IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 19, 2021, accepted July 21, 2021, date of publication July 26, 2021, date of current version August 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3100125

Statistical Test Coverage for Linux-Based Next-Generation Autonomous Safety-Related Systems

IMANOL ALLENDE[1], NICHOLAS MC GUIRE[2], JON PEREZ-CERROLAZA[1], (Senior Member, IEEE), LISANDRO G. MONSALVE[1], JENS PETERSOHN[3], AND ROMAN OBERMAISSER[4]

[1]Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), 20500 Arrasate, Spain
[2]OpenTech EDV Research GmbH, 2193 Wilfersdorf, Austria
[3]Elektrobit, 91058 Munich, Germany
[4]Department of Electrical Engineering and Computer Science, University of Siegen, 57076 Siegen, Germany

Corresponding author: Imanol Allende (iallende@ikerlan.es)

ABSTRACT Autonomous systems represent a significant leap forward in the ongoing technological evolution of dependable and safety-related systems, integrating features such as artificial intelligence, high-performance computing devices, General Purpose Operating Systems (GPOS) (e.g., GNU/Linux) and security requirements. Nonetheless, traditionally employed safety techniques and measures were not defined for safety-related systems with such features. Consequently, the need to research new methods and measures emerges in order to be able to achieve appropriate safety assurance. In this manuscript, we explore the limitations of traditional test coverage techniques, and we provide two complementary methods to pave the way towards the testing of Linux-based complex safety-related systems. The methods, which are based on statistical analyses, are presented and applied to a Linux-based Autonomous Emergency Braking (AEB) case study, specifically focusing on the kernel execution path test coverage.

INDEX TERMS Autonomous systems, Linux, risk, safety, statistics, test coverage.

I. INTRODUCTION

Next-generation autonomous systems represent a breakthrough for different industrial sectors and technological domains. These state-of-the-art systems incorporate groundbreaking technologies that create novel use-cases and incorporate a higher level of autonomy to the existent ones. Among these technologies, we can find, for instance, Artificial Intelligence (AI) algorithms. Nonetheless, the complexity of these systems has increased significantly over the last decade, with requirements such as high computing performance and accelerators, remote software updates, and security. Besides, autonomous systems are currently even being deployed in use-cases with functional safety requirements (e.g., autonomous vehicles) [1], where a system failure can lead to a catastrophe (e.g., loss of human lives), entailing a significant shift for dependable and safety-related domains. Besides, the complexity growth and the use of new technologies bring multiple challenges to the functional safety domain and, hence, hinders the safety assurance of these systems.

The autonomous systems that we are attempting to develop today entail a significant shift from classical safety systems. While traditional approaches were successful and effective for the classic safety systems for which they were defined, autonomous systems have features that limit the applicability of traditional techniques and approaches [2]. Although companies are announcing the development of highly complex dependable systems, there is still a need to pave the way towards the assurance of such dependable and safety-related complex systems. Unfortunately, no safety standard exists that takes into account the specific features of such complex systems. The techniques and measures recommended by current safety standards were not designed and intended for these types of systems (see Section II). So, the update of current safety standards and the definition of new types of safety standards are in progress (e.g., ISO/PAS 21448:2019 Road vehicles — Safety of the intended functionality (SOTIF) [3]).

The ideal for a classical safety-related system is to (i) know all failure modes, (ii) understand all failure conditions,

The associate editor coordinating the review of this manuscript and approving it for publication was Luca Cassano.

and (iii) have adequate evidence to justify the assumptions. Nevertheless, this is generally considered not technically feasible for next-generation autonomous safety systems [4]. Consequently, for complex software elements, an alternative approach is necessary to achieve adequate assurance.

To pave the way towards the safety assurance of these complex systems, we focus our efforts on analyzing the test coverage of these systems, specifically of the Operating System (OS). One of the most significant transformations of these safety-related systems is the need for an OS that meets, among others, the features mentioned above. Linux is the leading OS in different domains [5] and provides much of the features required by the next-generation safety-related systems (e.g., computing performance, concurrent computing, security, updating capabilities) [6]. Therefore, different initiatives aim to achieve an appropriate safety assurance of a Linux kernel subsystem to be the operating system of these safety-related systems [7]–[9]. Two of the most renowned initiatives are OSADL's SIL2LinuxMP project [10], [11] and Linux Foundation's ELISA project [8]. Companies of different industrial domains participate in both projects, as there is a remarkable interest in achieving the certification of Linux for safety-related systems. However, the kernel has not been developed with functional safety in mind. Furthermore, the kernel size and complexity do not facilitate test coverage measures required by safety standards. Hence, a question that may arise is: Why Linux?

Companies and governments rely upon Linux even for crucial applications (e.g., banking, telecommunications). Governmental Agencies, such as the Defense Information Systems Agency (DISA) of the United States, use Linux for their systems with cybersecurity requirements [12]–[14]. Space domain also relies on Linux for a wide range of their systems [15]. For instance, SpaceX relies on Linux for their primary flight control systems on the Falcon 9 launch vehicle and Dragon spacecraft. [15]–[17]. Linux can even be found on Mars. Together with the Perseverance rover, NASA deployed on the surface of Mars a Linux-based drone (i.e., Ingenuity) [18]. Consequently, we can state that Linux is already increasingly relied upon for mission-critical systems.

This manuscript extends previous research works [19], [20] that describe preliminary statistical test coverage analysis methods for Linux-based safety-related systems, with a simple but reproducible case study. The current publication provides two complementary contributions in the area of statistical test coverage analysis. On the one hand, we describe a statistical analysis method that estimates the test coverage with the remaining uncertainty. In other words, the described method statistically estimates the maximum number of kernel execution paths that an application can exercise in order to calculate the number of not-covered or untested execution paths. On the other hand, to advance in the estimation of the residual risk of these systems as a result of the untested or not-covered paths, we examine a method that calculates the execution probability of the not-covered paths. The R scripts that has been developed and used to conduct the research activities are publicly available in a Git repository [21]. The repository also collects the data obtained from the case study.

The remainder of this publication is organized as follows. Section II provides a summarized problem statement in the area of test coverage of Linux-based complex safety-related systems. Section III collects the literature related to the analyses that are presented in this manuscript. Then, Section IV introduces the case study and the data set acquisition used as guiding example for the proposed methods. Section V presents the statistical test-coverage analysis method, using the previous case study's data as a guiding example. Section VI proposes a method to estimate the execution probability of not-covered execution paths, which complements the previous statistical test-coverage analysis method. Finally, Section VII draws the obtained conclusions and includes an outlook on future work.

II. PROBLEM STATEMENT

The safety domain has traditionally considered testing as one of the main methods to establish the safety system correctness evidence. IEC 61508-4 Ed 2 defines dynamic testing as running software and/or hardware in a controlled mode to demonstrate the presence of the required behavior and the absence of unwanted one [22]. This manuscript considers IEC 61508 the reference standard for the conducted research with the aim of extending the contribution to other safety domains. Besides, IEC 61508 and ISO 26262 (automotive) defines equivalent testing techniques [22], [23]. The results obtained from these methods have been considered adequate assurance for testing classical safety-related applications. However, with the increasing complexity of the software in applications such as next-generation autonomous systems, relevant test coverage is hardly achievable (if feasible) by dynamic testing and, thus, there is a need to define novel complementary methods and approaches [20]. This also affects the qualification of the Linux kernel due to distinct factors:

1) *Total Existing Paths:* The Linux kernel can be considered quite large due to the breadth of supported hardware and the number of resources it offers. The latest kernel versions have approximately 27 million Lines Of Code (LOC) [24]. Although all these lines do not form a Linux-based system (e.g., it does not use all the drivers available in the kernel), a Linux-based safety-related system is still estimated at around 1 million LOC [7].
   Therefore, it is considered technically unfeasible to get 100% execution paths tested as some of them are not even exercised by the given application in the target system [20]. Even achieving a reduced while justifiable coverage[1] seems questionable.
2) *Execution Path Variability:* Different studies show how the Linux kernel execution is non-deterministic [25]–[29]. This means that the kernel does not follow the same execution path, given the same application

[1] IEC 61508-3 Ed 2 Annex B.2 permits $\leq$ 100% if justified

with identical input parameters. In other words, for an identical system-call with the same input parameters, several possible execution paths may exist. Consequently, it is no longer feasible to assess the correctness of this type of systems with a unique iteration of a test-case for each combination of input parameters [30]. Besides, it is challenging (if feasible) to force the execution of a specific path.

We can consider that the Linux kernel is formed by several interdependent state machines, which are asynchronous between them, and thus, execution paths are dependent of uncontrolled system states [20]. Simplifying, the system state can be considered an input to the function, but generally not reproducible.

3) *Existing Static Analysis Tools:* have traditionally been used to identify all possible execution paths. Although these types of tools are widely used for direct call examination, their applicability for kernel execution path analysis is limited. For instance, these tools have significant difficulties in solving indirect calls [31], which are commonly used in the kernel, and also provide traces formed by dead code that are never executed [32]. There is also the issue of dynamic allocation and management of resources that may lead to almost infinitely deep paths that are theoretically possible but increasingly unlikely and notably impossible to deterministically trigger. Thereby, these types of tools also provide traces with a defacto zero or negligible probability of execution.

Due to these limitations, an updated approach needs to be defined. Dynamic analysis tools could be used instead of static analysis tools [32]. Dynamic analysis is performed during system-running time and, hence, allows recording the actual traces during execution. Using this approach, it is possible to identify the path variability by repeated execution, and it does not share the limitations of static tools (e.g., indirect calls).

In the case of the Linux kernel, dynamic analysis tools record the execution traces that an application exercises at the kernel-level, on behalf of a user-space task. For that purpose, the FTrace tool, a tracing tool that allows recording the function call sequences, can be used. Besides, IEC 61508 classifies as Highly Recommended (HR) structural test coverage for entry points and statements, which is equivalent to path coverage. This implies that there is no need to record inline functions, and consequently, FTrace is a suitable tool for recording the execution traces. However, as the main limitations of non-determinism remain, certain questions need to be also answered for dynamic tracing.

- If we identify a subset of the existent traces while the kernel is exercised repeatedly by an application, how do we know the total number of traces?
- Thus, how do we quantify the test coverage?
- How do we know the risk related to the untested (unknown) paths?

In this publication, we try to answer these questions by presenting and analyzing two complementary methods to pave the way towards acceptable safety assurance, explicitly focusing on the test coverage of Linux-based safety-related systems. On the one hand, we propose a statistical analysis that allows quantifying the coverage of the kernel paths taking as full coverage reference the paths that have a relevant probability of occurrence. In other words, the method does not consider full coverage of all possible execution paths but rather the paths with a relevant probability of occurrence. On the other hand, we propose a statistical method that estimates the execution probability of the not-covered/untested kernel paths with the aim of quantifying the risk associated with these paths ($risk = probability * severity$). Both of these methods are presented and analyzed in the context of a Linux-based Autonomous Emergency Braking (AEB) case study. Note that the case study is a guiding example for the methods, which are potentially extendable to other case studies [19], [20].

III. RELATED WORK

The available literature shows that the technical challenges for the safety assurance of complex safety-related systems have been extensively examined. Among others, we find studies in the field of high computing performance multi-core devices [33], machine learning algorithms [34], remote software updating [35], and isolation architectures [36]. There are also a significant number of studies that rely on probabilistic or statistical methods to advance in certain aspects of safety certification, such as Measurement-Based Probabilistic Timing Analysis (MBPTA) [37], deep-learning uncertainty [38]–[40], and scheduling [41].

IV. CASE STUDY - DATA SET ACQUISITION

This Section describes the case study and the data set acquisition used as a guiding example in the description of the proposed methods for analyzing test coverage with the remaining uncertainty (Section V) and estimating the execution probability of untested paths (Section VI). The case study is a research-grade AEB system based on Linux kernel and Machine Learning (ML) algorithms and representative of next-generation safety-related systems. Research activities, and therefore the data collection, are focused on the AEB system's control unit, where the Linux kernel runs on a defined system-context.

A. EXPERIMENT SETUP

The study is performed exercising a given application in a specific target platform and system-context. Hence, we need to define these three components for further analysis. For this publication, we assume the following definitions:

- *Target Platform:* The selected platform is an Nvidia Jetson Nano with a quad-core ARM Cortex-A57 processor and a Maxwell Graphics Processing Unit (GPU) architecture of 128 cores. The object detection algorithms are

executed on the GPU cores while the control unit runs in the CPU cores.

- *Application:* The analysis is performed with an AEB case study. The application brakes or alerts (depending on the criticality and proximity) when a *pedestrian* or an *obstacle* is detected. The objective of the application is to reduce the number of accidents or, at least, the severity of the unavoidable ones.
  - The system runs a Linux version 4.9.
  - The platform is configured with a Layers of Protection Analysis (LOPA) architecture, isolating the control task (safety task) on a dedicated processor. More detailed information about the LOPA architecture can be found in the following publication [36].

  The research activities are entirely focused on the control unit running the Linux kernel, especially in testing the kernel execution paths. The rest of the units (i.e., perception understanding (GPU), vehicle control) are considered out of the scope of this research, as they are not based on Linux and they are replaced by a stress benchmark (i.e., hackbench) that simulates the Worst Case Scenario (WCS) described in the system-context.
- *System-Context:* refers to the context in which the system will operate and, therefore, the cases to be tested. Thus, the system-context is a series of test-cases. To perform the analysis thoroughly, we have decided to use an ego-vehicle point-of-view video with several critical scenarios where the AEB system's control unit must take different decisions.
  Besides, to simulate the WCS of the system, one of the cores is heavily loaded to stress the system. This heavy CPU load and interruptions are performed with the widely used *hackbench* stress benchmark. The aim is to accelerate the observation of previously unrecorded traces [29]. Note that the LOPA architecture should reduce the interference significantly [36].
- *Recording:* Ftrace tool is used to record kernel traces. This tool, included in the mainline kernel, allows saving the execution traces at the kernel level that a specific application has exercised. From a research perspective, the FTrace tool is considered a reliable tool due to its widespread usage by the kernel developer community. However, the need for tool qualification (T3 class tool [22]) should be considered for future industrial safety system testing activities.

Figure 1 illustrates the architecture of the AEB system's control unit that is exercised for the presented research activities. The control unit runs the Linux kernel, and it is based on the LOPA safety architecture described by the SIL2LinuxMP project [7], [36]. This architecture uses Linux's container technology, widely used in security and server domains, in order to isolate different criticality tasks. The software application is based on four software partitions, executed each of them in one CPU core. The SIL2 task, which is executed redundantly in two cores, is the safety-related task that is analyzed in this research. The SIL0 task is dedicated to the Human Machine Interface (HMI) (warnings to the driver) and Monitoring is in charge of diagnosing both SIL2 tasks. Furthermore, the GPU executes the perception understanding. The case study uses Ethernet to communicate with other vehicle control subsystems such as the speed and brake controls.

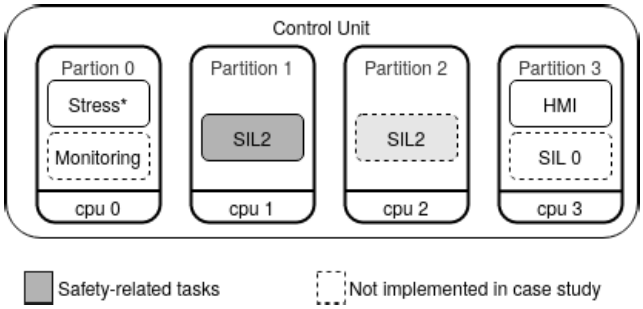

FIGURE 1. AEB case study: control unit architecture.

In order to provide a preliminary assessment and identify the potential of the methods described in this manuscript, the research activities entirely focus on the testing of the safety function in its WCS. For this purpose, certain functions, which are considered non-essential from the safety testing perspective, have been replaced by a stress benchmark. Therefore, we adapt Partition 0 to simulate a WCS by means of the *hackbench* stress benchmark. Figure 1 shows with dashed line boxes the functions that have been replaced by the benchmark. If the results show a strong potential of the described methods, future extended research should consider the incorporation of all the functions.

It is worth mentioning that assessing the suitability of this AEB system for a real autonomous car is out of the scope of this publication. Rather than that, the intention is to use it for our study, as it provides some building blocks that are common in a commercialized vehicle.

B. DATA SET RECORDING

For this study, we focus on recording the kernel execution traces that exercise the safety-related (SIL2) task. The application requests the utilization of the kernel (or the resource it provides) through different system-calls. System-calls are considered the entrance function to the kernel. Therefore, the work described in this manuscript is based on the recording of all system-calls requested by the application. Furthermore, the identification and classification of the execution traces are performed through a hash function (e.g., MD5). Therefore, each system-call trace results in a hash value. Designating a hash value to each specific execution trace sequence facilitates data analysis. It allows estimating the execution frequency of each specific trace and also identifying the appearance of a trace that has not previously been executed. Consequently, we can consider different hash values as different *unique-traces*. Figure 2 shows an overview diagram of the data collection and post-processing process.

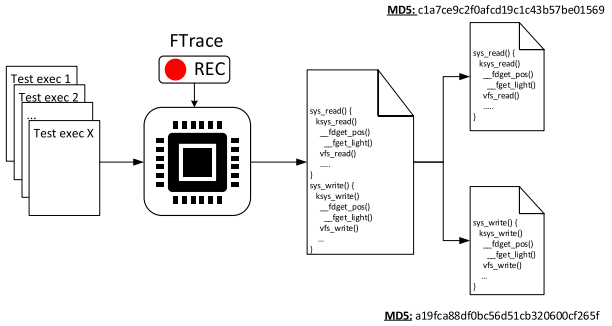

FIGURE 2. Data collection overview diagram.

These traces are not the complete records of exercising the kernel per se, as we only record the system-calls. Although the system-calls represent the vast majority of the execution trace, there are a few function calls between system-calls. Therefore, it cannot be considered a completely proper verification of the kernel but rather a contribution to the verification of the kernel interface. However, if the method shows effectiveness for the kernel interface, it could be expanded to the overall kernel.

TABLE 1. Results of the recorded data set divided in system-calls.

System-calls	Number of executions	Number of different traces	Common trace (%)
clone	261554	1303	34.28
futex(wait)	294963	13	89.07
futex(wake)	332221	6	88.68
ppoll	1095756	21	73.83
read	330574	41	40.36
recvmsg	1979515	33	86.76
sendto(server)	293570	134	29.84
sendto(hmi)	291786	178	94.16
write	602494	112	98.36
writev	283491	231	46.08
TOTAL	5765924	1196	NA

Table 1 collects the results of the recorded data set. Recording this data has involved an effort of exercising the system for more than six thousand hours. The recorded data is also available in the following repository [21]. Table 1 lists the system-calls that are exercised by the application, the number of times that the system-calls have been called, the number of different traces that have been executed in each case (also known as *unique-traces*), and, finally, the proportion of times that the most common paths have been executed. Note that we define *unique-trace* to each different function call sequence. In other words, two execution traces with the same function call sequence belong to the same *unique-trace*. As a result, the AEB system has exercised a total of 1996 different unique-traces.

There are repeated system-calls but with different inputs (e.g., futex(wait), futex(wake)). This is because they are called with different input parameters. For example, the same application can open a file or a device, but the traces may have nothing to do with each other. If we inspect the execution frequency of the distinct traces, it is possible to observe that in each system-call, there are a reduced number of traces that are the ones commonly executed. Consequently, we can state that each system-call can follow different execution paths, but the execution probability of them is not equiprobable. Figure 3 illustrates in a logarithmic scale how there are a small number of traces that are executed with a significantly higher frequency. The shown data corresponds to the entire data set, with all test-campaigns and system-calls. Therefore, each different execution path has its own execution probability.

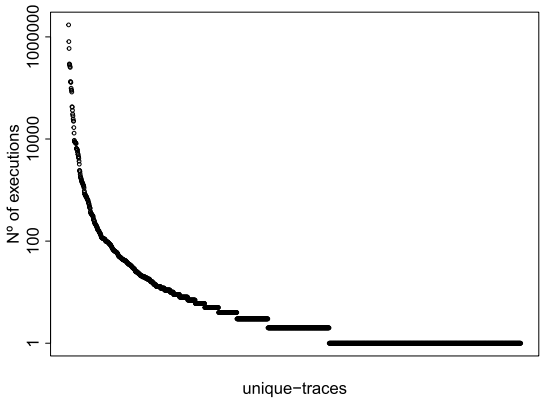

FIGURE 3. Execution frequency of the different traces.

The traces that a system-call executes can be classified into two groups: common traces and rare-traces. If rare-traces are inspected, it is possible to identify that they are variations of the most common traces. They are mainly built by having the most common trace as trunk and with a series of calls executed at different points of the trace due to variations of the global state of the system, which are not under the control of the application. Although the execution probabilities of these variations are lower than the probability of the common trace, each of the variations has a different execution probability. Figure 4 illustrates an example of a *common* and a *rare-trace* of system-call write. The beginning of the traces is the same; however, there is a point where the rare-trace executes a branch that starts with *rt_spin_lock_slowunlock()*. After the branch, both traces continue executing the same trace until the end. Furthermore, as Table 1 shows, there are a few system-calls, such as *read()*, where their common path is not so commonly executed. However, this is due to the existence of several common paths. For example, in *read* system-calls the second most common path is executed 39.51% of the time.

Note that the total number of different traces is not the sum of the second column's results. In this case study, we identified equivalent traces between *sendto(server)* and *sendto(hmi)*. Besides, the total value of *'Common trace (%)'* is Not Applicable (NA) as the value is only representative with respect to each system-call.

V. METHOD 1: TEST COVERAGE UNCERTAINTY QUANTIFICATION

The objective of the method described in this section is to: quantify the number of traces that have not yet been observed but are to be expected and, thus, quantify the uncertainty. In other words, the goal is to estimate the proportion of paths that the application exercises compared to the totality of paths

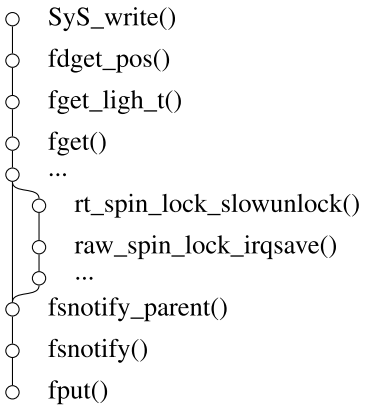

FIGURE 4. Execution trace example (system-call write) with two possible paths: common-trace (short) and rare-trace (branched).

that have a relevant probability of occurrence. Note that this is very much distinct from the paths that could theoretically be reached (e.g., think of theoretically possible infinite retries in some cases).

For test coverage uncertainty quantification, parametric and non-parametric statistical approaches can be considered. The work reported in this section is based on the publication [36], where a parametric approach was defined. Nevertheless, in this research, we examine the suitability of non-parametric estimators. Besides, this manuscript provides a higher analysis detail as the AEB case study is more representative than the simpler case study described in [19], [36].

A. DESCRIPTION OF THE APPROACH

Our approach is based on the species estimation techniques available in the literature. For years, different statisticians and biologists have investigated the estimation of the number of species [42], [43]. Estimating the species richness (number of different species) is a complex task as the number of observed species increases with every increment of the sampling effort [44], [45]. These research contributions did not initially target technology domains, even less functional safety. However, the literature collects different non-parametric estimators that could be appropriate for our research [42], [43], [46], [47].

In this case, the traces are identified as the system is exercised (dynamic analysis). Therefore, we do not know all the traces that can be exercised with a reasonable probability. Suppose each different execution path is considered a 'species'. In that case, it is possible to employ the non-parametric species estimators to estimate the number of traces that have a relevant probability of appearing. Therefore, the study is based on one primary assumption:

- *Assumption:* the number of new execution traces (i.e., different from those that occurred previously) will increase as the number of test iterations increases. If the cumulative number of traces is plotted, it will result in a *species accumulation curve*, where there will be a moment that the curve approaches the asymptote. The asymptote represents the moment when all the possible execution traces have been exercised and, consequently, it represents 100% of test coverage.

Since reaching the asymptote with testing techniques is a potentially not feasible task, the aim is to estimate the total number of traces analyzing the recorded rare-traces. Chao *et al.* state that events with a lower frequency of occurrence provide more information for species richness [42], [48]. After all, events that occur commonly provide almost no information about rare events [48].

B. TRACES ACCUMULATION

The accumulation of new traces can be examined while the number of test-campaigns increases. Accumulation curves show the tendency to encounter new species as the sample size increases. It is possible to estimate the accumulation curve based on a sample-based species frequency data set [49]. For this purpose, we calculate the mean and the standard deviations of the accumulation curve by sub-sampling the data set without replacement. This method is also known as *Random method* and allows obtaining the accumulation curve without depending on the order in which the traces have been recorded.

Figure 5 depicts the accumulation curve obtained from the recorded data set. The curve shows the traces richness by sampling effort with 95% Confidence Interval (CI). As it is possible to observe, the increase of the curve is more pronounced at the beginning while the trend is to grow more slowly afterward. However, it is possible to confirm that the data set obtained has not yet reached the asymptote. In other words, if we were to continue collecting data, we would still see a significant number of new traces. Once the asymptote is reached, one could say that it is relatively difficult to identify any new trace that has not been executed previously.

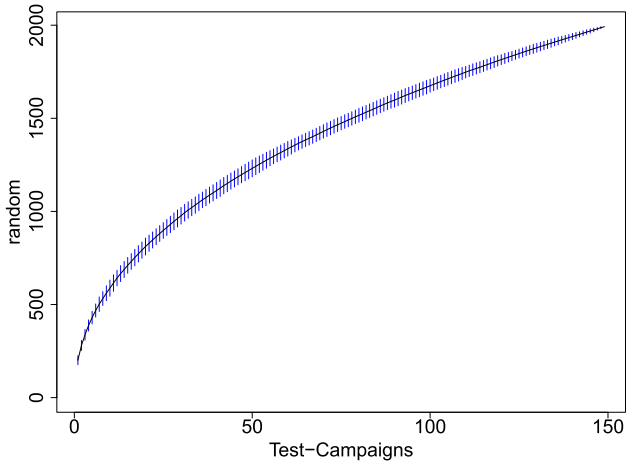

FIGURE 5. Accumulation curve estimated with the random method with 95% CI.

C. DATA SET VALIDATION

Even though the traces are collected by differentiating the system-calls, this method performs the analysis with the data formed by all the system-calls. This allows obtaining the test

coverage of the application under test. However, to carry out the study with all the system-calls together, it is necessary to validate some collected data requirements.

1) CROSS VALIDATION

Cross-validation is a model validation technique that assesses the independence of results of statistical analysis. Therefore, it allows evaluating the stability of the non-parametric results with the incorporation of all system-calls to the analysis. K-Fold Cross Validation analyses if all system-calls follow equivalent new trace appearances and, hence, ensures the steadiness of the results of the statistical study. In other words, it allows confirming that all the system-calls belong to the same population. K-Fold is based on examining the Mean Squared Error (MSE) of the accumulative curve with different combinations (round-robin) of the identified group of traces (folds) [50].

The aim is to know if different system-calls belong to the same population. Therefore, each fold is formed by the traces of a specific system-call. Nonetheless, as Table 1 indicates, each system-call provides a different number of unique-traces. As each fold needs to have the same length, the analysis is performed by dividing some system-calls into different folds (e.g., clone) and joining some of them into one fold (e.g., futex(wait), futex(wake)).

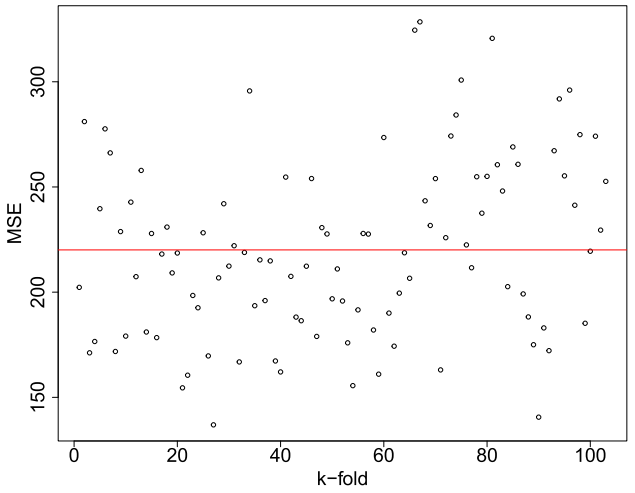

FIGURE 6. MSE values with k equal to 100. Horizontal line represents the mean value.

Figure 6 depicts the MSE results of each fold, with a total of 100. Performing the analysis with 100 folds allows having folds formed with 20 traces and, thus, represented more adequately the system-calls with a small number of traces. The fold order follows the same as Table 1, the first folds represent *clone()* data and the last folds to *writev()*. The obtained MSE are adequate to validate that the system-calls come from the same population and, thus, it is possible to follow the analysis.

D. ESTIMATION OF NUMBER OF PATHS

The accumulation curve shows the tendency to find new traces while testing iterations continue. Nevertheless, it does not estimate the richness of the traces, i.e., the number of traces that an application can exercise in the kernel. The literature describes several non-parametric estimators [46], [47], [51], [52]. The most commonly used estimators can be classified into two types [53]:

- *Incidence-Based Estimators:* considers the number of test-campaigns in which each unique-trace has been executed. For instance, how many traces have been executed only in one test-campaign? In two test-campaigns?
- *Abundance-Based Estimators:* considers the number of times that each unique-trace has been executed. For instance, how many traces have been executed only once? Twice?

Note that *unique-trace* is the equivalent to specie in this research.

Figure 7 illustrates the classification of several non-parametric estimators that are analyzed through this Chapter.

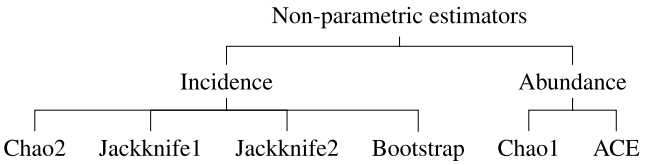

FIGURE 7. Relation among non-parametric estimators.

All the identified estimators (Figure 7) employ the less frequent traces (rare-traces) to estimate the richness as the majority of the information is provided by these traces [42].

1) INCIDENCE-BASED ESTIMATORS

Chao2 estimator based on incidence takes into account traces that execute only in one test-campaign (i.e., uniques), the number of traces that occur in two test-campaigns (i.e., duplicates), and the total number of traces recorded among all the recorded test-campaigns [54]. Equation 1 provides a bias-corrected form for the case that there are no *duplicates* [46], [52].

$$S_P = \begin{cases} S_0 + \dfrac{a_1(a_1-1)}{2(a_2+1)}\dfrac{N-1}{N} & \text{if } a_2 = 0 \\ \\ S_0 + \dfrac{a_1^2}{2a_2}\dfrac{N-1}{N} & \text{if } a_2 > 0 \end{cases} \quad (1)$$

- S_P is the total richness of the traces (observed + non-observed).
- S_0 is the number of traces presented in the recorded data set.
- N is the number of test-campaigns.
- a_1 represents the number of traces that are executed only in one test-campaign (i.e., uniques).
- a_2 represents the number of traces that are executed in two test-campaigns (i.e., duplicates).

There also estimators based on jackknife and bootstrap statistical techniques. Jackknife is a resampling method that recalculates the estimator leaving out sub-sets from the data

set successively and then calculates the average value. Bootstrap is a widely used statistical resampling method [50]. It is a test based on random sampling with replacement. Therefore, it allows generating many *"new"* data sets by randomly sampling with replacement from the original data set.

First order Jackknife estimator is based on the fact that we have unseen as many traces as the ones that appear on one test-campaign [43], [55].

$$S_P = S_0 + \frac{a_1(N-1)}{N} \tag{2}$$

Jackknife2 depends on the traces that are executed in one and two test-campaigns (i.e., uniques and duplicates) in the recorded data set [52].

$$S_P = S_0 + \frac{a_1(2N-3)}{N} - \frac{a_2(N-2)^2}{N(N-1)} \tag{3}$$

Bootstrap is based on the approach that there is the same number of unseen traces as those that are lost after doing re-sampling with replacement [47]. p_i execution frequency of trace i.

$$S_P = S_0 + \sum_{n=1}^{S_0}(1-p_i)^N \tag{4}$$

2) ABUNDANCE-BASED ESTIMATORS

Chao *et al.* proposed an estimator that is based on the abundance of the species [42]. The estimator analyzes the traces that have been executed once (i.e., singletons) or twice (i.e., doubletons). Equation 5 is similar to the bis-corrected Equation 1.

$$S_P = S_0 + \frac{a_1(a_1-1)}{2(a_2+1)} \tag{5}$$

Note that in this case a_1 and a_2 refer to singletons and doubletons:

- a_1 represents the number of traces that are executed only once (i.e., singletons).
- a_2 represents the number of traces that are executed twice (i.e., doubletons).

Abundance-based Coverage Estimator (ACE) analyze species richness considering two groups of traces [47], [52], [56]. On the one hand, S_{abund} represents the number of abundant *unique-traces*. The common approach is to declare abundant individuals with an occurrence higher of ten. S_{rare} represents the number of *unique-traces* that are rarely (e.g., traces executed less than ten times). ACE estimator is described in Equation 6. N_{rare} refers to the total number of executions of rare traces.

$$\begin{aligned} S_P &= S_{abund} + \frac{S_{rare}}{C_ace} + \frac{a_1}{C_{ace}}\gamma^2 \\ &\quad where, \\ C_{ace} &= 1 - \frac{a_1}{N_{rare}} \\ \gamma^2 &= max[\frac{S_{rare}}{C_{ace}} \frac{\sum_{i=1}^{10} i*(i-1)a_i}{N_{rare}(N_{rare}-1)} - 1, 0] \end{aligned} \tag{6}$$

E. ESTIMATION

Using the estimators we have introduced, both abundance and incidence-based, we calculate the number of total traces that are likely to be executed by the safety function. These estimations are performed using the recorded 150 test-campaigns.

Table 2 provides the richness results obtained by each estimator and the percentage of the traces that have not been observed. Results show certain concordance between the majority of the estimators. All estimators except Bootstrap and Jackknife 1 show results above 3000. Note that the data set collects 1996 unique-traces.

TABLE 2. Estimated total number of traces and percentage of unseen traces.

Abundance-based			
Chao2	**Jackknife1**	**Jackknife2**	**Bootstrap**
3384 (41.01%)	2854 (30.06%)	3445 (42.06%)	2357 (15.31%)

Incidence-based	
Chao1	**ACE**
3315 (39.78%)	3366 (40.75%)

The results obtained from the different estimators are depicted in Figure 8. The box-plot shows the results with the 95% CI of each estimator. Furthermore, it allows visualizing the differences or the alignments of the results.

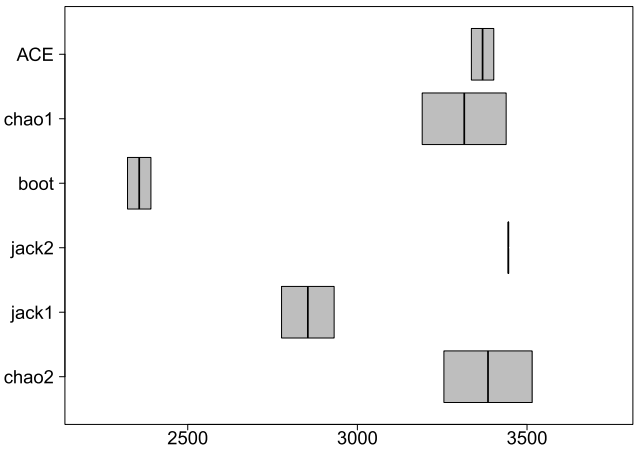

FIGURE 8. Comparison of the obtained results with 95% CI.

If we inspect Table 2 and Figure 8, we see that Chao1 and 2 estimators, Jackknife2 and ACE estimators provide similar results. The certain equivalence of the results obtained with estimators that are even based on different approaches (i.e., abundance and incidence) justifies trusting the results obtained with them. Consequently, we can state that the total number of traces is in the range of 3315 and 3445. As a worst case approach, we could set the total number of traces to 3645, which belongs to the upper CI of Chao2. The results obtained with Bootstrap and Jackknife1 estimator cannot really be judged at this stage of the research.

From a classical safety perspective, the untested paths that we observe here would be translated into a low test-coverage. If we consider 3645 the total number of traces, we get a test

coverage of 54.75%. This value is far from full coverage that has been traditionally achieved (or attempted), where a full coverage was feasible controlling the input parameters of the respective test-function. Moreover, current safety standards do not discern between occurrence probabilities of unobserved (untested) paths as traditional safety operates in a *"software is deterministic"* paradigm. Thus, a notion of path probability would not make much sense. However, given the observed data, it would not be reasonable to claim that in the above example the coverage would be only of 54.75% The unobserved paths have a low probability of occurring as they depend on a random state of the system. Therefore, we believe that there is a need to complement this result with the execution probabilities of the paths and, consequently, know the risk that entails a system with this coverage. This is analyzed and addressed in Section VI.

VI. METHOD 2: EXECUTION PROBABILITY

The study presented in this section aims to estimate the execution probability of the traces that did not appear in the collected set of data in order to calculate the residual risk that they entail. As they are unknown, they must be considered a source of risk due to the fact that they hide untested behavior. Therefore, it is necessary to quantify this residual risk that remains in the system under test. The risk is calculated with the following equation:

$$Risk = Probability\ of\ Occurrence\ x\ Severity\ of\ Consequences \quad (7)$$

As Equation 7 shows, the risk is associated with the probability of occurrence of an event and the severity of the consequences of this event. The study presented in this publication is focused on the probability of occurrence, leaving the estimation of severity as a future task. For a conservative lower bound one can simply set the severity to catastrophic, which is an overly pessimistic approach but a valid baseline.

A. DESCRIPTION OF THE APPROACH

The estimation of the probability of the occurrence of an unknown event is a field that has been examined by different statisticians. I. J. Good proposes a method, known as the Good-Turing frequency estimation method, to estimate the probability of appearing unseen events [57]. Due to the complexity of the method, Gale *et al.* propose a simplified Good-Turing method [58]. Nowadays, this simplified variant technique is used in different fields (e.g., linguistics) where it has found wide acceptance [59].

Simple Good-Turing statistical technique allows estimating the probability of unknown events [58]. Consequently, this technique allows calculating the probability of executing one of the unknown traces identified in the previous method (Section V). The statistical method is based on species abundance distributions. These distributions are based on frequency-frequency data (e.g., number of traces that occurred once, number of traces that occurred twice).

B. MODELING FREQUENCY DATA

Good-Turing statistical technique uses the frequency of frequency data to estimate the probability [58]. This approach, besides, uses the freq-freq data of the execution traces (i.e., the number of traces that have been executed once, the number of traces that have been executed twice, etc.). This data also needs to be modeled to perform the analysis adequately, as the model allows comprehending the mechanism that generates the recorded data. Power-law distributions are commonly used to model frequency distribution where the frequency of an event correlates with the size of the event [60]. As it is mentioned in Section IV-B and shown in Figure 4, the rare-paths are a synthesis of the common trace plus some extra branches at different points of the execution. Figure 4 shows how are constituted the majority of the traces with low-frequency executions. This assumption can be validated by comparing the number of functions executed in commonly executed traces and rarely executed ones. The results obtained from this analysis are collected in Figure 9.

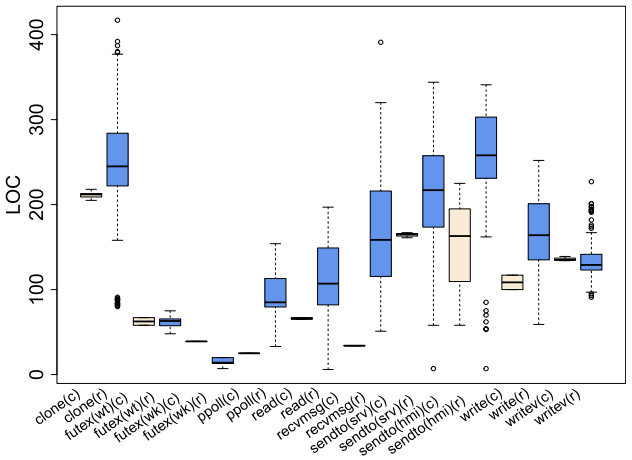

FIGURE 9. Execution trace length depending on common (c) or rare (r) trace.

Figure 9 validates the argument of employing Power-law model in the vast majority of the cases. Thus, as preliminary research, we believe it is valid to use this model. However, it also illustrates a small number of corner cases. For instance, the rare-paths of system-calls *futex(wake)* or *writev* show a slightly lower number than the common ones. Furthermore, there are some outliers too in some system-calls. If we manually check these outliers (e.g., read), we see that these traces belong to failed executions. Thus, we can consider them "valid outliers" as they have occurred, but they do not break with the assumption we have described.

Figure 10 depicts the Cumulative Distribution Function (CDF) of the frequencies-of-frequencies values of the recorded data set and the estimated Power-law distribution. Visual inspection confirms the data fits the model; thus, it seems the appropriate model to continue the study.

Figure 11 illustrates the estimated Power-law model (blue line) with the recorded frequencies of occurrence data.

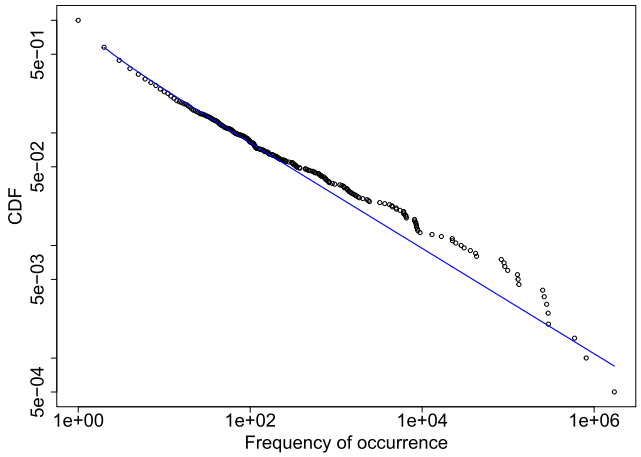

FIGURE 10. Data CDF of the frequency of occurrence data and the fitted power-law (blue line).

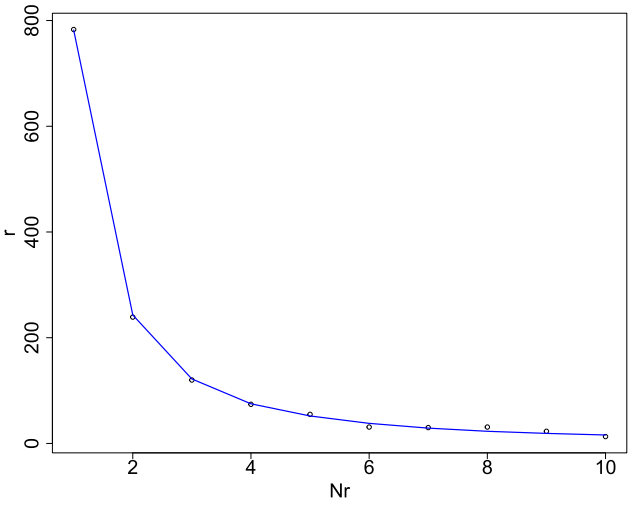

FIGURE 11. Frequency distribution of all recorded system-calls (black points) and the estimated power-law model with function $F(r) = ar^b$ (blue line).

C. DATA SET VALIDATION

The data needs to be cross-validated to follow the statistical analysis. This validation is performed with K-Fold. The validation is similar to the one explained in Section V-C1. However, in this case, we compare the power-law models (originals vs. without k fold). The analysis results in reasonable MSE values: mean equals 263, minimum values in 190 and maximum 357. Therefore, we can state that the results are adequate.

D. EXECUTION PROBABILITY ESTIMATION

Simple Good-Turing is employed with the aim of estimating the execution probability of the set of traces that we have identified as untested [59]. This statistical technique is based on Equation 8, where $r = 0$ represents the unseen traces:

$$P_r = \begin{cases} \dfrac{N_1}{N} & \text{for } r = 0 \\ (r+1) \cdot \dfrac{N_{r+1}}{N \cdot N_r} & \text{for } r \geq 1 \end{cases} \tag{8}$$

- r: defines the rate occurrence of the traces.
- N_r: defines the number of traces that have been executed with rate r.
- N_{r+1}: represents the count of rate $r+1$.
- N: total number of recorded traces.
- P_r: probability of traces with rate r.

As pointed by the literature, one of the main issues is the unreliability of Simple Good-Turing for high rates [61]. As we pointed out in Section IV-B, there are a small number of traces that are executed most of the time on Linux-based systems. Therefore, we obtain significantly high rates. However, the most common approach to deal with this unreliability is to divide the analysis into two analyses. On the one hand, Simple Good-Turing is used with the smaller rates and, on the other hand, Maximum Likelihood Estimation (MLE) for the higher rates. The line that determines what is considered a high or a low rate is estimated employing Shannon Entropy. Shannon Entropy quantifies the amount of information of a data set, thus, we select a dividing rate where the two classifications provide the most equivalent amount of information. After that, the probabilities are normalized to achieve a sum of one.

Table 3 provides the estimated probabilities for each rate. The probability of untested kernel traces (P_{zero}) is defined as the probability of rate 0.

TABLE 3. Probability results for each rate.

rate	probability	rate	probability
0	$1.426390 \cdot 10^{-4}$	8	$1.356093 \cdot 10^{-6}$
1	$1.133577 \cdot 10^{-7}$	9	$1.247990 \cdot 10^{-6}$
2	$2.747237 \cdot 10^{-7}$	10	$1.823985 \cdot 10^{-6}$
3	$4.485210 \cdot 10^{-7}$	...	...
4	$6.323149 \cdot 10^{-7}$	...	...
5	$7.997474 \cdot 10^{-7}$	592661	$1.081005 \cdot 10^{-1}$
6	$9.743922 \cdot 10^{-7}$	809052	$1.475699 \cdot 10^{-1}$
7	$1.157287 \cdot 10^{-6}$	1717572	$3.132826 \cdot 10^{-1}$

1) CONFIDENCE INTERVAL (CI)

Random sampling with replacement (bootstrapping) is commonly used in statistics to examine the variation of estimates and estimate the CIs [62]. However, when working with frequency-based distributions, re-sampling is often problematic. While most common events (i.e. those with higher frequency) are not affected by the re-sampling, events that rarely occur (i.e., those of lower frequency) get re-sampled, and hence, dramatically distorted. In other words, bootstrap underestimated the traces with a low execution frequency as it generates data sets with a significantly lower number of traces with low frequencies, and therefore, underestimation of unseen paths. To avoid this problem, an alternative approach is to sample (without replacement) a subset of the whole data set, to achieve a smaller data set but that is reliable and non-distorted.

TABLE 4. Confidence interval for untested execution probability.

Confidence Interval	2.5%	Estimated	97.5%
Estimated Probability	$1.620313 \cdot 10^{-4}$	$1.426390 \cdot 10^{-4}$	$1.724785 \cdot 10^{-4}$

For the purpose of calculating the CIs, we randomly sample 90% of the data set a thousand times. The results obtained are shown in Table 4. This means that if we were to collect the data again and redo the analysis, the results would most likely be below the upper CI (97.5%). Thus, we can consider upper CI as the WCS. Moreover, upper and lower CIs show even higher results than the estimated value. Although this may seem odd at first glance, the result makes sense and is beneficial for our study. As CIs are estimated with smaller data sets, the obtained probabilities are higher. In other words, a larger data set reduces the probability of observing a trace that has not previously been observed. This is also shown in following Section VI-D2.

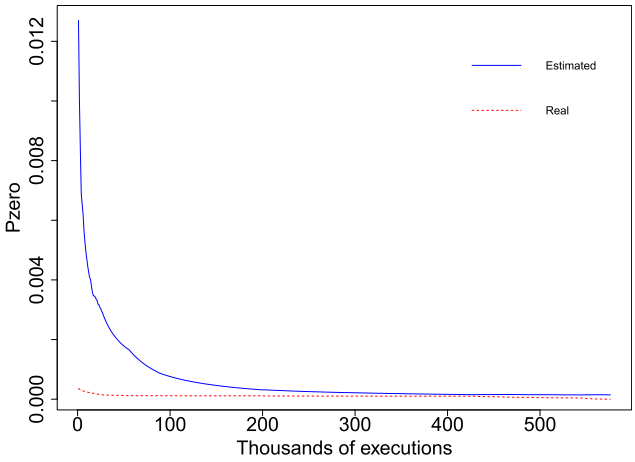

FIGURE 12. P_{zero} values along with execution traces increase (blue points). P_{zero} real value estimated with the remaining data set.

2) PROBABILITY DECREASE WITH TESTING EFFORT

The adequacy of the Good-Turing approach can be confirmed by comparing the estimated results (Simple Good-Turing) with the probability of the new traces that exist in the remaining of the data set (MLE). For instance, in the 100 test-campaign, we calculate the probability of unseen traces with Simple Good-Turing and the MLE probability of the traces that appear between the test-campaign 100 and 150. As we know the traces that appear in the remaining data set and the total number of traces, we calculate the probability using MLE. The results are shown in Figure 12.

It can be seen how the estimated probability decreases as the number of executions increases in Figure 12 (blue line). Therefore, the probability of executing non-tested traces decreases by gaining knowledge of the system. In the event that the probability is too high for a certain system, this probability can be decreased by continuing the analysis. Besides, we can confirm that the Good-Turing approach provides stable results after 100-200k executions. The graph also shows the real probability, that is, the one we calculate knowing the remaining data set. Note this probability is a rough estimation, as it does not take into account unknown traces, only the traces that appeared in the following test-campaigns.

E. DISCUSSION OF THE RESULTS

Once we have measured the probability of $1.42e^{-4}$, a reasonable question is: is it sufficiently low for a SIL2 safety-related system? Current safety standards do not provide any reference value to determine whether this probability result is within a tolerable risk. However, the standards do provide reference ranges for dangerous hardware failures. For instance, IEC 61508 safety standard defines Probability of Dangerous Failure per Hour (PFH). In contrast to software, the standard considers the existence of random failures in hardware and, hence, requires compliance with the PFH ranges. But as observed in these research activities, the stochastic nature of the untested path is not in its code-sequence but in the occurrence of its activation. It is not possible to deterministically create a system state that would deterministically enter one particular path. Consequently, it seems legitimate to work towards obtaining ranges for the software execution on a given hardware, such as those that exist for hardware.

As there is currently no suitable reference for this software execution on a given hardware, in this manuscript we take as a reference the values established by PFH. In this manner, we can obtain an approximate idea if the described method (following the As Low As Reasonably Practicable (ALARP) principle) may fit the reference ranges. For a Safety Integrity Level (SIL) 2 the standard determines the PFH range to $10^{-6} < PFH \leq 10^{-7}$. If we follow the most conservative approach, where the execution of any untested trace leads to catastrophic consequences, we can estimate the probability of a dangerous failure per hour. In the case of a redundant architecture such as the SIL2LinuxMP's two-out-of-two (2oo2) architecture, the probability of executing an untested path in both cores at the same time is ${P_{zero}}^2$. It is essential to take into account that safety is a system property and, hence, the values set by the standard are more attainable with this type of architecture. Consequently, we would only need to estimate the average execution frequency of system-calls to be able to translate it to a PFH comparable value. SIL2 task is executed with a period of 50 milliseconds and exercises an average of 25 system-calls per period. As a result, the system exercises $18 \cdot 10^5$ system-calls per hour.

$$Freq = 3.6 \cdot 10^6 \cdot \frac{25}{50} = 18 \cdot 10^5 \tag{9}$$

Bearing in mind that we are following the most pessimistic approach, where the execution of any unknown trace is considered dangerous, we can estimate the probability of dangerous failure per hour. For this purpose, we use the Binomial probability mass function.

$$Probability = 1 - \binom{Freq}{0} \cdot P_{zero}(1 - P_{zero})^{Freq} \tag{10}$$

We rest to 1 the probability of not executing any untested path in *Freq* executions. In other words, the cumulative probability of at least one failure. If we have a redundant architecture, such as a two-out-of-two (2oo2), the probability of executing an untested trace simultaneously (in the same execution cycle) on both cores would be ${P_{zero}}^2$.

$$Probability = 1 - \binom{Freq}{0} \cdot {P_{zero}}^2 (1 - {P_{zero}}^2)^{Freq} \quad (11)$$

$$Probability \simeq 35 \cdot 10^{-3} \quad (12)$$

As in can be observed the obtained value is significantly larger than the PFH range determined in the safety standard.

$$Probability \gg PFH_{sil2} \quad (13)$$

It is necessary to take into account that this calculation has been performed following the most conservative and, hence, pessimistic approach. Emphasize that this work intends is not to declare that Linux is safe, but to contribute with methods that can make that evaluation. Indeed, it is important to note that these results are achieved with around six thousand hours of execution data and that the values can be reduced with further exercise. For instance, if we perform the risk calculation with a reduced data set (equivalent of four thousand hours exercising effort), the obtained value is $15 \cdot 10^{-2}$. Therefore, the data set size or the testing effort implies a considerable difference in the risk reduction.

VII. CONCLUSION & FUTURE WORK

This publication introduces two statistical methods with the aim of progressing in the field of test coverage of next-generation safety-related autonomous systems, focusing in this case specifically on the Linux kernel execution path test coverage. The methods presented in this manuscript are not dependent on each other. Moreover, they can be considered complementary as the results provided by them add extra information and knowledge to the other. It is important to mention that the research objective is not to identify the specific statistical methods but rather to examine the viability and benefits of statistical-based alternatives to cope with these systems' complexity. On the one hand, the publication studies different non-parametric estimators to quantify the number of traces that are not covered during the verification phase. On the other hand, we examine a technique to be able to estimate the execution probability of those untested traces. Both alternatives show positive results in order to pave the way towards the safety assurance of next-generation safety-related systems.

Test coverage estimation by non-parametric analysis seems an interesting alternative to traditional techniques. Although the accumulation curve shows the large number of different paths that an application such as AEB can exercise, it also exhibits a trend to achieve the asymptote. Consequently, we are able to estimate the total number of paths that have a relevant probability of occurrence and the number of paths that have not been covered. Although the estimators do not give exactly the same results, most of them give significantly similar ranges. Therefore, it can be argued that the total value of traces that can be run is in the range where most estimators coincide. Moreover, this argument is strengthened by the fact that some of these estimators are based on different data analyses (i.e., incidence vs. abundance). However, there are some estimators that show a significant variability between their results, especially bootstrap estimators. J. Béguinot state that Jackknife 2 is usually the most suitable non-parametric estimator for species richness calculation [63]. In addition, the author argues that Chao method is also appropriate when the data set is close to the sampling completeness. Thus, it is necessary to continue investigating which estimator fits more appropriately to this type of system. For this purpose, we should examine the estimators with other use-cases and with other system-context. For instance, we believe it would be interesting to investigate the estimators' variability with the same application but different system-context (e.g., different CPU loads).

We also consider the execution probability estimation of untested paths as a step forwards in the field of test coverage of complex safety-related systems. Contrary to the techniques that have been employed traditionally, we take into account the uncertainty that these systems possess. In fact, the non-parametric estimators show a test coverage percentage significantly below 100%. Consequently, this analysis complements the previous one and allows us to determine the risk that entails the non-covered paths. This method also contributes in providing adequate explanation when full coverage is not achievable, as stated by the IEC 61508 standard (IEC 61508-3 Ed2 Table B.2). Nonetheless, there is a need for a reference value, equivalent to PFH ranges for hardware failures, to comprehend the risk associated with untested paths. Consequently, we believe this needs to be discussed with Certification Authorities (CAs).

As aforementioned, in this preliminary assessment, we have entirely focused on the system-call exercised by the SIL2 task, leaving aside small pieces of inter-calls. Since the obtained technical results seem promising, further analysis needs to be carried out to include the kernel function-calls that occur between system-calls. We believe that this can be achieved by sampling on internal kernel functionality in much the same way as we have sampled on the kernel's system-call interface.

It is essential to note that these methods are only justifiable performing continuous monitoring of the process. Continuous monitoring allows updating the results constantly and, hence, it strengths the results while the data set increases. Contrary to biology, where obtaining new samples is generally not that easy, in the software case, it is potentially easier. Consequently, a significant higher assurance should be doable by exercising the system continuously and using this approach during the intensive test campaigns used to validate safety-related systems.

Finally, we believe it is interesting and necessary that the described methods undertake a thorough review by

additional safety experts and a CA. As mentioned above, this study intends to gain insight into the benefits of statistical alternatives concerning traditional techniques. Nevertheless, it is also essential to identify the limits and drawbacks that these state-of-the-art techniques may have. Furthermore, we believe that these methods may be potentially valid for other OSs or software layers, although we cannot confirm it as the analysis has not been conducted.

REFERENCES

[1] M. Fisher, L. A. Dennis, and M. Webster, "Verifying autonomous systems," *Commun. ACM*, vol. 56, no. 9, pp. 84–93, 2013.

[2] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, MA, USA: MIT Press, 2016.

[3] *Road Vehicles—Safety of the Intended Functionality*, document 21448:2019 ISO/PAS, 2019.

[4] N. Mc Guire and I. Allende, "Approaching certification of complex systems," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2020, pp. 70–71.

[5] Corbet and G. Kroah-Hartman, "Linux kernel development report," Linux Found., San Francisco, CA, USA, Tech. Rep., 2017.

[6] L. Bulwahn, T. Ochs, and D. Wagner, *Research on an Open-Source Software Platform for Autonomous Driving Systems*. Munich, Germany: BMW Car IT GmbH, 2013.

[7] OSADL. *SIL2LinuxMP—The Safety Project for Linux*. Accessed: Apr. 2021. [Online]. Available: https://sil2.osadl.org/

[8] *Enabling Linux in Safety Applications (ELISA)*. Accessed: Apr. 2021. [Online]. Available: https://elisa.tech/

[9] C. Hernandez, J. Flieh, R. Paredes, C.-A. Lefebvre, I. Allende, J. Abella, D. Trillin, M. Matschnig, B. Fischer, K. Schwarz, J. Kiszka, M. Ronnback, J. Klockars, N. McGuire, F. Rammerstorfer, C. Schwarzl, F. Wartet, D. Ludemann, and M. Labayen, "SELENE: Self-monitored dependable platform for high-performance safety-critical systems," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 370–377.

[10] A. Platschek, N. Guire, and L. Bulwahn, "Certifying Linux: Lessons learned in three years of SIL2linuxMP," in *Proc. Embedded World*, Nuremberg, Germany, Feb. 2018.

[11] *SIL2 61508 Ed 2 Compliance Route. SIL2LinuxMP—The Safety Project for Linux*, Nicholas Mc Guire, Lanzhou, China, 2019.

[12] *Linux Grows on Government Systems*. Accessed: Feb. 2021. [Online]. Available: https://www.govtech.com/security/Linux-Grows-on-Government-Systems.html%

[13] *Trust SUSE US Federal Government Solutions*. Accessed: Feb. 2021. [Online]. Available: https://www.suse.com/sector/federal/

[14] *DISA Has Released the Red Hat Enterprise Linux 8 STIG*. Accessed: Feb. 2021. [Online]. Available: https://www.redhat.com/es/blog/disa-has-released-red-hat-enterprise-lin%ux-8-stig

[15] H. Leppinen, "Current use of Linux in spacecraft flight software," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 32, no. 10, pp. 4–13, Oct. 2017.

[16] *ELC: SpaceX Lessons Learned*. Accessed: Feb. 2021. [Online]. Available: https://lwn.net/Articles/540368/

[17] J. Gruen, *Linux in Space*. San Francisco, CA, USA: The Linux Foundation, 2012.

[18] H. F. Grip, J. Lam, D. S. Bayard, D. T. Conway, G. Singh, R. Brockers, J. H. Delaune, L. H. Matthies, C. Malpica, T. L. Brown, and A. Jain, "Flight control system for NASA's Mars helicopter," in *Proc. AIAA Scitech Forum*, 2019, p. 1289.

[19] I. Allende, N. Mc Guire, J. Perez, L. G. Monsalve, J. Fernandez, and R. Obermaisser, "Estimation of Linux kernel execution path uncertainty for safety software test coverage," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1446–1451.

[20] I. Allende, N. Mc Guire, J. Perez, L. G. Monsalve, and R. Obermaisser, "Towards Linux based safety systems—A statistical approach for software execution path coverage," *J. Syst. Archit.*, vol. 116, Jun. 2021, Art. no. 102047.

[21] STC: Statistical Test Coverage. *Repository. Available: GIT Clone*. Accessed: May 2021. [Online]. Available: https://sil2.osadl.org/git/STC.git

[22] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, document IEC 61508 (1-7), Int. Electrotechnical Commission, 2nd edition, 2010.

[23] *Road Vehicles-Functional Safety*, Standard ISO26262 ISO. 26262, International Standard ISO/FDIS, 2011.

[24] *Linux in 2020: 27.8 Million Lines of code in the Kernel, 1.3 Million in Systemd*. Accessed: May 2021. [Online]. Available: https://www.linux.com/news/linux-in-2020-27-8-million-lines-of-code-in-%the-kernel-1-3-million-in-systemd/

[25] N. M. Guire, P. Okech, and G. Schiesser, "Analysis of inherent randomness of the Linux kernel," in *Proc. 11th Real-Time Linux Workshop*, 2009, pp. 1–35.

[26] P. Okech and N. M. Guire, "Analysis of statistical properties of inherent randomness," in *Proc. 12th Real-Time Linux Workshop*, 2010, pp. 1–8.

[27] P. Okech, N. M. Guire, C. Fetzer, and W. Okelo-Odongo, "Investigating execution path non-determinism in the Linux kernel," in *Proc. 14th Real-Time Linux Workshop, Lugano. (OSADL)*, 2013, pp. 1–8.

[28] P. Okech, N. M. Guire, and C. Fetzer, "Utilizing inherent diversity in complex software systems," in *Proc. Austral. Syst. Saf. Conf. (ASSC). Austral. Comput. Soc.*, 2014, pp. 1–8.

[29] P. Okech, N. Mc Guire, and W. Okelo-Odongo, "Inherent diversity in replicated architectures," 2015, *arXiv:1510.02086*. [Online]. Available: http://arxiv.org/abs/1510.02086

[30] Z. Gao, Y. Liang, M. B. Cohen, A. M. Memon, and Z. Wang, "Making system user interactive tests repeatable: When and what should we control?" in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2016, pp. 55–65.

[31] K. Lu and H. Hu, "Where does it go?: Refining indirect-call targets with multi-layer type analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1867–1881.

[32] M. Simunovic, "CallGraph tool—How (not) to develop a call graph detection tool," in *Proc. ELISA Workshop*, May 2020, pp. 1–33.

[33] J. P. Cerrolaza, R. Obermaisser, J. Abella, F. J. Cazorla, K. Grüttner, I. Agirre, H. Ahmadian, and I. Allende, "Multi-core devices for safety-critical systems: A survey," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–38, Sep. 2020.

[34] S. Burton, L. Gauerhof, and C. Heinzemann, "Making the case for safety of machine learning in highly automated driving," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Cham, Switzerland: Springer, 2017, pp. 5–16.

[35] I. Agirre, P. Onaindia, T. Poggi, I. Yarza, F. J. Cazorla, L. Kosmidis, K. Gruttner, M. Abuteir, J. Loewe, J. M. Orbegozo, and S. Botta, "UP2DATE: Safe and secure over-the-air software updates on high-performance mixed-criticality systems," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 344–351.

[36] I. Allende, N. Mc Guire, J. Perez, L. G. Monsalve, N. Uriarte, and R. Obermaisser, "Towards Linux for the development of mixed-criticality embedded systems based on multi-core devices," in *Proc. 15th Eur. Dependable Comput. Conf. (EDCC)*, Sep. 2019, pp. 47–54.

[37] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 91–101.

[38] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., 2016.

[39] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," 2015, *arXiv:1511.02680*. [Online]. Available: http://arxiv.org/abs/1511.02680

[40] A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 4762–4769.

[41] S. Draskovic, R. Ahmed, P. Huang, and L. Thiele, "Schedulability of probabilistic mixed-criticality systems," *Real-Time Syst.*, pp. 1–46, Feb. 2021. [Online]. Available: https://link.springer.com/article/10.1007/s11241-021-09365-4#article-info

[42] A. Chao, "Non-parametric estimation of the number of classes in a population," *Scandin. J. Statist.*, vol. 11, pp. 265–270, Jan. 1984.

[43] E. P. Smith and G. V. Belle, "Nonparametric estimation of species richness," *Biometrics*, vol. 40, pp. 119–129, Mar. 1984.

[44] A. E. Magurran, "Species abundance distributions over time," *Ecol. Lett.*, vol. 10, no. 5, pp. 347–354, May 2007.

[45] R. K. Colwell, C. X. Mao, and J. Chang, "Interpolating, extrapolating, and comparing incidence-based species accumulation curves," *Ecology*, vol. 85, no. 10, pp. 2717–2727, Oct. 2004.

[46] J. Oksanen, R. Kindt, P. Legendre, B. O'Hara, M. H. H. Stevens, M. J. Oksanen, and M. Suggests, "The Vegan Package," *Community Ecol. Package*, vol. 10, nos. 631–637, p. 719, 2007.

[47] J. Oksanen, "Vegan: Ecological diversity," *R Project*, vol. 368, pp. 1–11, Dec. 2013.

[48] A. Chao and C.-H. Chiu, "Species richness: Estimation and comparison," in *Wiley StatsRef, Statistics Reference Online*, vol. 1. Apr. 2014, pp. 1–26.

[49] R. Kindt, P. Van Damme, and A. J. Simons, "Patterns of species richness at varying scales in western kenya: Planning for agroecosystem diversification," *Biodiversity Conservation*, vol. 15, no. 10, pp. 3235–3249, Sep. 2006.
[50] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112. New York, NY, USA: Springer, 2013.
[51] N. J. Gotelli and R. K. Colwell, "Quantifying biodiversity: Procedures and pitfalls in the measurement and comparison of species richness," *Ecol. Lett.*, vol. 4, no. 4, pp. 379–391, Jul. 2001.
[52] N. Gotelli and R. Colwell," *Estimating Species Richness*, vol. 12, pp. 39–54, Jan. 2011.
[53] C. V. Basualdo, "Choosing the best non-parametric richness estimator for benthic macroinvertebrates databases," *Revista de la Sociedad Entomológica Argentina*, vol. 70, nos. 1–2, pp. 1–3, 2017.
[54] A. Chao, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, vol. 43, pp. 783–791, Dec. 1987.
[55] K. P. Burnham and W. S. Overton, "Estimation of the size of a closed population when capture probabilities vary among animals," *Biometrika*, vol. 65, no. 3, pp. 625–633, 1978.
[56] R. B. O'Hara, "Species richness estimators: How many species can dance on the head of a pin?" *J. Animal Ecol.*, vol. 74, no. 2, pp. 375–386, Mar. 2005.
[57] I. J. Good, "The population frequencies of species and the estimation of population parameters," *Biometrika*, vol. 40, nos. 3–4, pp. 237–264, 1953.
[58] W. A. Gale and G. Sampson, "Good-turing frequency estimation without tears," *J. Quant. Linguistics*, vol. 2, no. 3, pp. 217–237, 1995.
[59] G. Sampson, *Empirical Linguistics*. London, U.K.: A&C Black, 2002.
[60] E. White, B. Enquist, and J. Green, "On estimating the exponent of power-law frequency distributions," *Ecology*, vol. 89, pp. 905–912, May 2008.
[61] *Lecture Notes of: Artificial Intelligence II*, Prof. Olga Veksler, London, U.K., 2012.
[62] B. Efron, "Better bootstrap confidence intervals," *J. Amer. Statist. Assoc.*, vol. 82, no. 397, pp. 171–185, 1987.
[63] J. Béguinot, "Basic theoretical arguments advocating Jackknife-2 as usually being the most appropriate nonparametric estimator of total species richness," *Annu. Res. Rev. Biol.*, vol. 10, no. 1, pp. 1–12, Jan. 2016.

IMANOL ALLENDE received the master's degree in embedded systems from Mondragon University. He is currently pursuing the Ph.D. degree in computer science, with Prof. Roman Obermaisser, with Siegen University. Since 2014, he works as a Researcher at Ikerlan Research Centre, where he has participated in different research projects involving safety-related systems-based open-source software. His research interest includes autonomous safety-related systems, with a focus on Linux-based systems.

NICHOLAS MC GUIRE started working on magnetic bearing control systems at the Technical University of Vienna. In 1995, he moved towards the other end of the spectrum towards clusters at the Institute for Computational Material Science, University of Vienna. With the focus shifting to real-time and distributed embedded systems, initially maintaining RTLinux/GPL, from 2001 to 2005, safety related systems were an almost natural next step, in 2003. His main topic is system safety since he founded OpenTech, with a special focus on the utilization of FLOSS/COTS components in safety-related systems and thus integration of high-complexity pre-existing (SOUP) components while retaining system level safety properties.

JON PEREZ-CERROLAZA (Senior Member, IEEE) received the Ph.D. degree in computer science from TU Wien, in 2011. He is a Principal Researcher at Ikerlan in the field of dependable autonomous systems. He has worked for more than fifteen years in the development and certification of industrial safety-critical systems, such as on-board SIL4 railway signaling systems (ERTMS/ETCS). His research interests include dependability, machine learning, and cybersecurity technologies.

LISANDRO G. MONSALVE received the master's degree in embedded systems engineering from the University of the Basque Country, in 2008. He joined IKERLAN's Electronics Department, in 2010, and moved to the Hardware Systems Team, in 2016. During this time, he has taken part in the development of real-time embedded systems for industrial applications as well as for research projects. He is currently coordinating Hardware Systems Department research activities related to lower layers' software in topics, such as the use of Linux in embedded systems with real time and safety related requirements and the design and customization of frameworks for the development of portable software for microcontrollers. In the industrial field, he has worked on applied research projects developing control platforms for different companies in the renewable energy, transport, and health sectors.

JENS PETERSOHN received the master's degree in physics from The University of Toledo, OH, USA. He has been active in a number of different industries in the last 25 years, including fluid dynamics research at NASA, large scale super computer system development at Cray/SGI and for the past 15 in the automotive industry focusing on software architectures and novel new solutions to the challenging problems facing the industry in the next years.

ROMAN OBERMAISSER received the master's degree in computer sciences and the Ph.D. degree in computer science, with Prof. Hermann Kopetz, as a research advisor, from Vienna University of Technology, in 2001 and 2004, respectively, and the habilitation (venia docendi) certificate for technical computer science, in 2009. He is currently a Full Professor at the Division for Embedded Systems, University of Siegen. His research interest includes system architectures for distributed embedded real-time systems. He wrote a book on an integrated time-triggered architecture published by Springer-Verlag, USA. He is the author of several journal papers and conference publications. He has also participated in numerous EU research projects (e.g., SAFEPOWER, universAAL, DECOS, and NextTTA) and was a Coordinator of the European research projects DREAMS, GENESYS, and ACROSS.