Enhancing the Reliability of Perception Systems using N-version Programming and Rejuvenation

Júlio Mendonça[1], Fumio Machida[2], Marcus Völp[1]
[1]*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg*, Luxembourg
[2]*Department of Computer Science, University of Tsukuba*, Japan
{julio.mendonca, marcus.voelp}@uni.lu, machida@cs.tsukuba.ac.jp

Abstract—**Machine Learning (ML) has become an indispensable feature for real-world complex systems, such as perception systems of autonomous systems and vehicles. However, ML-based systems are sensitive to input data, faults, and malicious threats that can degrade output quality and compromise the complete system's correctness. Ensuring a reliable output of ML-based components is crucial, especially for safety-critical systems. In this paper, we investigate architectures of perception systems using N-version programming for ML to mitigate the dependence on a singular ML component. Additionally, we combine the use of multiple ML modules with a time-based rejuvenation mechanism to maintain a healthy system over extended periods. We propose a set of models and functions to evaluate the reliability of an N-version perception system subject to faults, malicious threats, and rejuvenation. Our numerical experiments show that a rejuvenation mechanism could benefit a multiple-version system, with a reliability improvement superior to 13%. Also, the results indicate that rejuvenation could further improve output reliability when ML modules' accuracy is high.**

Index Terms—**Byzantine fault, N-version system, Machine learning, Perception, Rejuvenation, Reliability**

I. Introduction

The rapid development of machine learning (ML) has significantly impacted various domains, such as computer vision, natural language processing, autonomous driving, and robotics. Its use has become a powerful ally for autonomous systems of different domains to perform complex tasks accurately and efficiently with minimal human intervention. Perception systems are one of the most crucial components for autonomous systems, such as autonomous vehicles, employing ML to execute complex perception tasks. These systems perceive and interpret the surrounding environment, relying on sensors such as cameras, lidars, and radars, and must ensure a safe and reliable output that genuinely represents the authentic environment [1].

As other systems, perception systems are also subject to faults (e.g., caused by bit-flip errors [2]) and malicious attacks (e.g., adversarial attacks [3]), that can degrade ML modules' accuracy or stop it completely, precluding the system's capability of providing a correct output. Perception errors may have severe and irreversible consequences depending on the application, such as in autonomous vehicle accidents [4]. As a result, guaranteeing a reliable and correct perception output even when those systems are subject to faults and malicious attacks is a major challenge.

Several studies have explored different methods and techniques to build more reliable perception systems [1], [5], [6]. Researches adopting N-version programming (NVP) [7] for ML-based systems has demonstrated significant improvement in the system's output reliability. The main idea of NVP is to utilize replicated and diverse system components capable of performing the same tasks. When adopting multiple versions, various architectural possibilities exist due to diverse ML modules and input data. Modeling approaches have investigated different architectures. For instance, Ege et al. [8] have proposed a general model for computing the reliability of an N-version programming software that could be extended to ML-based systems. The two-version system [9], [10] and the three-version system [11] are the architectures of $N = 2$ and 3, respectively, known to be theoretically effective for improving the system output reliability. Besides, Latifi et al. [12], Gujarati et al. [13], and Xu et al. [14] have experimentally analyzed N-version for ML systems to improve output reliability. However, previous studies have not explored the impacts caused by faults or malicious attacks on these systems. Furthermore, the combination with other techniques, such as software rejuvenation, has not been sufficiently investigated [15].

Therefore, this paper explores the NVP ML and software rejuvenation combination to enhance perception systems' output reliability. We model and evaluate perception system architectures employing multiple ML modules and time-based software rejuvenation to recover compromised ML modules proactively. We use Deterministic and Stochastic Petri Net (DSPN) to represent (1) faults and malicious attacks' effects on a perception system, (2) recovery actions after failures, and (3) a proactive time-based rejuvenation mechanism for the ML modules. Then, we define reliability functions to capture the output reliability in different states that the system may assume. The reliability functions are defined based on a voting scheme necessary to decide whether a request is correct when multiple versions are used (e.g., majority voting). This paper explicitly focuses on voting schemes used by Byzantine fault-tolerant (BFT) protocols, where correctness can only be guaranteed when there are fewer compromised components than a pre-defined threshold. We perform numerical experiments in four- and six-version perception systems using the proposed models and reliability functions. Our analysis shows that the reliability of the six-version system adopting a rejuvenation mechanism is higher than the four-version system without adopting rejuvenation. Additionally, we investigate different scenarios to find the optimum values of key input parameters to maximize the system output reliability in both cases where rejuvenation is used or not. Our contributions are:

- adoption of a rejuvenation method for perception system architectures with N-version ML;
- reliability models and functions to evaluate the expected output reliability of perception systems, taking into account faults, malicious attacks, rejuvenation, and BFT voting schemes;
- numerical analysis to find optimum values of key input parameters where output reliability is maximized.

The remainder of the paper is organized as follows. §II presents fundamental concepts and related work. §III presents a perception system architecture adopting a rejuvenation mechanism. §IV presents the developed models and functions to analyze the reliability of N-version perception systems. §V presents and discuss the numerical analysis, demonstrating the proposed models' applicability. Finally, §VI concludes this paper and briefly explains future work.

II. Background and Related Work

A. Software rejuvenation

Rejuvenation is a preventive maintenance technique to avoid system degradation by proactively recovering components to a safe and healthy state before they degrade. It has been studied and applied in different systems and areas, such as BFT protocols and cloud environments, to enhance fault tolerance, reliability, and availability. Rejuvenation modeling has also been successfully employed to evaluate availability [16], the impact of aging-related bugs in software systems [17], and reliability [18]. However, few works consider this technique as an instrument to improve output reliability in ML systems [15].

B. Byzantine Fault-Tolerance & DSPN modeling

Byzantine fault-tolerant (BFT) consensus protocols have been designed to ensure systems consistency even in the presence of Byzantine faults, which are arbitrary, and include malicious behaviors [19]. BFT protocols are commonly designed to run on $n \geq 3f+1$ replicas to guarantee systems' correctness under a pre-defined number of f faulty or malicious replicas, but adaptive protocols have been proposed recently [20]. For the common case, $2f+1$ replicas must reach a consensus to validate and execute a request. However, protocols considering rejuvenation mechanisms have also been proposed to mitigate failure and attacks in BFT-based systems. These protocols must contain $n \geq 3f + 2r + 1$ replicas to support f faulty or malicious replicas and r replicas simultaneously rejuvenating [21]. Therefore, to reach a consensus and execute a request, $2f+r+1$ replicas must agree on the request. Further details can be found on [19] and [21].

Deterministic and Stochastic Petri Nets (DSPNs) is a formalism highly adopted for modeling systems faults, rejuvenation, and malicious attack behaviors and evaluating performance and dependability metrics such as reliability. DSPNs allow both stochastic and deterministic transitions to represent events in systems. While stochastic transitions follow an exponential time distribution, deterministic ones follow a uniform time distribution. DSPNs follow Petri nets representation, where tokens are small black circles, places white circles, and transitions rectangles. Immediate transitions are thin black, stochastic transitions are white and deterministic transitions are bold black rectangles. Places can store tokens, and the transitions firing consume tokens from one place and generate new tokens in another. Arcs are represented by arrows and connect places and transitions, defining the tokens' flow through the places, and can have weights. Inhibitor arcs are defined by an arrow ending with a small white circle, capable of disabling a transition when its weight is met. We refer the reader to [22] for further details.

III. Rejuvenation for N-version Perception Systems

This section describes adopting a rejuvenation mechanism for an N-version perception system. The primary motivation for combining these two techniques is to provide more reliable output in perception systems. Figure 1 presents an architecture for such a perception system. In the architecture, we consider (1) different types of sensors providing input data for ML modules that run inside the perception system; (2) N-version ML modules subject to faults and attacks to execute perception tasks; (3) a voter to collect the individual results of each ML module and decide the final perception output based on a pre-defined voting scheme; and (4) a time-based rejuvenation mechanism to rejuvenate the healthy or compromised ML modules back to a healthy state.

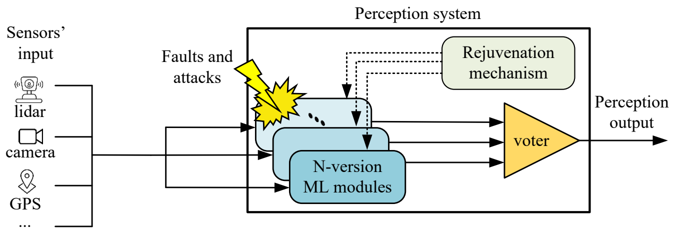


Fig. 1: An architecture of a perception system subject to faults and attacks that adopts N-version ML and a time-based rejuvenation mechanism.

Faults and malicious attacks in perception systems may compromise ML modules' accuracy. In such a case, we can assume ML modules have three possible states: healthy (*H*), compromised (*C*), or non-operational (*N*). When in a healthy state, an ML module operates normally. However, when Byzantine faults or malicious attacks affect an ML module, it enters a compromised state (*C*), consequently reducing its accuracy and the probability of producing correct outputs. In the case faults or attacks endure, the ML module will eventually enter a non-operational state (*N*), incapable of executing perception tasks. Usually, failure detection tools can easily detect whether a component is operational. Thus, by leveraging such tools, the system may employ a recovery mechanism to recover an ML module from a non-operational (*N*) to a healthy (*H*) state.

When using a rejuvenation mechanism, like in the architecture shown in Figure 1, ML modules benefit it by recovering from ongoing faults or attacks. For instance, one may consider reloading and redeploying an ML module from a safe memory location as a rejuvenation action. Hence, the rejuvenation process consists of recovering ML modules from a healthy (*H*) or compromised (*C*) state back to the healthy state (*H*). Although an ML module cannot execute perception tasks while rejuvenating, it may benefit the whole system by returning the total capacity to complete a perception task after rejuvenation.

Since ML modules can perceive the environment differently, a voting system is necessary to decide the final perception output. The voter may assume different schemes to decide about each final perception output. For instance, voting schemes include simple majority (e.g., 2-out-of-3) [11] or unanimity between the ML modules (e.g., 5-out-of-5) [12]. In this paper, we focus on analyzing BFT-like voting schemes. Specifically, we focus on perception systems that deal with a pre-defined number of compromised f ML modules and can simultaneously support r ML modules rejuvenating or recovering (see §II-B).

IV. Reliability Models for N-version Perception Systems

This section describes the developed models to evaluate reliability in N-version perception systems. We first present the fault and threat model assumed for the models and the modeling assumptions. Next, we present the developed DSPNs for the system. Lastly, we explain the reliability functions defined for a four- and six-version perception system.

A. Threat and Fault Model

This section summarises the assumed malicious attacks and faults represented in the proposed models as follows: (1) attacks and faults can continuously happen in the system, leading to a misperception of the ground truth; (2) attackers can compromise the accuracy of one ML module per time, for instance, through adversarial [3] or evasion attacks [23], which happen in a given amount of time; (3) transient faults such as memory failure or bit flip errors [2] may also happen, degrading ML modules' accuracy or completely stopping an ML module; and (4) after being compromised due to faults or attacks, if not rejuvenated, ML modules eventually enter a non-operational state.

B. Modeling assumptions

To model a perception system with N-version modules and a time-based rejuvenation mechanism, we assume the following:

A.1 faults of an ML module in a healthy state (*H*) are considered dependent with a factor of $\alpha \in [0, 1]$ as assumed in [8]. However, faults of ML modules in a compromised state (*C*) are no more dependent on other modules, as transient faults in this state become random. Also, persistent attacks in this state may completely stop the ML module;

A.2 the voter can be configured with different schemes. When not considering a rejuvenation mechanism, we model the voter needing $2f + 1$ correct outputs to produce a correct perception output. Similarly, we only consider a perception error when at least $2f + 1$ ML modules output incorrectly. In other cases, the output is considered inconclusive but safe, and the voter safely skips the output.

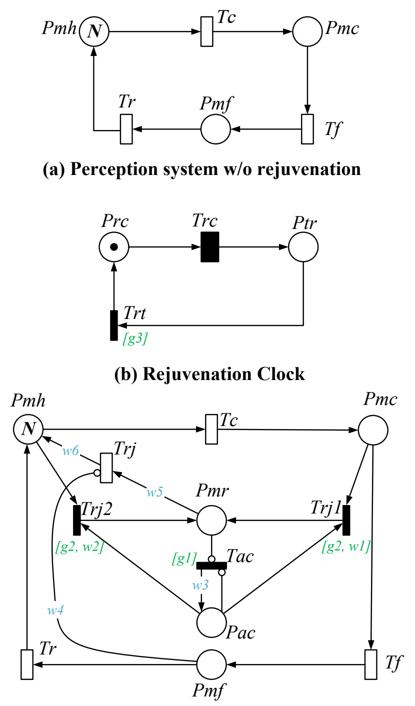
A.3 When considering a rejuvenation mechanism, we model the voter needing $2f + r + 1$ correct outputs to produce a correct perception output. For instance, in a 6-version system where $f = 1$, $r = 1$, and therefore, $n \geq 6$, it is necessary at least $2f + r + 1 = 4$ correct outputs from the ML modules for a request to be classified as correct (e.g., 4-out-of-6 voting). When at least $2f + r + 1$ ML modules output incorrectly, it is considered a perception error. In other cases, the output is considered inconclusive but safe, and the voter safely skips the output.

A.4 we do not consider failures in the voter and rejuvenation clock for the sake of simplicity.

C. DSPN models

We adopt DSPN models to compute the reliability of the N-version ML system and leverage its capabilities to model failure behaviors and a time-based rejuvenation mechanism. Figure 2 (a) presents the DSPN for a system with *N* ML modules subject to faults and attacks but not rejuvenation. The place *Pmh* represents an ML module in a healthy state. The number of tokens *N* in this place represents the number of ML module versions. The exponential transition *Tc* models attacks or faults that partially compromise the accuracy of the ML module. When *Tc* fires, a token is consumed from place *Pmh*, and another is generated in place *Pmc*, representing an ML module not running at total accuracy. The transition *Tf* is enabled by the presence of a token in the place *Pmc*. When *Tf* fires, a token is consumed from place *Pmc*, and another is generated in place *Pmf*, representing the ML module crashed and cannot execute a perception task. An ML module in a non-operational state needs to be repaired. The transition *Tr* represents the repair event. The firing of *Tr* consumes a token from *Pmf* and generates another in *Pmh*, implying that the ML module is fully operating again.

Figures 2 (b) and (c) present a DSPN model for a rejuvenation clock — to trigger the rejuvenation mechanism at pre-defined intervals — and a perception system subject to rejuvenation, respectively. The rejuvenation clock model (Figure 2 (b)) uses a deterministic transition to control the interval in which the ML modules are rejuvenated. A token in place *Prc* enables *Trc*. When *Trc* fires, it consumes a token from place *Prc* and generates another one in place *Ptr*. A token in the place *Ptr* satisfies the guard function *g1*, and the transition *Tac* (in the DSPN of Figure 2 (c)) can fire. Next, when *g3* is satisfied, *Trt* can fire, consuming a token from



(a) Perception system w/o rejuvenation

(b) Rejuvenation Clock

(c) Perception system adopting rejuvenation

Fig. 2: DSPNs for an N-version perception system (a) not considering rejuvenation and (b, c) adopting a time-based rejuvenation mechanism.

TABLE I: Guard functions and arc weights for the DSPN models of Figures 2 (b) and (c).

Transition	Guard/Weight	Enabling Function/Value
Tac	g1	*(#Pac + #Pmr) = 1*
Trj1, Trj2,	g2	*(#Pmf + #Pmr) < r*
Trt	g3	*(#Pmr + #Pac) > 0*
Trj1	w1	*IF (#Pmc = 0): (0.00001) ELSE (#Pmc/(#Pmc + #Pmh));*
Trj2	w2	*IF (#Pmh = 0): (0.00001) ELSE (#Pmh/(#Pmc + #Pmh));*
-	w3, w4	*r*
-	w5	*IF (#Pmr < r): (#Pmr) ELSE (r);*
-	w6	*#Pmr*

place *Ptr* and generating another one in place *Prc*, indicating the clock is reset.

The operation of the perception system adopting rejuvenation (Figure 2 (c)) is similar to the previously explained in Figure 2 (a). It only differs regarding the rejuvenation mechanism, which works as follows. The firing of *Tac* generates *r* tokens in the place *Pac*. Tokens in the place *Pac* can enable either *Trj1* or *Trj2*. The guard function *g2* must be satisfied to fire *Trj1* or *Trj2*. The weights *w1* and *w2* model the firing probability of *Trj1* and *Trj2*, meaning that the system cannot distinguish between healthy or compromised ML modules to rejuvenate. When *Trj1* fires, one token is consumed from places *Pmc* and *Pac*, and one token is generated in place *Pmr*. When *Trj2* fires, it consumes one token from places *Pmh* and *Pac*, generating another token in place *Pmr*. Tokens in the place *Pmr* enable transition *Trj*, indicating ML modules are rejuvenating. The rejuvenation process is complete when *Trj* fires. The firing of *Trj* consumes a maximum number of *r* tokens and generates the same number of consumed tokens in place *Pmh*. The arc weights *w5* and *w6* model this behavior.

D. Reliability functions definition

We define reliability functions for an N-version perception system subject to faults, attacks, and rejuvenation. Unlike the general reliability model proposed by Ege et al. [8], in ML-based systems, an ML module is unlikely to hold its perception accuracy when in a compromised state. Thus, instead of relying on the equivalent failure probability p during the whole system execution, we also define $p'(> p)$ as the output failure probability of an ML module in a compromised state.

Since ML module states change due to attacks, faults, or rejuvenation (see §III), the system's expected reliability depends on the state probability. Defining S as the set of reachable states in the DSPN model presented in §IV-C, we can represent a state of an N-version perception system as $(i, j, k) \in S$ where i, j, and k represent the number of ML modules in healthy, compromised, and non-operational states, respectively. Next, we also define $R_{i,j,k}$ as the output reliability of a perception system with N-version ML modules in a state (i, j, k). Then, we leverage the DSPN models to compute each state's steady-state probability $\pi_{i,j,k}$. In this way, by assigning $R_{i,j,k}$ as the rewards for the individual states, we obtain the expected system output reliability as

$$\mathbb{E}[R_{sys}] = \sum_{(i,j,k)\in S} \pi_{i,j,k} R_{i,j,k} \qquad (1)$$

Next, we derive the reliability functions for a four- and six-version perception system to embed it in the DSPN models for computing the expected reliability.

1) Reliability functions for a four-version system: We model a four-version system without adopting rejuvenation. This system requires at least four ML modules to support one compromised module (i.e., $f = 1$), where $n \geq 3f + 1$. As assumption A.2 explains (see §IV-B), a perception error occurs when three or more ML modules classify a request wrongly. Thus, the system reliability functions are defined only for the states satisfying $k \leq 1$. In this way, we define the reliability function matrix $\mathbf{R_{f4}}$ whose (i, j) element corresponds $R_{i,j,k}, i, j, k \in \{0, 4\}, k = 4 - (i + j)$ if the state fulfills the voting rule, otherwise 0.

$$\mathbf{R_{f4}} = \begin{bmatrix} 0 & 0 & 0 & R_{3,0,1} & R_{4,0,0} \\ 0 & 0 & R_{2,1,1} & R_{3,1,0} & 0 \\ 0 & R_{1,2,1} & R_{2,2,0} & 0 & 0 \\ R_{0,3,1} & R_{1,3,0} & 0 & 0 & 0 \\ R_{0,4,0} & 0 & 0 & 0 & 0 \end{bmatrix}. \qquad (2)$$

By using Eq. 1 and $\mathbf{R_{f4}}$, we can define $\mathbb{E}[R_{4v}]$ as a reward function to the DSPN model of Figure 2 (a) and compute the expected reliability for a four-version system without adopting rejuvenation. Due to space limitations, we detail each reliability function on §A of the Appendix.

2) Reliability functions for a six-version system applying rejuvenation: A system supporting $f = 1$ compromised modules and $r = 1$ modules simultaneously rejuvenating requires at least $n \geq 3f + 2r + 1$ ML modules. Therefore, we model a six-version system that adopts rejuvenation. In this configuration, the system outputs an error when four or more ML modules classify a request wrongly (see assumption A.3 on §IV-B). The system reliability functions are defined only for the states satisfying $k \leq 2$. In this way, we define the reliability function matrix $\mathbf{R_{f6}}$ whose (i, j) element corresponds $R_{i,j,k}, i, j, k \in \{0, 6\}, k = 6 - (i + j)$ if the state fulfills the voting rule, otherwise 0. Each reliability function is expanded on the §B of the Appendix. Finally, the expected reliability for a six-version system adopting rejuvenation $\mathbb{E}[R_{6v}]$ can be defined as a reward function in the DSPN models of Figures 2 (b) and (c) using the $\mathbf{R_{f6}}$ into Eq. 1.

$$\mathbf{R_{f6}} = \begin{bmatrix} 0 & 0 & 0 & 0 & R_{4,0,2} & R_{5,0,1} & R_{6,0,0} \\ 0 & 0 & 0 & R_{3,1,2} & R_{4,1,1} & R_{5,1,0} & 0 \\ 0 & 0 & R_{2,2,2} & R_{3,2,1} & R_{4,2,0} & 0 & 0 \\ 0 & R_{1,3,2} & R_{2,3,1} & R_{3,3,0} & 0 & 0 & 0 \\ R_{0,4,2} & R_{1,4,1} & R_{2,4,0} & 0 & 0 & 0 & 0 \\ R_{0,5,1} & R_{1,5,0} & 0 & 0 & 0 & 0 & 0 \\ R_{0,6,0} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

V. Numerical Experiments

This section presents numerical experiments to demonstrate the models' applicability in computing the expected output reliability of N-version perception systems. We first describe the experimental setup and input parameters adopted. Then, we evaluate and discuss the reliability results of systems with four- and six-version perception systems with or without rejuvenation.

A. Experiments setup

We employed the TimeNET tool [24] to run and analyze the DSPN models. Table II shows the input parameters and their default values adopted in the experiments. It also shows the associated transition of the parameters in the DSPN models.

Most parameters are hard to find in the literature, so we estimate some parameters to demonstrate the models' applicability. We modeled scenarios where: the number of ML modules is $N = 4$ or 6; the safely supported number of compromised ML modules is $f = 1$; and the number of simultaneously recovering or rejuvenating ML modules is $r = 1$. For the error probability dependency between modules (α), we use a default value of 0.5, which means 50% of error dependency. We adopt an average of the inaccuracy of neural networks LeNet, AlexNet, and ResNet that we experimentally used to classify the German Traffic Sign dataset [25] as the inaccuracy of a healthy ML module (p). We estimate

TABLE II: Default input parameters for the DSPN models.

Param.	**Associated Transition**	**Value**
N	-	4 or 6
f	-	1
r	-	1
α	-	0.5
p	-	0.08
p'	-	0.5
$1/\lambda_c$	*Tc*	1523 s
$1/\lambda$	*Tf*	3000 s
$1/\mu$	*Tr*	3 s
$1/\mu_r$	*Trj*	*#Pmr* $\times$ 3 s
$1/\gamma$	*Trc*	600 s

the inaccuracy of a compromised ML module as $p' = 0.5$ since outputs in a compromised state become random. We leverage the results from Oboril et al. [26] and employ their computed mean time between faults (MTBF) as the mean time to compromise/degrade a module ($1/\lambda_c$). The mean time to failure ($1/\lambda$) may depend on specific scenarios, so we reasonably use $2 \times 1/\lambda_c \cong 3000$ for this parameter. Since different hardware configurations may interfere with recovering an ML module, we assume $1/\mu = 3$ seconds for the mean time to repair. The same value is used as the mean time to rejuvenate an ML module ($1/\mu_r$). However, since r ML modules can rejuvenate simultaneously, the final value depends on the number of ML modules that are rejuvenating, thus, $1/\mu_r = \#Pmr \times 3$ seconds. Lastly, we adopt a default value of 600 seconds for the rejuvenation interval ($1/\gamma$), but we also vary this value to investigate its impact on system reliability.

B. Reliability results and discussion

Using the DSPN models of Figure 2, the reliability functions (i.e., $\mathbb{E}[R_{4v}]$ and $\mathbb{E}[R_{6v}]$), and the default input parameters of Table II, we evaluated the expected reliability for each system configuration. The computed expected reliability was 0.8233477 for the four-version (without rejuvenation) and 0.93464665 for the six-version (adopting rejuvenation) using the default input parameters. Therefore, these first results show that using a rejuvenation mechanism would improve the system reliability by about 13%. The better results for the system using the rejuvenation mechanism can be explained mainly by the proactive recovery of compromised ML modules, making the system have a higher number of modules in a healthy state.

Next, we investigate how the rejuvenation mechanism impacts the system's expected reliability using different rejuvenation intervals. We varied the rejuvenation interval ($1/\gamma$) between 200 and 3000 seconds. Figure 3 shows the reliability results achieved. The plot shows that more frequent rejuvenation procedures are better for the six-version system in terms of reliability. Thus, increasing the value of $1/\gamma$ after a certain point would decrease the system's reliability. In this way, knowing the system parameters would be possible to find the optimum rejuvenation interval to maximize the

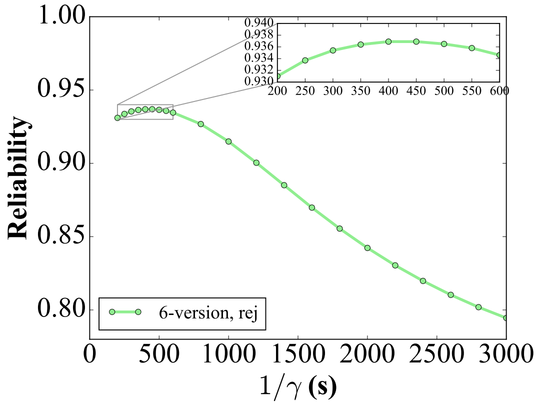


Fig. 3: Influence of the rejuvenation interval over the expected reliability in a six-version perception system.

system's reliability. For instance, considering the adopted input parameters, the maximum reliability is reached for a rejuvenation interval of 400 - 450 seconds.

We also evaluate the influence of other key parameters over the expected reliability through a sensitivity analysis. Figure 4 presents the reliability results when varying the parameters (a) mean time to compromise/degrade a module, (b) error probability dependency between modules, (c) ML modules inaccuracy when in a healthy state, and (d) the ML modules inaccuracy when in a compromised state. Figure 4 (a) shows that both systems behaved as expected, where a higher value of $1/\gamma_c$ would imply higher reliability for the system as it should remain in a healthy state for more time. Additionally, the four-version system without rejuvenation outperforms the six-version system under specific circumstances: (1) when $1/\lambda_c < 525$ seconds, and (2) when $1/\lambda_c > 6000$ seconds. For all other values of $1/\lambda_c$, the six-version system performs better in terms of reliability. It means that when $1/\lambda_c$ is less than the defined rejuvenation interval (in this case, it was $1/\gamma = 600$ seconds), the system probably would not benefit from a rejuvenation mechanism. Besides, when $1/\lambda_c$ is too high compared to the rejuvenation interval, the benefits of a rejuvenation mechanism may be marginal.

Figure 4 (b) shows the results when varying the error dependency between the ML modules (α). The plot shows that when the error dependency is small, it positively impacts the system's reliability, especially when adopting a time-based rejuvenation mechanism. However, the impact of α is small, especially for the four-version system, where the reliability decreases by about 1.5% when varying α from 0.1 (low error dependency) to 1 (total error dependency). The impact is also slight for the six-version system, about 6.6%. When the system adopts a rejuvenation mechanism, the ML modules should spend more time in a healthy state where error dependency exists. Thus, α impacts the system's reliability more when its value is close to 1.

The ML modules' inaccuracy when in a healthy state (p) influence over the reliability is shown by Figure 4 (c). The results reveal that the six-version system presents better reliability than the four-version for all cases. However, the impact is negatively higher on the six-version system, about 13% when varying p from 0.01 to 0.2. On the other hand, the negative impact on the four-version system is only about 5%.

Lastly, we investigated the impacts of the ML modules' inaccuracy when in a compromised state (p') over the reliability (see Figure 4 (d)). The results show that a rejuvenation mechanism could mitigate a higher reliability degradation even when p' is high (e.g., $p' = 0.8$). We understand that p' is hard to estimate or obtain since it depends on different fault or attack types. Therefore, opting for a system with rejuvenation may cover broader scenarios to maintain higher reliability. On the other hand, when obtaining the value of p', verifying whether a rejuvenation mechanism is beneficial would be feasible. For instance, using the input parameters of our numerical experiments, a six-version system adopting rejuvenation is only beneficial when $p' > 0.3$. Otherwise, the four-version system without a rejuvenation mechanism presents better reliability.

VI. Conclusion

This paper investigated the combination of N-version ML and time-based rejuvenation for perception systems. We proposed a set of models and functions to compute the output reliability of perception systems and demonstrated its applicability by evaluating a four- and six-version system. The proposed models consider the misbehavior of ML modules due to faults, malicious attacks, and the effects of using a proactive rejuvenation mechanism. Our results suggested that the output reliability of a perception system could be improved by adopting a rejuvenation mechanism in most scenarios. We also performed a sensitivity analysis on key input parameters to find thresholds where the system output reliability is maximized. In future work, we aim to experimentally analyze our proposed approach in perception and other systems.

Acknowledgment

This work has been supported by the German research council (DFG) and by the Luxembourg Fond Nationale de Recherche (FNR) through the Core Inter Projects ByzRT (C19-IS-13691843) and ReSAC (C21/IS/15741419). This work is supported in part by JSPS KAKENHI Grant Numbers 22K17871.

References

[1] D. Gruyer, V. Magnier, K. Hamdi, L. Claussmann, O. Orfila, and A. Rakotonirainy, "Perception, information processing and modeling: Critical stages for autonomous driving applications," *Annual Reviews in Control*, vol. 44, pp. 323–341, 2017.

[2] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, "Robust machine learning systems: Reliability and security for deep neural networks," in *International Symposium on On-Line Testing And Robust System Design*, 2018.

[3] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," *Applied Sciences*, vol. 9, no. 5, p. 909, 2019.

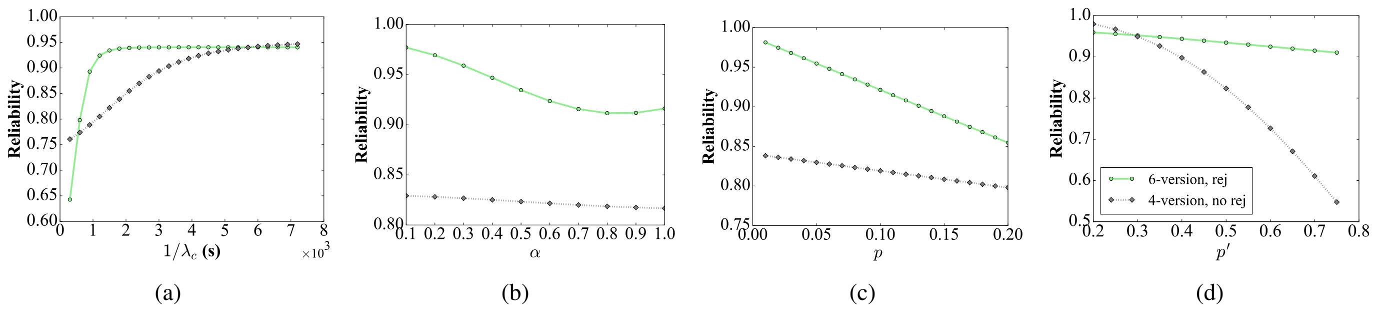


Fig. 4: Influence of the (a) mean time to compromise/degrade a module, (b) error probability dependency between modules, (c) ML modules inaccuracy when in a healthy state, and (d) the ML modules inaccuracy when in a compromised state over the expected system reliability.

[4] National Transportation Safety Board, "Final reports for 2 advanced driver assistance system crash investigations published," 2020. [Online]. Available: https://www.ntsb.gov/news/press-releases/Pages/NR20200319.aspx

[5] M. Berk, O. Schubert, H.-M. Kroll, B. Buschardt, and D. Straub, "Exploiting redundancy for reliability analysis of sensor perception in automated driving vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5073–5085, 2020.

[6] M. Qiu, P. Bazan, T. Antesberger, F. Bock, and R. German, "Reliability assessment of multi-sensor perception system in automated driving functions," in *2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2021, pp. 104–112.

[7] A. Avizienis, "The n-version approach to fault-tolerant software," *IEEE Transactions on software engineering*, no. 12, pp. 1491–1501, 1985.

[8] M. Ege, M. Eyler, and M. Karakas, "Reliability analysis in n-version programming with dependent failures," in *Proceedings 27th EUROMICRO Conference. 2001: A Net Odyssey*. IEEE Comput. Soc, 2001, pp. 174–181.

[9] F. Machida, "N-version machine learning models for safety critical systems," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 6 2019, pp. 48–51.

[10] ——, "Using Diversities to Model the Reliability of N-version Machine Learning System," 2021. [Online]. Available: https://doi.org/10.36227/techrxiv.16435656.v1

[11] Q. Wen and F. Machida, "Reliability models and analysis for triple-model with triple-input machine learning systems," in *IEEE Conference on Dependable and Secure Computing (DSC)*, 2022.

[12] S. Latifi, B. Zamirai, and S. Mahlke, "Polygraphmr: Enhancing the reliability and dependability of cnns," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[13] A. Gujarati, S. Gopalakrishnan, and K. Pattabiraman, "New wine in an old bottle: N-version programming for machine learning components." Institute of Electrical and Electronics Engineers Inc., 10 2020, pp. 283–286.

[14] H. Xu, Z. Chen, W. Wu, Z. Jin, S.-y. Kuo, and M. Lyu, "Nv-dnn: Towards fault-tolerant dnn systems with n-version programming," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2019.

[15] M. Voelp and P. Esteves-Verissimo, "Intrusion-tolerant autonomous driving," in *IEEE International Symposium on Real-Time Computing*, 2018.

[16] J. Bai, X. Chang, G. Ning, Z. Zhang, and K. S. Trivedi, "Service availability analysis in a virtualized system: a markov regenerative model approach," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 2118–2130, 2020.

[17] A. M. M. Paing, "Analysis of availability model based on software aging in sdn controllers with rejuvenation," in *IEEE Conference on Computer Applications (ICCA)*, 2020.

[18] K. S. Trivedi and K. Vaidyanathan, "Software reliability and rejuvenation: Modeling and analysis," in *Performance Evaluation of Complex Systems: Techniques and Tools*, M. C. Calzarossa and S. Tucci, Eds., 2002, pp. 318–345.

[19] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99, 1999, p. 173–186.

[20] D. S. Silva, R. Graczyk, J. Decouchant, M. Völp, and P. Esteves-Verissimo, "Threat adaptive byzantine fault tolerant state-machine replication," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, 2021, pp. 78–87.

[21] P. Sousa, N. F. Neves, and P. Veríssimo, "Proactive resilience through architectural hybridization," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06, 2006, p. 686–690.

[22] M. Marsan and G. Chiola, "On petri nets with deterministic and exponentially distributed firing times," in *Advances in Petri Nets 1987*, 1987, vol. 266, pp. 132–145.

[23] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, 2019.

[24] A. Zimmermann, "Modelling and performance evaluation with timenet 4.4," in *Quantitative Evaluation of Systems*, N. Bertrand and L. Bortolussi, Eds. Cham: Springer International Publishing, 2017, pp. 300–303.

[25] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: A multi-class classification competition," in *The 2011 International Joint Conference on Neural Networks*, 2011.

[26] F. Oboril, C. Buerkle, A. Sussmann, S. Bitton, and S. Fabris, "Mtbf model for avs - from perception errors to vehicle-level failures," in *IEEE Intelligent Vehicles Symposium (IV)*, 2022.

Appendix

A. Reliability Functions for the four-version model

This section details the reliability functions for a four-version system, where $f = 1$ and $n = 4$. Considering (i, j, k) as the perception system states with i healthy, j compromised, and k non-operational ML modules, the reliability functions are defined only for the states satisfying $k \leq 1$. When $k > 1$, the voter can not have $2f + 1$ perception outputs from the ML modules about a request, which leads to safely skipping the request or a perception error. Therefore, the reliability in these states is 0.

$$\begin{aligned}
R_{4,0,0} &= 1 - [p\alpha^3 + 4p\alpha^2(1-\alpha)] \\
R_{3,1,0} &= 1 - [p\alpha^2 + 3p\alpha(1-\alpha)p'] \\
R_{3,0,1} &= 1 - p\alpha^2 \\
R_{2,2,0} &= 1 - [pp'^2 + 2p\alpha p'(1-p')] \\
R_{2,1,1} &= 1 - p\alpha p' \\
R_{1,3,0} &= 1 - [p'^3 + 3pp'^2(1-p')] \\
R_{1,2,1} &= 1 - pp'^2
\end{aligned}$$

$$R_{0,4,0} = 1 - [p'^4 + 3p'^3(1-p')]$$
$$R_{0,3,1} = 1 - p'^3$$

B. Reliability Functions for the six-version model

This section presents the reliability functions for a six-version system subject to rejuvenation, where $f = 1$, $r = 1$, and $n = 6$. When considering (i, j, k) as the perception system states with i healthy, j compromised, and k non-operational or rejuvenating ML modules, the reliability functions are defined only for the states satisfying $k \leq 2$.

$$R_{6,0,0} = 1 - [p\alpha^5 + 6p\alpha^4(1-\alpha) + 15p\alpha^3(1-\alpha)^2]$$
$$R_{5,1,0} = 1 - [p\alpha^4 + 5p\alpha^3(1-\alpha) + 10p\alpha^2(1-\alpha)^2 p']$$
$$R_{5,0,1} = 1 - [p\alpha^4 + 5p\alpha^3(1-\alpha)]$$
$$R_{4,2,0} = 1 - [p\alpha^3 p'^2 + 2p\alpha^3 p'(1-p') + 4p\alpha^2(1-\alpha)p'^2 + 8p\alpha^2(1-\alpha)p'(1-p') + 6p\alpha(1-\alpha)^2 p'^2]$$
$$R_{4,1,1} = 1 - [p\alpha^3 + 4p\alpha^2(1-\alpha)p']$$
$$R_{4,0,2} = 1 - p\alpha^3$$
$$R_{3,3,0} = 1 - [p\alpha^2 p'^3 + 3p\alpha^2 p'^2(1-p') + 3p\alpha(1-\alpha)p'^3 + 3p\alpha^2 p'(1-p')^2 + 9p\alpha(1-\alpha)p'^2(1-p') + 3p(1-\alpha)^2 p'^3]$$
$$R_{3,2,1} = 1 - [p\alpha^2 p'^2 + 2p\alpha^2 p'(1-p') + 3p\alpha(1-\alpha)p'^2]$$
$$R_{3,1,2} = 1 - p\alpha^2 p'$$
$$R_{2,4,0} = 1 - [p\alpha p'^4 + 4p\alpha p'^3(1-p') + 2p(1-\alpha)p'^4 + 6p\alpha p'^2(1-p')^2 + 8p(1-\alpha)p'^3(1-p') + p(2-\alpha)p'^4]$$
$$R_{2,3,1} = 1 - [p\alpha p'^3 + 3p\alpha p'^2(1-p') + 2p(1-\alpha)p'^3]$$
$$R_{2,2,2} = 1 - p\alpha p'^2$$
$$R_{1,5,0} = 1 - [p'^5 + 5p'^4(1-p') + 10pp'^3(1-p')^2]$$
$$R_{1,4,1} = 1 - [p'^4 + 4pp'^3(1-p')]$$
$$R_{1,3,2} = 1 - pp'^3$$
$$R_{0,6,0} = 1 - [p'^6 + 6p'^5(1-p') + 15p'^4(1-p')^2]$$
$$R_{0,5,1} = 1 - [p'^5 + 5p'^4(1-p')]$$
$$R_{0,4,2} = 1 - p'^4$$