Rejuvenation and the Age of Information

Daniel Menasché, Kishor Trivedi, Eitan Altman

Rejuvenation and the Age of Information

Daniel Sadoc Menasché
Department of Computer Science
Federal University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
sadoc@dcc.ufrj.br

Kishor Trivedi
Department of Computer Science
Duke University, United States
ktrivedi@duke.edu

Eitan Altman
Inria, Université Côte D'Azur
Sophia Antipolis, France
eitan.altman@inria.fr

***Abstract*—Two decades after the seminal paper on software aging and rejuvenation appeared in 1995, a new concept and metric referred to as the age of information (AoI) has been gaining attention from practitioners and the research community. In this vision paper, our aim is to show the similarities and differences between software aging and information aging. In particular, modeling frameworks that have been applied to software aging, such as the semi Markov approach can be immediately applied in the realm of age of information. Conversely, we indicate that questions pertaining to sampling costs associated with the age of information can be useful to assess the optimal rejuvenation trigger interval for software systems.**

I. Introduction

The age of information (AoI) is a fundamental concept in networked systems [1], [2], [3]. The AoI is the time elapsed since the generation of the last successfully received message containing update information about its source system. In essence, it captures how old is the information available at each node of a network, and has important implications on the rate at which nodes should retrieve content updates. Such updates, in turn, are at the core of most applications running on mobile systems wherein users have constrained resources (e.g., battery and WiFi coverage).

Software rejuvenation has been studied for over two decades in the realm of computer systems and software [4], [5]. Although there is a clear relationship between AoI and software aging and rejuvenation, previous research has not leveraged the interplay between models developed for those domains.

Software aging manifests as an increased failure rate or decreased performance in long running software systems. Naturally, the software aging leads to either a software failure or a performance slow down. One approach to deal with software aging is called software rejuvenation, where the software state is quickly refreshed. On the other side, the Age of Information is about information freshness. Connections between software aging and the Age of Information (AoI) are the subject of this vision paper:

1) mathematical methods used for software aging, such as semi Markov processes, are applicable to the analysis of age of information. Such models are discussed in Section III,
2) monitoring and discovery are related to the cost of sampling and of getting information. Fresh data has its price. For software aging and rejuvenation, as well as for age of information, one cares about the cost to get fresh data [6]. There is a cost to refresh a software system (and decrease software aging) or to refresh a virtual message (and decrease age of information). In Section IV we study aspects related to monitoring and sampling,
3) a number of software aging types are caused by stale information. We discuss the implications of information aging to software systems in Section V.

In the next section we provide an introduction to the interplay between software aging and age of information mechanisms. The remainder of the paper follows the outline presented in the previous bullets. Then, Section VI presents a case study, Section VII reports related work and Section VIII concludes the paper.

II. Interplay between software aging and age of information

In this section we provide a summary of software aging mechanisms described in the literature and describe the interplay between software aging and age of information mechanisms in real software systems.

A. Fundamentals of software aging

Software aging in long-running software systems is said to occur when the software system shows a significant increase in the failure rate as its execution progresses. For these systems the software performance may also show partial degradation with the progress of execution.

The software aging mechanisms reported in the software aging literature [7] are known to trigger aging related bugs (ARBs) [8], by one of several causes such as, (1) numerical error accumulation, (2) resource leakage, (3) fragmentation, and, (4) performance degradation due to shared resource (software deadlocks may occur depending on the time to live, or age, of the data lock). Some real case analysis of these software defect mechanisms are provided in [9], [7], [5], [10] and reported in Table I, such as, (1) round off accumulation errors in the Patriot missile defense system, (2) file descriptor exhaustion in Linux based software systems, (3) failure to run garbage collection in Java based systems due to memory fragmentation, (4) database deadlocks due to cycles in the request/release graph.

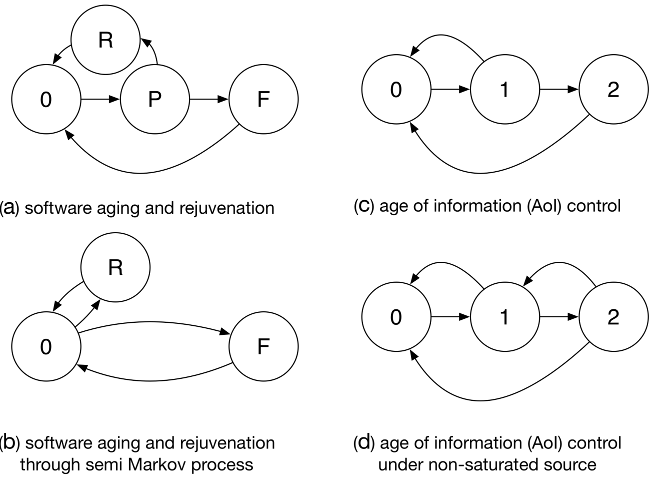
R
0
P
F
(a) software aging and rejuvenation
0
1
2
(c) age of information (AoI) control
R
0
F
(b) software aging and rejuvenation through semi Markov process
0
1
2
(d) age of information (AoI) control under non-saturated source

Fig. 1. Rejuvenation and AoI (from [5], [15], [16], [17], respectively)

B. Measurements of software aging: static and dynamic thresholds

Software performance indicators are pieces of information that can age as a consequence of software performance degradation. Information aging is intrinsically related to rejuvenation approaches based on static thresholds to detect system performance degradation [11], [12]. Using adaptive thresholds – possibly combined with statistical prediction techniques – solves the problem of aging of static thresholds. Still, the rate at which the thresholds must be rejuvenated should be determined. In [11], [12], aging detection using sampling of response times and software rejuvenation methods are presented and shown to have positive impact on the customer affecting metrics. In [13], [14], software rejuvenation is shown to be able to extinguish worm epidemics in tactical MANETs.

C. Slow execution after restart

Many software systems require learning, hot caches, and population of internal states to operate at high-performance. For example, in operating systems, after reboots, disk access will be more frequent, because file caching will be operating in the cold state [18]. In this case, the lack of control over the Age of Information will be the source of performance degradation. As additional information is gathered about the system, e.g., as caches get populated with useful data, applications will show improved performance. Therefore, in this case a bootstrapping warm-up is required for optimal performance.

D. Dynamic versus static information

Most of the literature on the age of information assumes that information is static (e.g., news articles) and that its value degrades over time due to aging. Obsolete information should be refreshed. However, one may also consider dynamic information, which is updated locally based on prediction algorithms. Such algorithms are prone to failure due to numerical errors, and the error accumulation depends on the age of information. Therefore, for information that is dynamically updated, the uncertainty about data increases over time both due to numerical errors and due to intrinsic increase in uncertainty as the prediction horizon increases. Errors due to actions taken with outdated information have been broadly analyzed in the software aging literature [19], and bridge the fields of software aging and age of information.

TABLE I
SOFTWARE AGING CLASSES AND THEIR CONNECTION TO STALE INFORMATION

Class	Information	Aging aspect
Numerical error	time since boot	error in time estimate gradually increases
Resource leakage	amount of consumed resources	consumption estimate gets outdated, and resources blocked (out of resources)
Fragmentation	degree of fragmentation and object states	object states get outdated, requiring updates causing fragmentation and more frequent garbage collection
Performance degradation due to shared resource	state of shared resource	state gets outdated, causing race conditions and deadlocks

III. Preventive maintenance: Markov and Semi Markov models for aging and rejuvenation

A. Preventive maintenance: condition-based versus time-based maintenance

In the literature on preventive maintenance, Markov models are generally used for condition-based (inspection-based or measurement-based) preventive maintenance [20] while semi Markov models have been used for time-based preventive maintenance [21]. In the former, aging is indicated by a set of states, each indicating a level of degradation. Even if the sojourn time in each state is exponentially distributed, cumulative time to failure will be non-exponential and can have an overall increasing failure rate. If the system is detected to be in a low-degradation state, preventive maintenance is immediately triggered. In the latter, time to failure is considered to be a non-exponentially random variable and thus is not synthesized by a sequence of states. If this distribution as well the mean time to carry out reactive (unscheduled) recovery and the mean to carry out proactive (preventive) recovery are known then the optimal time to trigger rejuvenation can be determined.

B. Observable and non-observable states

Referring to Figure 1(a), which corresponds to Fig. 2 in [5], Huang et al. [5] considered the failure probable state S_P, as an observable state and thus in the traditional language of preventive maintenance literature, in Figure 2 of [5] the authors considered condition-based or inspection-based preventive maintenance (rejuvenation in this case). If this is indeed the case then we should trigger rejuvenation immediately from the S_P state and not wait for any length of time (as was done in the original paper). We can alternatively consider the transition to the failure state S_F from the robust state S_0 via the failure probable state S_P as merely an artifice so that the overall time to failure is hypo-exponentially distributed [22],

TABLE II
COMPARING AGE OF INFORMATION AND SOFTWARE AGING

	variable of interest	rejuvenation	control variable	uncertainty over	failure	sampling to learn
software aging	software age	process restart	software restart rate	time to failure	software crash	process state
age of information	information age	information refresh	refresh rate	network availability age at source	read outdated news	age at source

TABLE III
CONDITION-BASED VERSUS TIME-BASED MAINTENANCE

	condition-based	time-based
AoI rejuvenation triggered by	signal from source, signal from GPS or contact with WiFi	alarm indicating elapsed time
Software rejuvenation triggered by	signal from monitor or system slow down	alarm indicating elapsed time

[23]. Since the hypo-exponential distribution has a failure rate that is increasing with age, we know that, in general, in such a case, preventive maintenance could be useful. In this case, we can consider the state S_P as not observable and hence the rejuvenation trigger clock should start on entry to state S_0. This is traditionally known as time-based rejuvenation wherein the optimal time to trigger rejuvenation can be studied [24], [21], [23].

The time to trigger rejuvenation under a time-based rejuvenation can be deterministic (or generally distributed). This eventually led to a paper published in ISSRE 1995 [25]. Because of the existence of a generally distributed timed transition (rejuvenation trigger) that is concurrent with the transition from S_0 to S_P as well as the transition from S_P to S_F, the resulting stochastic process is not even a semi-Markov process but a Markov regenerative process. Thus, Fig. 1(a) (which corresponds to Figure 2 of [5]) morphs into a Markov Regenerative Stochastic Petri Net (MRSPN) (see Figure 1 in [25]).

C. CTMC for condition-based maintenance and SMP for time-based maintenance

In order to avoid the complication of working with fully non-Markovian models, we consider a clear separation between the time-based case and condition-based case. In the former we use an SMP that includes a time trigger (Fig. 1(b)) and in the latter we use CTMC where the trigger is based on state identification and not on any time trigger (Fig. 1(a)). This is what has been used in the subsequent papers [15], [26], [27].

D. Condition-based and time-based rejuvenation for AoI

In the realm of Age of Information (AoI) researchers have also used semi Markov models but in a different fashion, and there has not been a systematic comparison of the Markov and semi Markov approaches – the semi Markov approach has been used in the literature of age of information in the following references [28], [29], [30], [31], [32], [33] and the Markov approach has been used in [17], [16]. The discussion in the previous sections motivates the following considerations for AoI control (see Table III):

- condition-based AoI control: if the controller receives signals about the state of the network and about the state of the source, a Markov model is adequate to capture the state dynamics;
- time-based AoI control: if the controller actions are based on triggers that are dependent on the time that has elapsed since the previous refresh, a semi-Markov model is adequate to capture the age of information.

E. Markov models for aging control

In Figures 1(c) and 1(d) we illustrate two Markov models that have been used to study time-based AoI control. In those models, the states characterize the age. Markov models simplify the derivation of the optimal control strategy using Markov Decision Processes (MDPs). Alternatively, semi-Markov Decision Processes (SMDPs) can be used to analyze the control problem in the case of general residence time distributions.

In Figures 1(c) and 1(d), the age varies between 0 and 2. In Fig. 1(c) it is assumed that after a successful rejuvenation, the age will always decrease to 0 (saturated source) whereas in Fig. 1(d) a successful rejuvenation may be followed by a decrease of the age from 2 to 1, meaning that the source contained outdated information at the time of the rejuvenation (non saturated source). In both cases, note that there is only one single event of interest, namely the rejuvenation. In Fig. 1(a) and 1(b), in contrast, there are two events of interest: rejuvenation and failure, which are concurrent.

Consider a user that accesses his mobile device and consumes outdated information. If this event is assumed to be concurrent to the rejuvenation event, then we again have two events of interest: information rejuvenation and failure due to access of stale information. Under that framework, the models in Figures 1(a) and 1(b) can be used to study the dynamics of AoI. Conversely, if a software smoothly degrades over time, and the degradation level is solely a function of the time since the last rejuvenation, the models in Figs. 1(c) and 1(d) can be used to analyze software aging and rejuvenation.

In Section VI we provide a case study of an additional type of software aging, in addition to the four already mentioned in Section II, and we show the application of SMP modeling to this case study.

IV. SAMPLING, SOFTWARE REJUVENATION AND AOI

In this section we describe how software rejuvenation relates to Age of Information approaches. In particular, software rejuvenation can be used to mitigate software epidemics [34], [24]. We leverage the relationship between rejuvenation and sampling established in [34], [24] to indicate its applicability in the field of Age of Information, where sampling also plays a key role [33].

A. Sampling, epidemics and software rejuvenation

In [34], [24], a survivability model was introduced with the goal of assessing infection and contagion probability in mission-critical systems that are the target of advanced persistent threats. The attack model used to replicate advanced persistent threats consisted of three attack phases: (1) single node, initial breach, (2) attack propagation to other nodes, and, (3) attack causes suspicious behavior. The goal of the research was to assess the effectiveness of possible countermeasures, such as, quarantine, and software rejuvenation. Three survivability models were developed, corresponding to each attack phase, in the context of advanced persistent threats. They were designed to capture the effect of neighbor nodes on different system metrics related to advanced persistent threats, such as, infection probability, and contagion probability.

An approach related to software rejuvenation was implemented to restore nodes to non-infected states. The authors have found that: (1) malware infections related to advanced persistent threats could be mitigated by using quarantine, software rejuvenation, and, node sampling for infections, (2) sampling plays a key role in software rejuvenation, as it is important to know the system state, to determine when/if to rejuvenate, and, (3) software rejuvenation and quarantine are able to extinguish the network infection, when exogenous infection rate is small.

B. Sampling, epidemics and AoI

In the context of Age of Information, **infection probability** can be seen as an Age of Information metric that is susceptible to software epidemics. As the epidemic spreads in the network, the opportunity for epidemics mitigation ages, if adequate countermeasures are not promptly activated. Therefore, there is a game theory aspect to worm epidemics that is reflected in the tradeoff between pro-actively activating countermeasures and their impact on software performance, as shown in [34], [24].

In the realm of the age of information, epidemics are related to the spread of information, e.g., across users timelines. An example of negative epidemic is fake news propagation in social networks. Rejuvenation, in turn, is one possible countermeasure against fake news. The methods developed in [34], [24], together with the timeline model discussed in [35], set the groundwork to analyze the interplay between rejuvenation and AoI, noting that rejuvenation corresponds to the sharing of a positive post which removes a fake news from the top of the timeline of a user. The age of fake news increases as a function of the rate at which positive posts are shared.

V. Software aging types and relationship to AoI

In this section we describe how Age of Information concepts can help derive novel software rejuvenation schemes by identifying the Age of Information metrics that are related to the four software aging types introduced in Section II, namely: (1) numerical error accumulation, (2) resource leakage, (3) fragmentation, and, (4) performance degradation due to a shared resource (software deadlocks may occur depending on the time to live, or age, of the data lock). These were taken from the software aging literature [7]. We expect that by identifying the Age of Information metric new methods that are focused on Age of Information and its rejuvenation will be developed.

A. Numerical error accumulation

In the Patriot missile battery failure example [7], the Age of Information metric is the **error in system time computation since boot time**. As the system ages, this Age of Information metric increases. The missile failure was related to an error in the system time computation that involved the use of a 24 bit fixed point register truncation, which created a small computer arithmetic error. However, this small error, when multiplied by the large number representing the elapsed time since boot time, created a large enough error that caused the Patriot missile battery to fail its mission of tracking and intercepting an incoming Scud missile. This failure occurred once the Patriot missile battery was up for over 100 hours. As a mitigation procedure, a software rejuvenation could be triggered to reduce the error in system time computation since boot time.

B. Resource leakage

Software aging as a consequence of resource leakage has been surveyed in detail in [36]. Specifically, the following resources have been reported in the software aging literature to show leakage: memory, file descriptors, sockets, space consumption in data storage systems, locks, threads and processes. A resource of type X is *active* if there is at least one process using that resource. The related Age of Information metric for a resource type X is **the number of resources of type X that are active in the system at time t.**

C. Fragmentation

Memory fragmentation in Java-based systems can impact the performance of its garbage collection engine. Memory fragmentation occurs when the free memory blocks sizes are not large enough to satisfy memory requests. Out-of-memory error is a Java exception that is thrown when memory requests cannot be met. The Age of Information metric for memory fragmentation is **the degree of fragmentation**. When the degree of fragmentation is outdated, the garbage collection engine may be activated more frequently causing system slow down. Some approaches that could be envisioned for software rejuvenation in this case are related to better tuning of the Java garbage collector [37]. In addition, approaches for rejuvenating a memory fragmented system are described in [36].

TABLE IV
AoI, SCHEDULING AND LAXITY

	AoI	Software aging
Schedule rejuvenation of	piece of information	software component
Laxity is slack in time until	successful information refresh	successful software rejuvenation

D. Performance degradation due to shared resource

The age of information metric related to Performance degradation due to data corruption or deadlocks is **waiting time for accessing shared resource** X. As the contention for a shared resource increases, the expected waiting time for that resource will show Age of Information as represented by the increased waiting time for the resource. Several approaches for software rejuvenation of shared resource are presented in [36].

VI. Age control, scheduling and laxity

Next, we illustrate the applicability of the relationship between concepts from software aging and age of information in the realm of scheduling. To that aim, we indicate that the concept of laxity, which is motivated by load balancing schedulers and preventive maintenance, is also helpful for the analysis of age of information (see Table IV).

A. Laxity

In real-time systems, an important metric is the probability a task will meet its deadline, given the task laxity. Laxity is defined as the task slack, or the difference between the deadline time and the total remaining task execution time, if the task is scheduled for immediate execution. In the realm of information system, the deadline associated to a piece of information may correspond to the instant by which it must have been refreshed, as by that time the value of stale information vanishes to zero.

In a time slotted system, if an event of interest (e.g., task completion or information update) does not occur at a given slot, at that slot the age is incremented, and the laxity or the residual lifetime are correspondingly decremented. Therefore, the maximum lifetime (of information) as estimated at a given slot is given by

$$MoI = AoI + RoI + LoI, \qquad (1)$$

where *RoI* refers to the expected residual lifetime of information (e.g., time until refresh), *LoI* refers to the laxity of information, and *MoI* refers to the maximum lifetime tolerated for that piece of information (see Fig. 2).

A successful rejuvenation of a piece information requires, for instance, Internet access (e.g., through a WiFi network which may be intermittently available). Therefore, the expected time until successful rejuvenation is the mean time to perform a scheduled rejuvenation, e.g., accounting for the uncertainties about network access.

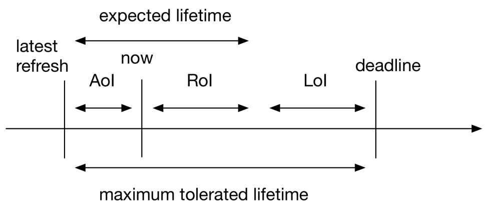

Fig. 2. Age of Information (AoI) and residual lifetime of information together with laxity produce the maximum tolerated lifetime.

B. An approach for aging detection and rejuvenation control

We next illustrate an approach for aging detection and rejuvenation control. The approach monitors an Age of Information metric (AoI-related metric), as for example *laxity*, and activates rejuvenation to ensure the system will meet its hard real-time deadlines [38].

Given a piece of information, the following mechanism can be used to control the AoI.

Age of Information Rejuvenation Algorithm

```
1) For every piece of information $i$, if $LoI(i) < LaxityThreshold(i)$ trigger AoI rejuvenation.
2) At every time slot boundary:
    a) for every piece of information $i$, increment $AoI(i)$.
    b) if $AoI(i)$ reached its target value trigger AoI rejuvenation.
    c) decrement $LoI(i)$ if the expected time to refresh information $i$ is delayed (e.g., due to unforeseen network unavailability).
```

Note that the rejuvenation at step 1) is executed in a preventive fashion, given that conditions are not-favorable and the process of rejuvenation should be triggered immediately. Otherwise, the deadline may not be met. At step 2b), in contrast, rejuvenation is triggered in a reactive fashion, according to a pre-established schedule.

C. Aging and real-time task schedulers

Next, we draw a parallel between the algorithm presented in the previous section and classical algorithms for task scheduling in multi-processor systems. Our aims are to (i) illustrate some results from the multi-processor task scheduling literature that may inspire AoI rejuvenation algorithms and (ii) in some instances, indicate more directly that age of information plays a key role in the problem of task scheduling.

1) General remarks about task scheduling: When a task is not scheduled for immediate execution, the task probability of meeting its deadline will typically decrease as a function of time [38], [39], [40]. The AoI rejuvenation algorithm presented above can be adopted for general task scheduling as follows. Consider a set of tasks, I, which are described by a minimum allowed $LaxityThreshold(i)$. At every time slot, for each task i, either $L(i)$ or $R(i)$ are decremented (where L and R are the laxity and expected residual life of tasks, respectively). If any $L(i) < LaxityThreshold(i)$, the AoI-related rejuvenation procedure is triggered.

AoI-related rejuvenation, in the system model considered in the remainder of this section, consists of activating the load balancing scheduling algorithm to search for a better processor match to meet the tasks laxity requirements. Note that such a rejuvenation implies also a rejuvenation in the age of information (as information about processes will be updated). Nonetheless, it may also imply additional overhead (such as task migration) [38], [39], [40], [27], [41].

2) System model: To make the discussion that follows concrete, we borrow a typical system model used for the analysis of scheduling of tasks in a multi-processor system. The system model considered in this section follows [38] and consists of the following features:

- the system is composed of n fast processors connected by a load balancing fast packet switching network,
- the system is synchronous, time slotted,
- a load balancing scheduler takes into account number of busy slots in each processor and task laxity,
- tasks arrive at local nodes according to a Poisson process,
- tasks cpu execution times are discrete,
- each cpu supports a maximum number of busy slots,
- tasks are scheduled for execution at a local cpu or at a remote cpu using a load balancing scheduling network.

The load balance scheduling network used for real-time scheduling employs three stages [38]: (1) a sort network stage to sort task allocation requests and cpu offers using laxity and number of busy slots, (2) a matching network stage to match laxity requests to number of busy slots per cpu, (3) a self-routing network to deliver task matches to the scheduled cpu.

3) Markov modeling, $M^k/D/n$: In [38] a slotted hard-real time system with n processors was modeled as an $M^k/D/n$ Markovian model to obtain lower bounds on the average fraction of tasks that are dropped because they miss the hard real-time deadline. Tasks were modeled with laxity requirements and cpu demands as described in the system model above. The state of the Markov chain was defined as $S = (N_0, N_1, ..., N_n)$, where N_k is the number of busy slots in processor k at given time instant. The resulting Markov model was solved numerically and shown to have a good accuracy when compared against realistic simulation results.

4) Semi-Markov modeling: In Section III, we introduced the application of SMP to software aging and rejuvenation that includes a time trigger, as presented in [27]. In the hard real-time system considered in this section, the time trigger is represented by the expiration of the time slot and the associated re-computation of the remaining tasks laxities and number of busy slots per processor. The activation of the rejuvenation trigger is modeled by state R as shown in Figure 1(b).

In [41], the authors have formulated an optimization problem by using semi-Markov decision process and have applied it to model a dynamic software rejuvenation policy for a multistage degradation system. They have shown that the proposed control-limit policy is optimal. The approach used was to sample the state of software system, and to determine the best times to initiate the software rejuvenation, taking into account the expected operational cost. In the context of hard-real time system and Age of Information, we expect that the proposed approach in [41] can be used for developing an optimal control limit approach based on sampling of the remaining laxity and using as expected operational cost the probability of the task missing the hard-real time deadline.

5) Blackbox modeling: In [39], [40] an approach for using task restarts was modeled with the objective to create an optimal strategy where the expected task remaining computation time might violate the task laxity requirement. The authors have assumed that tasks completion times were independent between successive attempts and have formulated an analytic model to derive the probability that a given task will meet its hard real-time deadline. We envision that the approach introduced in [39], [40] could be used in the realm of Age of Information rejuvenation, following an approach similar to the one introduced in this section.

VII. Related Work

The terminology used in the fields of age of information and software aging suggests that there is significant potential synergy between the two. Nonetheless, the only reference to rejuvenation in the literature about age of information (AoI) appears in [2], where the authors refer to Juventas as the ancient Roman goddess for youth and rejuvenation.[1]

As the age of information concept (which dates back to 2010) is much more novel than that of software aging and rejuvenation (which dates back to the 1994-1995 timeframe [4], [5]), it is not surprising that the notion of age of information has not been reported in the literature of software aging [36]. The workshop on software aging and rejuvenation is in its 11-th edition [42], whereas the workshop on age of information is in its 2nd edition [43]. We envision that as the two fields mature, and the two communities interact, the interplay between the two will also become clearer and evolve.

VIII. Conclusion

In this paper we have considered the applicability of software aging and rejuvenation research to the Age of Information paradigm. We have presented a brief summary of the interplay between software aging and age of information by considering the several aspects of software aging and rejuvenation research, as for example, the types of aging reported in the literature, measurement of software aging and slow execution after restart. We have also considered dynamic vs. static information for software aging. In each case, we have identified the Age of Information metric that could be used to support new software rejuvenation approaches based on the Age of Information concept.

We have provided an overview of semi-Markov approaches toward aging and rejuvenation and have provided guidance on when to use CTMC, for condition-based approaches, or SMP, for time-based scheduling of rejuvenation or refresh. We

[1]The corresponding ancient Greek goddess for youth and rejuvenation is called Hebe serving nectar and ambrosia (rejuvenation techniques) to keep gods (important computer systems) forever young.

have also presented a case study, for hard-real time systems, which introduced an additional type of software aging and rejuvenation based on Age of Information concept, and we have presented an overview on how to transition to Age of Information concepts from the four software aging types considered in this paper.

This work opens up a number of interesting directions for future work. Intentional aging, for instance, may occur both for age of information (to increase click rates and revenue) as well for software aging (to increase the selling of new products). We envision that game theory can be instrumental to study those aspects, which are left as subject for future work.

Acknowledgment: This work was partially supported by CAPES, CNPq, eSulab Solutions, Inc. and FAPERJ, as well as by the joint Brazilian-French team THANES (jointly sponsored by FAPERJ and INRIA).

REFERENCES

[1] A. Kosta, N. Pappas, V. Angelakis *et al.*, "Age of information: A new concept, metric, and tool," *Foundations and Trends® in Networking*, vol. 12, no. 3, pp. 162–259, 2017.

[2] C. Li, S. Li, and Y. T. Hou, "A general model for minimizing age of information at network edge," in *INFOCOM*, 2019, pp. 118–126, https://www.cnsr.ictas.vt.edu/publication/general-model.pdf.

[3] Y. Sun, T. Z. Ornee, and M. K. C. Shisher, "The ongoing history of age of information," 2019, http://webhome.auburn.edu/~yzs0078/AoI.html.

[4] A. Avritzer and E. J. Weyuker, "Estimating the software reliability of smoothly degrading systems," in *ISSRE*, 1994, pp. 168–177.

[5] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Symposium on Fault-Tolerant Computing*. IEEE, 1995, pp. 381–390.

[6] S. Hao and L. Duan, "Economics of age of information management under network externalities," *arXiv preprint arXiv:1904.01841*, 2019.

[7] M. Grottke, R. Matias Jr, and K. Trivedi, "The fundamentals of software aging," in *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, 12 2008, pp. 1 – 6.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2004.2

[9] A. Avritzer and E. J. Weyuker, "The role of modeling in the performance testing of e-commerce applications," *IEEE Trans. on Software Engineering*, vol. 30, no. 12, pp. 1072–1083, Dec 2004.

[10] K. Vaidyanathan and K. Trivedi, "Measurement-based model for estimation of resource exhaustion in operational software systems," 02 1999, pp. 84 – 93.

[11] A. Avritzer, A. B. Bondi, M. Grottke, K. S. Trivedi, and E. J. Weyuker, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in *2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings*, 2006, pp. 435–444. [Online]. Available: https://doi.org/10.1109/DSN.2006.58

[12] A. Avritzer, A. B. Bondi, and E. J. Weyuker, "Ensuring stable performance for systems that degrade," in *WOSP*, 2005.

[13] A. Avritzer, R. Cole, and E. Weyuker, "Using performance signatures and software rejuvenation for worm mitigation in tactical manets," in *WOSP*, Jan 2007, pp. 172–180.

[14] A. Avritzer, R. G. Cole, and E. J. Weyuker, "Methods and opportunities for rejuvenation in aging distributed software systems," *Journal of Systems and Software*, vol. 83, no. 9, pp. 1568 – 1578, 2010, software Dependability.

[15] J. Zhao, Y. Jin, K. S. Trivedi, Y. Wang *et al.*, "Software rejuvenation scheduling using accelerated life testing," *ACM JETC*, vol. 10, no. 1, p. 9, 2014.

[16] E. Altman, R. El-Azouzi, D. Menasche, and Y. Xu, "Forever young: Aging control for hybrid networks," in *MOBIHOC*, 2019, pp. 91–100.

[17] I. Kadota and E. Modiano, "Minimizing the age of information in wireless networks with stochastic arrivals," *arXiv preprint arXiv:1905.07020*, 2019.

[18] H. Yamada and K. Kono, "Traveling forward in time to newer operating systems using shadowreboot," *SIGPLAN Not.*, vol. 48, no. 7, pp. 121–130, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2517326.2451536

[19] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *2010 IEEE/IFIP DSN*. IEEE, 2010, pp. 447–456.

[20] D. Chen and K. S. Trivedi, "Closed-form analytical results for condition-based maintenance," *Rel. Eng. & Sys. Safety*, vol. 76, no. 1, pp. 43–51, 2002.

[21] ——, "Analysis of periodic preventive maintenance with general system failure distribution," in *8th Pacific Rim International Symposium on Dependable Computing (PRDC 2001), 17-19 December 2001*, 2001, pp. 103–110.

[22] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, 2nd ed. John Wiley & Sons, 2001.

[23] K. Trivedi and A. Bobbio, *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, 2017.

[24] E. Altman, A. Avritzer, R. El-Azouzi, D. S. Menasche, and L. P. de Aguiar, "Rejuvenation and the spread of epidemics in general topologies," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE, 2014, pp. 414–419.

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

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

[27] K. Vaidyanathan and K. Trivedi, "A comprehensive model for software rejuvenation," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 124–137, April 2005.

[28] E. T. Ceran, D. Gündüz, and A. György, "Average age of information with hybrid arq under a resource constraint," *IEEE Trans. on Wireless Communications*, vol. 18, no. 3, pp. 1900–1913, 2019.

[29] J. Liu, X. Wang, B. Bai, and H. Dai, "Age-optimal trajectory planning for uav-assisted data collection," in *INFOCOM*. IEEE, 2018, pp. 553–558.

[30] E. Najm, "Bits through time," EPFL, Tech. Rep., 2019.

[31] E. Najm and E. Telatar, "Status updates in a multi-stream m/g/1/1 preemptive queue," in *INFOCOM*. IEEE, 2018, pp. 124–129.

[32] E. Najm, R. Yates, and E. Soljanin, "Status updates through m/g/1/1 queues with harq," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 131–135.

[33] T. Z. Ornee and Y. Sun, "Sampling for remote estimation through queues: Age of information and beyond," *arXiv preprint arXiv:1902.03552*, 2019.

[34] M. Grottke, A. Avritzer, D. Menasché, L. de Aguiar, and E. Altman, "On the efficiency of sampling and countermeasures to critical-infrastructure-targeted malware campaigns," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 4, pp. 33–42, 2016.

[35] A. Giovanidis, B. Baynat, and A. Vendeville, "Performance analysis of online social platforms," in *INFOCOM*. IEEE, 2019, pp. 2413–2421.

[36] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, 01 2014.

[37] S. Oaks, *Java Performance: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2014.

[38] A. Avritzer, M. Gerla, and J. W. Carlyle, "A load sharing interconnection network for hard real-time systems," *[1991] Proceedings. The Fifth International Parallel Processing Symposium*, pp. 591–598, 1991.

[39] A. Van Moorsel and K. Wolter, "Analysis and algorithms for restart," in *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.* IEEE, 2004, pp. 195–204.

[40] ——, "Analysis of restart mechanisms in software systems," *IEEE Trans. on Software Engineering*, vol. 32, no. 8, pp. 547–558, 2006.

[41] H. Eto and T. Dohi, "Determining the optimal software rejuvenation schedule via semi-markov decision process," *Journal of Computer Science*, vol. 2, no. 6, pp. 528–534, 2006.

[42] Wosar, "Workshop on software aging and rejuvenation," 2019, http://2019.issre.net/node/74.

[43] AoI, "Age of information workshop," 2019, https://infocom2019.ieee-infocom.org/age-information-workshop.