LpSp} = 0.0056 \text{ min}^{-1}$.

Simulations of the aging and rejuvenation process were performed for different T_{YR} values both for the original model (Figure 5) and for the model considering the battery charge level (Figure 6). Since the aging and rejuvenation model considers battery charge and mobile device usage, the optimal rejuvenation time is when the probability of the system being in active use and having a low charge is lowest, i.e., $\min(P_{AmLpR}(t) + P_{AmLpF}(t))$.

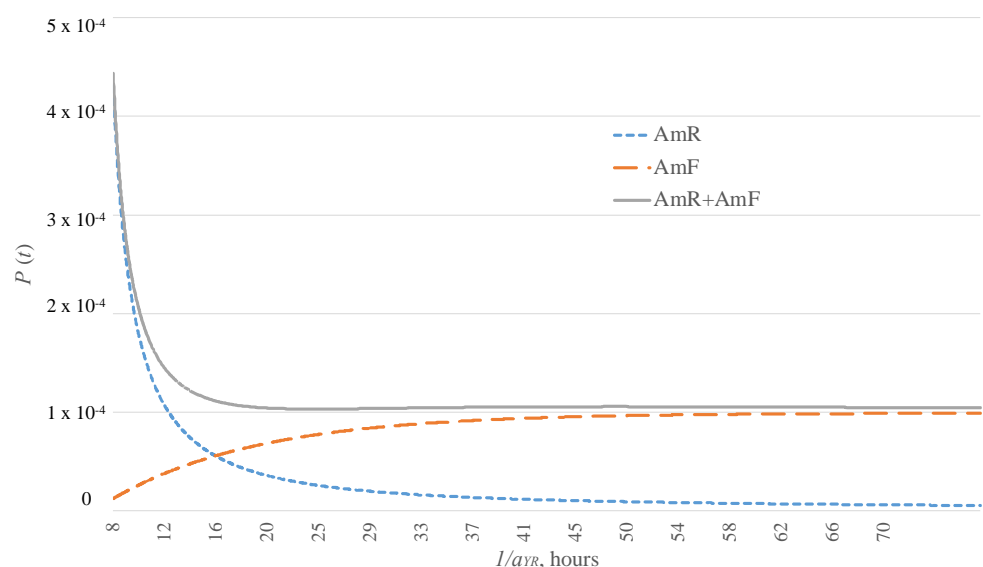


Figure 5. Results of the aging and rejuvenation process simulation using the original model not considering battery charge.

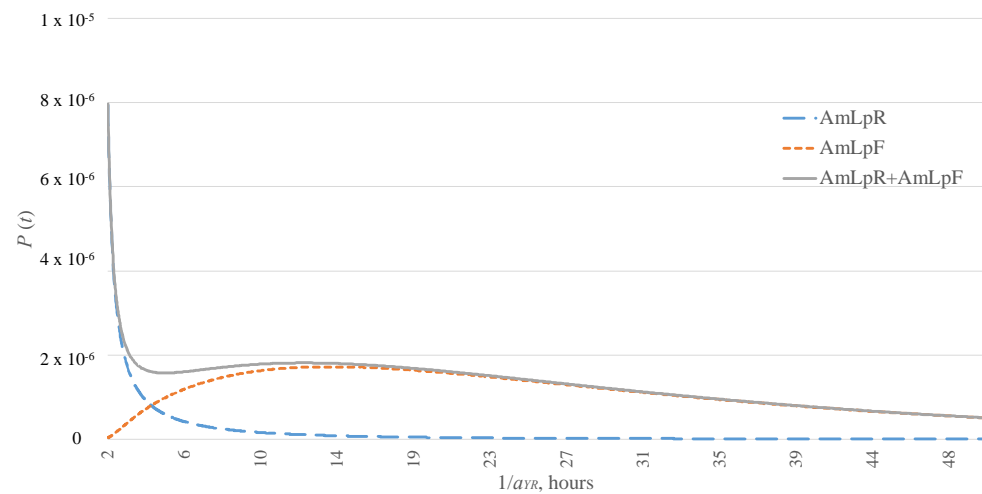


Figure 6. Results of simulation of aging and rejuvenation process using the battery-based model.

The model's simulation results without considering the battery charge (Figure 5) show that the optimal rejuvenation time can be estimated to be $1/a_{YR} = 840$ min (14 h). The lowest point of the curve AmR + AmF on the graph indicates that at the rejuvenation intensity $a_{YR} = 0.00119 \text{ min}^{-1}$, the lowest probability of interrupting the active use of the system by rejuvenation or aging failure can be ensured.

The model considering the battery charge (Figure 6) for the same aging conditions and mobile device activity shows an optimal rejuvenation time of about 240 min (4 h), which is 3.5 times less than the time estimated without considering the battery charge. Using the lowest point of the curve, AmLpR + AmLpF, the rejuvenation intensity can be determined: $a_{YR} = 0.00417 \text{ min}^{-1}$. A comparison of the obtained results shows that considering the battery charge has a significant impact on planning the time of software rejuvenation. Thus, if one schedules the software rejuvenation procedure without considering the battery state, the estimated starting time of this procedure will be significantly longer. However, during this time, the device's battery will most likely enter a low charge state, and performing the rejuvenation procedure will further discharge it, which will have a negative impact on the user experience.

The idea of considering the battery charge is that performing a rejuvenation procedure is impractical or impossible at a time when the mobile device has a critically low charge or is completely discharged. The obtained simulation results confirm this hypothesis.

4.2. Comprehensive Model of Aging and Rejuvenation for Android Operating System

To develop methods of software rejuvenation, it is essential to formulate and describe a comprehensive model that considers all the advantages of the previous two models. This paper presents a comprehensive model (Figure 7) of aging and rejuvenation based on CTMC, which assumes different levels of aging [22], the activity of the user using a mobile device [20], and battery charge [25]. The possible states of the system in this model are:

- Active Mobile + Stable Power + Young (AmSpY)—the user actively uses a mobile device with a high charge level and performance.
- Active Mobile + Stable Power + Aging (AmSpA)—the user actively uses a mobile device with a high charge level with a performance deterioration, but aging failure is unlikely.
- Active Mobile + Stable Power + Old (AmSpO)—the user actively uses a mobile device with a high charge level with a low performance level, likely leading to an aging failure.
- Active Mobile + Stable Power + Failure (AmSpF)—aging failure state during active use of a mobile device with a high battery; the user is forced to reboot the mobile device.
- Active Mobile + Stable Power + Rejuvenation (AmSpR)—the rejuvenation procedure is performed during active mobile device usage with a high battery charge.

- Sleep Mobile + Stable Power + Young (SmSpY)—the mobile device is in standby mode with high charge and performance.
- Sleep Mobile + Stable Power + Aging (SmSpA)—the mobile device is in standby mode with high charge and with performance deterioration, but aging failure is unlikely.
- Sleep Mobile + Stable Power + Old (SmSpO)—the mobile device is in standby mode with a high charge and low performance level, likely leading to aging failure.
- Sleep Mobile + Stable Power + Failure (SmSpF)—state of an aging failure in the standby mode of the mobile device with a high battery charge; the system automatically reboots the mobile device.
- Sleep Mobile + Stable Power + Rejuvenation (SmSpR)—the rejuvenation procedure is performed in the standby mode of a mobile device with a high battery charge.
- Active Mobile + Low Power + Young (AmLpY)—the user actively uses a mobile device with low charge and high performance.
- Active Mobile + Low Power + Aging (AmLpA)—the user actively uses a mobile device with a low charge and performance deterioration, but aging failure is unlikely.
- Active Mobile + Low Power + Old (AmLpO)—the user actively uses a mobile device with a low charge and low performance level, likely leading to aging failure.
- Active Mobile + Low Power + Failure (AmLpF)—the state of an aging failure during active use of a mobile device with a low battery; the user is forced to restart the mobile device.
- Active Mobile + Low Power + Rejuvenation (AmLpR) is the rejuvenation procedure that occurs when the user actively uses a mobile device with a low battery.
- Sleep Mobile + Low Power + Aging (SmLpA)—the mobile device is in standby mode with a low charge and performance deterioration, but aging failure is unlikely.
- Sleep Mobile + Low Power + Young (SmLpY)—a mobile device with low charge and high productivity is in standby mode.
- Sleep Mobile + Low Power + Rejuvenation (SmLpR)—the rejuvenation procedure is performed in the standby mode of a mobile device with a low battery.
- Off Power (Op)—the state of complete battery discharge.
- Sleep Mobile + Low Power + Failure (SmLpF)—the state of an aging failure in the standby mode of a mobile device with a low battery; the system automatically reboots the mobile device.
- Sleep Mobile + Low Power + Old (SmLpO)—a mobile device in standby mode with a low charge and low performance level, likely leading to aging failure.

The analytical representation of the model in the form of a system of Kolmogorov–Chapman differential equations is as follows:

$$\begin{aligned}
 \frac{dP_{AmSpY}(t)}{dt} &= -(a_{AmSm} + a_{SpLp} + a_{YA})P_{AmSpY}(t) + a_{SmAm}P_{SmSpY}(t) + a_{LpSp}P_{AmLpY}(t) + a_{FY}P_{AmSpF}(t) + a_{RY}P_{AmSpR}(t); \\
 \frac{dP_{AmSpA}(t)}{dt} &= -(a_{AmSm} + a_{SpLp} + a_{AO})P_{AmSpA}(t) + a_{SmAm}P_{SmSpA}(t) + a_{LpSp}P_{AmLpA}(t) + a_{YA}P_{AmSpY}(t); \\
 \frac{dP_{AmSpO}(t)}{dt} &= -(a_{AmSm} + a_{SpLp} + a_{OF})P_{AmSpO}(t) + a_{SmAm}P_{SmSpO}(t) + a_{LpSp}P_{AmLpO}(t) + a_{AO}P_{AmSpA}(t); \\
 \frac{dP_{AmSpF}(t)}{dt} &= -(a_{AmSm} + a_{SpLp} + a_{FY})P_{AmSpF}(t) + a_{SmAm}P_{SmSpF}(t) + a_{LpSp}P_{AmLpF}(t) + a_{OF}P_{AmSpO}(t); \\
 \frac{dP_{AmLpY}(t)}{dt} &= -(a_{AmSm} + a_{LpSp} + a_{YA} + a_{LpOp})P_{AmLpY}(t) + a_{SmAm}P_{SmLpY}(t) + a_{SpLp}P_{AmSpY}(t) + a_{FY}P_{AmLpF}(t) + a_{RY}P_{AmLpR}(t); \\
 \frac{dP_{AmLpA}(t)}{dt} &= -(a_{AmSm} + a_{LpSp} + a_{AO} + a_{LpOp})P_{AmLpA}(t) + a_{SmAm}P_{SmLpA}(t) + a_{SpLp}P_{AmSpA}(t) + a_{YA}P_{AmLpY}(t); \\
 &\dots \\
 \frac{dP_{Op}(t)}{dt} &= a_{LpOp} (P_{AmLpR}(t) + P_{SmLpR}(t) + P_{SmLpF}(t) + P_{SmLpO}(t) + P_{SmLpA}(t) \\
 &\quad + P_{SmLpY}(t) + P_{SmLpA}(t) + P_{SmLpY}(t) + P_{AmLpA}(t) + P_{AmLpO}(t) \\
 &\quad + P_{AmLpF}(t)).
 \end{aligned} \tag{1}$$

where $P_i(t)$ is the probability of the system being in the i -th state (AmSpY, AmSpA, AmSpO, SmSpY, SmSpA, SmSpO, AmLpY, AmLpA, AmLpO, SmLpY, SmLpA, SmLpO, AmSp, SmSp, AmSp, SmSp, AmSp, AmLpF, AmLpR, SmLpF, SmLpR, and Op, correspondingly), and a_{ij} is the transition intensity from state i to state j .

The developed model can conclude that the rejuvenation procedure should be provided in the “Sleep”, “Stable Power”, and “Aging” states. The rejuvenation procedure in the SmSpA state will not have a significant negative impact on the user experience. It will also be ahead of the device’s transition from a vulnerable to aging failure or battery discharge

state. The SmSpA stat's transition describes the rejuvenation procedure's execution in the comprehensive model to the SmSpR state with the intensity a_{AR} . Thus, the estimation of the optimal rejuvenation time is to find such a value of a_{AR} that results in the system most likely being in the SmSpA state at the time of the procedure. To find the optimal time for rejuvenation, it is necessary to provide the following conditions:

$$\begin{cases} \sum_{i \in S_A} P_i(t) \rightarrow \max \\ \sum_{j \in S_{FR}} P_j(t) \rightarrow \min \\ P_{SmSpA} \rightarrow \max \end{cases} \quad (2)$$

where S_A is the set of operating states of the system (AmSpY, AmSpA, AmSpO, SmSpY, SmSpA, SmSpO, AmLpY, AmLpA, AmLpO, SmLpY, SmLpA, and SmLpO), and S_{FR} is the set of inoperable states of the system (AmSp, SmSp, AmSp, SmSp, AmSp, AmLpF, AmLpR, SmLpF, SmLpR, and Op).

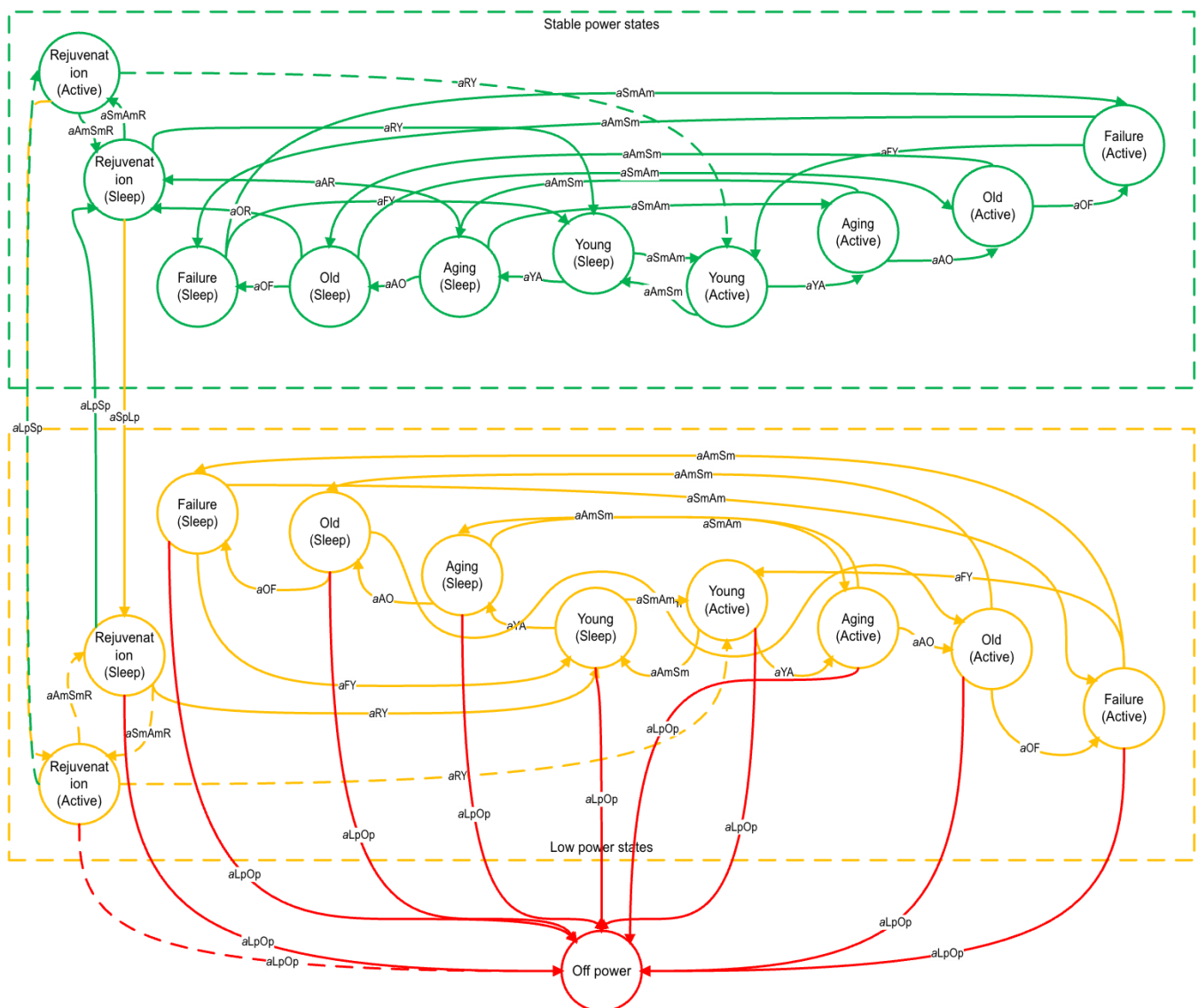


Figure 7. Graph of states and transitions of the comprehensive Android software aging and rejuvenation model.

5. Discussion

To apply an extended aging and rejuvenation model for assessing the rejuvenation time, an important task is to calculate the intensities of transitions between states and determine the system's condition at a particular time. The following model of aging factors is proposed to solve this problem, which sets of aging metrics and metrics for the corresponding aging factors can represent. The aging factor model considers the known aging metrics ALT, PSS, and GCT and the metrics proposed in our previous study [26,27]. The following sets of metrics were selected:

- Aging metrics: Activity launch time (ALT), frame draw time (FDT), junky frames ratio (JFR), proportional set size (PSS), garbage collection time (GCT).
- Metrics of mobile device activity: ALT, FDT, user process ratio, CPU usage ratio, PSS sum ratio.
- Battery charge status metrics: current battery charge, discharge estimated time.

Activity launch time is the time needed to launch an activity in the Android OS [28]. Frame draw time is the time needed to draw a frame. It is necessary that FDT does not exceed 16 ms to allow frame refresh rates above 60 fps, which is a typically convenient value. The ratio of missed or delayed frames that were not displayed to the total number of frames that should have been displayed is the junky frames ratio.

Tables 1–3 show the conditions and estimated values of the metrics at which it is possible to stay in one state or another. Future work will be devoted to the detailed study of the proposed thresholds and the effectiveness of their use in potential rejuvenation methods.

Table 1. Metrics and their thresholds for determining the aging level of the system.

Metric	Priority	State of the System			
		Young	Aging	Old	Failure
FDT	High	≤ 16 ms	>16 ms & ≤ 40 ms	>40 ms & ≤ 66 ms	>66 ms
PSS	High	$\leq 20\%$	$>20\%$ & $\leq 80\%$	$>80\%$ & $\leq 95\%$	$>95\%$
JFR	Medium	$\leq 20\%$	$>20\%$ & $\leq 80\%$	$>80\%$ & $\leq 95\%$	$>95\%$
ALT	Medium	<2 s	≥ 2 s & <5 s	≥ 5 s & <10 s	≥ 10 s
GCT	Medium	≤ 100 ms	>100 ms & ≤ 200 ms	>200 ms & ≤ 300 ms	>300 ms

Table 2. Metrics and their thresholds for determining the usage activity level of mobile devices.

Metric	Weight	Mobile Device Usage Activity	
		Sleep	Active
FDT	1.0	0	>0
ALT	0.5	0	>0
User processes ratio	0.5	$<20\%$	$\geq 20\%$
CPU usage ratio	0.25	$<20\%$	$\geq 20\%$
PSS sum ratio	0.25	$<20\%$	$\geq 20\%$

Table 3. Metrics and their thresholds for determining the battery charge status.

Metric	Weight	Battery Charge Status		
		High Power	Low Power	Off Power
Discharge estimated time	1.0	>1 h	≤ 1 h	0
Current battery charge	0.25	$>20\%$	$\leq 20\%$	0%

To ensure smooth GUI operation, it is necessary to display 60 frames per second ($FDT \leq 16$ ms), which is a condition for the system to be in the “Young” state. The aging phenomenon can be considered if the number of frames per second is less than 15 ($FDT > 66$ ms). PSS metric thresholds are defined relative to the total amount of available RAM. ALT metric thresholds are determined based on the official documentation [28],

where “hot” (ALT is below 1.5 s) and “warm” (ALT is below 2 s) starts can characterize the “Young” state; “cold” starts (ALT is below 5 s) can be ascribed to the “Aging” state; and an ALT value of more than 5 and 10 s characterizes the “Old” and “Failure” states, respectively. In the case of GCT, the lower value of 100 ms indicates a delay.

To calculate the intensities of transitions between aging states (Table 1), the methods to identify the existing positive trend for each time series can be used. The time series describes the behavior of the mobile device from the moment it is on to the moment it turns off. To identify possible trends, the Mann–Kendall trend test can be applied. If the trend exists, the intersection of the trend line with the threshold values gives the time limits for each state. The metric priorities are necessary to rule out cases where aging is observed only for low-priority metrics. For example, suppose the aging trend is observed in only one of the metrics with an average level. In that case, this can be rejected and not considered in the calculations of intensities.

In the case of determining the activity level of mobile devices (Table 2), FDT and ALT metrics can be indicators of the “Active” state at values greater than 0, and the user processes ratio, CPU usage ratio, and PSS sum ratio are greater than 20%.

The threshold values of battery charge metrics (see Table 3) will be the subject of our future work. They can be measured as absolute (estimated time to full discharge) and relative (battery charge percentage) values. Both approaches have their own limitations, could depend on the type of mobile device and the condition of its battery, and need further research.

In the case of activity metrics and thresholds (Table 2) and the battery charge status (Table 3), the calculation of the intensities should consider the frequency of changes in their states throughout the mobile device life cycle. The metric weights allow the state to select the largest sum of the weights. For example, if the measured FDT metric indicates the “Active” state while the rest indicates the “Sleep” state, then the latter should be selected because the sum of the weights for this state is 1.5 and is greater than the FDT weight. This example may describe a case where the user activated the phone screen, and the device displayed a home screen of an application, but no calculations take place. It is possible that the user finished using the phone. If the user continues to use it, the usage of system resources will increase, which will be reflected in the measured metrics.

To determine the current aging level, it is necessary to calculate the average value for the last N records of the metrics time series and determine the interval used in Table 1. The current metric values should be used to determine the current battery status and user activity.

6. Conclusions

The implementation of a software rejuvenation procedure is a recognized cost-effective tool to combat software aging effects. Typical systems that use software rejuvenation methods are transaction processing systems, web servers, spacecraft systems, etc. A real example can be the method of Apache web server rejuvenation, which terminates and rebuilds processes after a certain number of requests have been executed [29]. Another approach is to reboot virtual machines running in a cloud computing environment [30]. The telecommunications corporation AT&T implemented real-time billing software rejuvenation [31].

This paper summarizes the previously developed Android software aging and rejuvenation models and presents a comprehensive model of aging and rejuvenation for the Android operating system. This comprehensive model is based on continuous-time Markov chains and considers different aging levels, mobile device activity, battery status, “warm” and “cold” rejuvenation mechanisms, and possible strategies for rejuvenation.

The comprehensive model of aging and rejuvenation developed in this paper is an effective tool for designing and selecting the parameters of the software rejuvenation method. One of the consequences of software aging is aging-related failure. Software failure is an important part of reliability, and according to ISO/IEC 25010:2011, of software quality. The proposed models of aging and rejuvenation can be used to enhance software

quality in terms of decreasing the probability of aging-related failures. The comprehensive model allows obtaining expressions for indicators of software rejuvenation efficiency. For the proposed comprehensive aging and rejuvenation model, the conditions for finding the optimal rejuvenation time are determined, considering the system's sets of operational and inoperable states. The inoperable states include both failure and downtime during rejuvenation.

A model of the software aging factors for the Android operating system has been developed. The model includes three sets of metrics, viz., aging metrics (ALT, FDT, JFR, PSS, and GCT); metrics of mobile device activity (ALT, FDT, user process ratio, CPU usage ratio, and PSS sum ratio); and battery charge status metrics (current battery charge, discharge estimated time). The proposed thresholds of each metric allow for determining the current aging level and can be used for software rejuvenation time estimation. The software rejuvenation procedure, based on a comprehensive model of aging and rejuvenation, considers the mobile device's battery status and activity, improving the user experience.

Author Contributions: Conceptualization, V.Y. and B.U.; methodology, V.Y.; software, B.U.; validation, B.U. and N.S.; formal analysis, V.Y. and N.S.; data curation, B.U.; writing—original draft preparation, V.Y. and B.U.; writing—review and editing, N.S.; visualization, B.U.; supervision, V.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Research Foundation of Ukraine, grant number 2021.01/0103. Vitaliy Yakovyna and Natalya Shakhovska also thank the U4U Non-Residential Fellowship Program for financially supporting this research.

Data Availability Statement: Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

- Huang, Y.; Kintala, C.; Kolettis, N.; Fulton, N.D. Software rejuvenation: Analysis, module and applications. In Proceedings of the 25th Symposium on Fault Tolerant Computing, Pasadena, CA, USA, 27–30 June 1995. [\[CrossRef\]](#)
- Garg, S.; Puliafito, A.; Telek, M.; Trivedi, K. Analysis of preventive maintenance in transactions based software systems. *IEEE Trans. Comput.* **1998**, *47*, 96–107. [\[CrossRef\]](#)
- Bao, Y.; Sun, X.; Trivedi, K.S. A workload-based analysis of software aging and rejuvenation. *IEEE Trans. Reliab.* **2005**, *55*, 541–548. [\[CrossRef\]](#)
- Garg, S.; Puliafito, A.; Telek, M.; Trivedi, K.S. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In Proceedings of the Sixth International Symposium on Software Reliability Engineering, Toulouse, France, 24–27 October 1995. [\[CrossRef\]](#)
- Wang, D.; Xie, W.; Trivedi, K.S. Performability analysis of clustered systems with rejuvenation under varying workload. *Perform. Eval.* **2007**, *64*, 247–265. [\[CrossRef\]](#)
- Pfening, A.; Garg, S.; Puliafito, A.; Telek, M.; Trivedi, K.S. Optimal software rejuvenation for tolerating soft failures. *Perform. Eval.* **1996**, *27/28*, 491–506. [\[CrossRef\]](#)
- Okamura, H.; Dohi, T. A POMDP formulation of multistep failure model with software rejuvenation. In Proceedings of the IEEE Third International Workshop on Software Aging and Rejuvenation, Hiroshima, Japan, 29 November–2 December 2011. [\[CrossRef\]](#)
- Dohi, T.; Trivedi, K.S.; Avritzer, A. *Handbook of Software Aging and Rejuvenation*; World Scientific Publishing Co. Pte Ltd.: Singapore, 2020.
- Adams, E. Optimizing preventive service of software products. *IBM J. Res. Dev.* **1984**, *28*, 2–14. [\[CrossRef\]](#)
- Parnas, D.L. Software aging. In Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italia, 16–21 May 1994. [\[CrossRef\]](#)
- Bernstein, L.; Kintala, C.M.R. Software Rejuvenation. *CrossTalk* **2004**, *6*, 23–26.
- Cotroneo, D.; Natella, R.; Pietrantuono, R.; Russo, S. A Survey of Software Aging and Rejuvenation Studies. *ACM J. Emerg. Technol. Comput. Syst.* **2014**, *10*, 8. [\[CrossRef\]](#)
- Qiao, Y.; Zheng, Z.; Qin, F. An empirical study of software aging manifestation in android. In Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW), Ottawa, ON, Canada, 23–27 October 2016. [\[CrossRef\]](#)

14. Araujo, J.; Alves, V.; Oliveira, D.; Dias, P.; Silva, B.; Maciel, P. An Investigative Approach to Software Aging in Android Applications. In Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics, Manchester, UK, 13–16 October 2013. [CrossRef]
15. Cotroneo, D.; Iannillo, A.K.; Natella, R.; Pietrantuono, R. A Comprehensive Study on Software Aging across Android Versions and Vendors. *Empir. Softw. Eng.* **2020**, *25*, 3357–3395. [CrossRef]
16. UI/Application Exerciser. Available online: <https://developer.android.com/studio/test/monkey> (accessed on 13 December 2022).
17. Cotroneo, D.; Fucci, F.; Iannillo, A.K.; Natella, R.; Pietrantuono, R. Software aging analysis of the Android mobile OS. In Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016. [CrossRef]
18. Cotroneo, D.; Simone, L.D.; Natella, R.; Pietrantuono, R.; Russo, S. A Configurable Software Aging Detection and Rejuvenation Agent for Android. In Proceedings of the 11th International Workshop on Software Aging and Rejuvenation (WoSAR), Berlin, Germany, 27–30 October 2019. [CrossRef]
19. Wu, H.; Wolter, K. Software aging in mobile devices: Partial computation offloading as a solution. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Gaithersburg, MD, USA, 2–5 November 2015. [CrossRef]
20. Xiang, J.; Weng, C.; Zhao, D.; Tian, J.; Xiong, S.; Li, L.; Andrzejak, A. A New Software Rejuvenation Model for Android. In Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Memphis, TN, USA, 15–18 October 2018. [CrossRef]
21. Bucci, G.; Carnevali, L.; Ridi, L.; Vicario, E. Oris: A tool for modeling, verification and evaluation of real-time systems. *Int. J. Softw. Tools Technol. Transf.* **2010**, *12*, 391–403. [CrossRef]
22. Yakovyna, V.S.; Uhrynovskyi, B.V. Extended software aging and rejuvenation model for Android operating system considering different aging levels and rejuvenation procedure types. *Comput. Syst. Inf. Technol.* **2021**, *3*, 116–124. [CrossRef]
23. Cotroneo, D.; Natella, R.; Pietrantuono, R. Is Software Aging related to Software Metrics? In Proceedings of the IEEE Second International Workshop on Software Aging and Rejuvenation, 2 November 2010, 29 November–2 December 2011. [CrossRef]
24. Guo, C.; Wu, H.; Hua, X.; Lautnery, D.; Ren, S. Use Two-Level Rejuvenation to Combat Software Aging and Maximize Average Resource Performance. In Proceedings of the IEEE International Conference on High Performance Computing and Communications, New York, NY, USA, 24–26 August 2015. [CrossRef]
25. Yakovyna, V.S.; Uhrynovskyi, B.V. Android software aging and rejuvenation model considering the battery charge. *Radio Electron. Comput. Sci. Control* **2021**, *4*, 140–148. [CrossRef]
26. Yakovyna, V.S.; Uhrynovskyi, B.V. User-Perceived Response Metrics in Android OS for Software Aging detection. In Proceedings of the IEEE 15th International Conference on Computer Sciences and Information Technologies, Zbarazh, Ukraine, 23–26 September 2020. [CrossRef]
27. Yakovyna, V.S.; Uhrynovskyi, B.V. Aging of Native and Flutter Applications in Android OS in Various Usage Scenarios. In Proceedings of the IEEE 16th International Conference on Computer Sciences and Information Technologies, Lviv, Ukraine, 22–25 September 2021. [CrossRef]
28. App Startup Time—Android Developers. Available online: <https://developer.android.com/topic/performance/vitals/launch-time> (accessed on 13 December 2022).
29. Trivedi, K.S.; Vaidyanathan, K. Software Aging and Rejuvenation. In *Wiley Encyclopedia of Computer Science and Engineering*; Wiley: Chichester, UK, 2007. [CrossRef]
30. Bruneo, D.; Distefano, S.; Longo, F.; Puliafito, A.; Scarpa, M. Workload-Based Software Rejuvenation in Cloud Systems. *IEEE Trans. Comput.* **2013**, *62*, 1072–1085. [CrossRef]
31. Trivedi, K.S.; Vaidyanathan, K. Software Rejuvenation—Modeling and Analysis. In *Information Technology. IFIP International Federation for Information Processing*; Reis, R., Ed.; Springer: Boston, MA, USA, 2004; Volume 157, pp. 151–182. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.