SiPTA: Signal Processing for Trace-based Anomaly Detection

Mohammad Mehdi Zeinali Zadeh, Mahmoud Salem, Neeraj Kumar,
Greta Cutulenco and Sebastian Fischmeister
University of Waterloo, ON, Canada
{mmzeinal, m4salem, n26kumar, gcutulen, sfischme}@uwaterloo.ca

Abstract

Given a set of historic good traces, trace-based anomaly detection deals with the problem of determining whether or not a specific trace represents a normal execution scenario. Most current approaches mainly focus on application areas outside of the embedded systems domain and thus do not take advantage of the intrinsic properties of this domain.

This work introduces SiPTA, a novel technique for offline trace-based anomaly detection that utilizes the intrinsic feature of periodicity found in embedded systems. SiPTA uses signal processing as the underlying processing algorithm. The paper describes a generic framework for mapping execution traces to channels and signals for further processing. The classification stage of SiPTA uses a comprehensive set of metrics adapted from standard signal processing. The system is particularly useful for embedded systems, and the paper demonstrates this by comparing SiPTA with state-of-the-art approaches based on Markov Model and Neural Networks. The paper shows the technical feasibility and viability of SiPTA through multiple case studies using traces from a field-tested hexacopter, a mobile phone platform, and a car infotainment unit. In the experiments, our approach outperformed every other tested method.

1. Introduction

Even for well-tested deployed systems, undetected errors may lead to catastrophic failures similar to the failures of Therac-25 device [18] and Ariane 5 flight [19]. Therefore, safety-critical embedded systems typically used in medical, nuclear, and automotive domains require compliance with standards that recommend the use of software monitors to detect anomalies in the development and production phases (e.g., ISO-26262 [7] for automotive functional safety and DO-178C [8] for airborne systems).

In such systems, trace-based anomaly detection can act as a monitoring mechanism and invoke modules responsible for prevention and recovery from failures. In essence, trace-based anomaly detection aims at detecting execution patterns that do not conform to the normal functioning of the system. Anomalous patterns can indicate software bugs or malfunctions. This is also convenient, as such an analysis treats the system under scrutiny as a black box and does not require knowledge of its internals. The approach just leverages the computational power of computer processors to distinguish anomalous traces from normal traces, and therefore can help in reducing the likelihood of catastrophic failures.

However, the main challenge of trace-based anomaly detection is how to identify an incorrect behavior without raising too many false alarms. Related work in a comprehensive survey [10] measures the effectiveness of detection mechanisms through *false positives*, where the detector incorrectly raises an alarm for a normal execution behavior, and *false negatives*, where the detector overlooks an anomaly. High false alarm rates diminish the value of the mechanism, because the users stop trusting it.

Detection of anomalies can be done both, online (during run time) or offline (once the program has finished execution). The offline approach considers an entire trace of a system execution scenario for analysis, while the online approach can only work on streams of execution events collected during program runs to detect anomalies on-the-fly. Online anomaly detection techniques can therefore be used for monitoring program behavior and taking corrective actions during run time. They normally do so by incrementally adapting the threshold of the anomaly scores for the captured streams. Offline detection techniques will therefore need some adaptation to be used at run time, mainly in the amount of data required to detect anomalies. However, as pointed out in [11], some offline techniques cannot be used for online anomaly detection as they need to process the entire trace before deriving a conclusion.

This paper introduces SiPTA, which realizes a novel technique for offline program trace-based anomaly detection utilizing the intrinsic feature of periodicity found in embedded systems. As it will be explained later, SiPTA uses signal processing algorithms to identify periodic features in an embedded system. The contributions of this paper are outlined as follows:

- Introducing the concept of using intrinsic system periodicity for trace analysis.
- Demonstrating the feasibility and viability of using signal processing algorithms for trace analysis.
- Formalizing a generic framework for modeling traces and mapping them to signals and channels.
- Specifying a comprehensive set of metrics based on frequency spectra useful for the classifying traces.

In our study, we restrict our attention to traces obtained from embedded devices. We evaluate SiPTA through the analysis of sets of application-specific traces generated from QNX RTOS [2] running on deployed commercial platforms (i.e., a hexacopter platform, a car infotainment system, a phone OS, and an embedded development test kit) covering a wide range of execution scenarios. We chose to create our own dataset, since the established datasets [1, 3] are unsuited for embedded systems and are under criticism [13, 22].

ESWEEK'14 October 12 - 17 2014, New Delhi, India
Copyright © 2014 ACM 978-1-4503-3052-7/14/10...$15.00
http://doi.acm.org/10.1145/2656045.2656071

The remainder of the paper is organized as follows: Section 2 presents an overview of the proposed methodology for detecting anomalies; Section 3 explains SiPTA in detail; Section 4 outlines the experimental setup and briefly discusses the alternative approaches to SiPTA; Section 5 compares the experimental results for SiPTA and other alternative approaches; Section 6 discusses the results and threats to validity, followed by conclusion in Section 7.

2. Overview

Our work applies to detecting anomalies by analyzing traces from systems with recurring periodic processes, as often found in the embedded systems domain [24].

Figure 1 shows a generic work-flow for offline detection of anomalies in a system. The preprocessor extracts relevant information from the input traces to be analyzed by the underlying anomaly detection technique. This information can contain, for instance, event names, time-stamps, and process names. The detection engine outputs a comprehensive score for each of the input traces. For the training traces, this score represents the normal behavior of the system and can be called *normal behavior score*. For the testing phase, the same procedure results into an *anomaly score* for the input trace. Finally, a binary classifier decides whether this test trace is normal or anomalous by comparing this *anomaly score* to the *normal behavior score*.

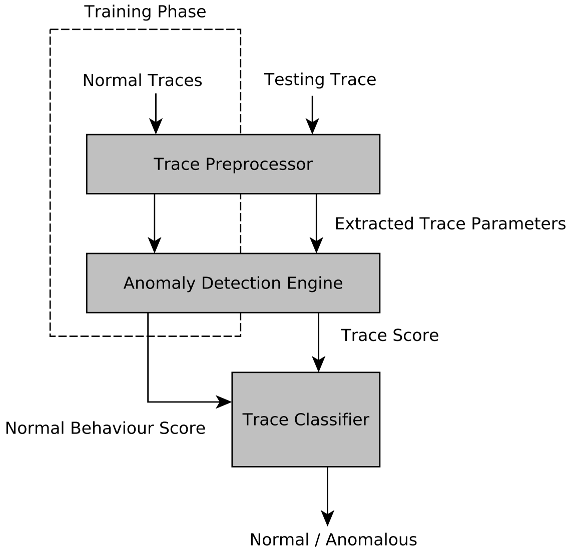

Figure 1: Work-flow Overview

SiPTA is the anomaly detection engine that we propose in this paper. It takes advantage of the periodic behavior exhibited in the traces [20], transforms the data and uses signal processing to identify anomalous traces. More specifically, SiPTA performs the following key steps:

1. **Modeling the trace to signals.** First, it extracts signals and channels from the input trace. These signals and channels are then valuated by assigning a time series to them.
2. **Extracting periodic features of the trace.** Next, SiPTA extracts periodic features from the time series obtained in the previous step. For this purpose, it uses discrete Fourier transform (DFT) to calculate the spectra of channels.
3. **Classifying using periodic features.** Finally, it applies metrics to estimate the expected periodic features for any normal trace. If a new trace does not meet these expectations, it will be considered as an anomalous trace.

In the context of detection of anomalies from sequential traces, most of research has been focused on the anomaly detection of operating system events. A recent survey paper [11] summarized the progress in that research area. They discussed two major approaches. The first approach uses Markovian modeling [27] to study the probabilistic characteristics of event transitions. This extends from first-order to higher-order Markov Models and some equivalent methods as probabilistic suffix trees (PST), and sparse Markov trees (SMT). The second approach models the event transition states through Finite State Automata (FSA) and Hidden Markov Models (HMM) methods. Besides these, there are other approaches to anomaly detection that do not fall into either of the two categories. These approaches to anomaly detection exploit mathematical concepts that are widely used in signal processing. For instance, wavelet transform was used in [12, 15, 21, 25, 26] and Fourier transform was used in [30]. Their work focused on detecting anomalies in network-based systems. In contrast, our approach introduces the idea of using frequency characteristics of event transitions.

3. SiPTA — Signal Processing for Trace Analysis

The following subsections, will formulate each of the parts of the proposed anomaly detection engine and trace classifier described in Figure 1. These parts are trace-to-signal modeling, signal processing algorithm, metrics, and classification. We first show how a trace is modeled as signals. Next, we show how the signal processing algorithm processes these signals to extract their periodic features. We then discuss the metrics which SiPTA uses to detect the difference between the results for training and testing traces. Finally, we discuss the classification method that SiPTA uses to classify the testing traces.

3.1 Trace-to-signal modeling

The first part of SiPTA is modeling traces to signals and channels. This allows the signal processing algorithm to process traces. This section provides the necessary terms and definitions for the trace model, time series, and the trace to signal association. To define trace-to-signal modeling, we should first formally model traces, signals, and then also model the relation between them.

3.1.1 Trace Model

A trace consists of entries. Each entry contains multiple parameters as data points. The following model formally specifies these parameters.

The parameter ρ specifies a permitted value in a trace element (e.g., a concrete process identifier value). A parameter set $\mathbb{P}$ specifies a set of parameters seen at a particular trace location (e.g., all process identifiers listed in the whole trace, or the concatenation of two or more columns, e.g., `PID` and `Syscall`). With these two definitions in place, we can now formalize a trace.

Definition 1 (Trace Entry). *A trace entry $E := \langle idx, t, P\rangle$ is a tuple consisting of a row index idx, a real-time value t, and a trace parameter sequence P.*

The parameter sequence P in a trace entry has a finite number of trace parameters. Let $M := |P|$. Then the i^{th} member of P is a member of $\mathbb{P}_i$. In other words we say that a parameter sequence $P = \langle p_1, \ldots, p_M\rangle$, (and consequently the trace entry $E = \langle idx, t, P\rangle$) is defined over $\mathcal{P} = \langle \mathbb{P}_1, \ldots, \mathbb{P}_M\rangle$ if and only if $\forall 1 \leq i \leq M : p_i \in \mathbb{P}_i$.

DEFINITION 2 (Trace). *A trace T is a temporally ordered sequence of trace entries E_i over $\mathcal{P}$. The order is based on $E_i.t$, which is the real time value of the entry.*

$$T = \langle E_1, ..., E_n \rangle$$

Also, the idx should be equal to the index of the entry, namely:

$$\forall 1 \leq i \leq n : E_i.idx = i$$

Naturally, a trace contains entries in temporal order. Thus the following property should hold for a trace with n entries.

$$\forall E_i, E_j \text{ with } i > j : E_i.t \leq E_j.t \qquad (1)$$

DEFINITION 3 (Filtered trace). *Let $\beta \in \mathbb{P}_j$, filtered trace $T(j, \beta)$ reduces a trace to entries that match a particular trace parameter β. The filtered trace is ordered based on the time values of its entries.*

$$T(j, \beta) := \langle T.E \mid T.E.P.p_j = \beta \rangle$$

EXAMPLE 1 (Trace). *Consider a trace that contains the following columns: time, process identifier, system call. A simple raw example trace is shown in the following table.*

Index	*Time*	*PID*	*Syscall*
1	0	3	`open`
2	3	3	`read`
3	5	4	`open`
4	9	3	`read`
5	10	4	`close`
6	12	3	`close`
7	13	3	`open`
8	18	4	`read`
9	21	4	`read`

The parameter sets $\mathbb{P}_1$ to $\mathbb{P}_2$ will describe the process identifier and the system call columns with $\mathbb{P}_1 = \langle 3, 4 \rangle$ and $\mathbb{P}_2 = \langle$`open,read close`*$\rangle$. The trace entries will then be : $E_1 = \langle 0, \langle 3, \texttt{open} \rangle \rangle$, $E_2 = \langle 3, \langle 3, \texttt{read} \rangle \rangle$, and $E_3 = \langle 5, \langle 4, \texttt{open} \rangle \rangle$ with the specific example values of $E_2.t = 3$ or $E_3.P.p_1 = 4$. The trace T will therefore be $T = \langle E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9 \rangle$.*

3.1.2 Trace-Signal Association

Creating the formal model T is the starting point for mapping a trace to channels and signals. This section defines *signal class*, *signal*, and *channel* in relation to a trace.

DEFINITION 4 (Channel). *A channel $c_{j,k}$ models the parameter ρ_k in a parameter set $\mathbb{P}_j$, where $1 \leq k \leq |\mathbb{P}_j|$.*

DEFINITION 5 (Signal). *A signal $s_j := \{c_{j,k} | 1 \leq k \leq |\mathbb{P}_j|\}$ is the set of all channels for a parameter set.*

DEFINITION 6 (Signal Class). *A signal class S is the set of all signals for a given trace:*

$$S(T) := \{s_j | 1 \leq j \leq M\}$$

Following up on Example 1, the channel $c_{1,1}$ corresponds to the process identifiers with value 3. The signal s_2 models the system call column. The channel $c_{2,1}$ will therefore correspond to syscall `open`. The signal class S for the example trace is $S = \{s_1, s_2\}$ modeling both the process identifier and the system call column.

3.1.3 Time Series

Based on the definition of a signal class S, we can now define the time series through time stamps, and method of constructing a time series which is inter-arrival method.

DEFINITION 7 (Real-Time Stamp Sequence). *A real-time stamp sequence is the projection of a (filtered) trace onto time, thus* $\mathrm{rtss}(T, j, \beta) = T(j, \beta).t$.

DEFINITION 8 (Logical-Time Stamp Sequence). *A logical-time stamp sequence is the projection of a (filtered) trace onto index, thus* $\mathrm{ltss}(T, j, \beta) = T(j, \beta).idx$.

Time stamps are the means to construct time series of channels or in simpler words the channels. This work uses the inter-arrival time series construction method which is was presented in [30]. We refer to this time series as inter-arrival time series.

DEFINITION 9 (Inter-arrival time series for *rtss*). *Given a real-time stamp sequence* $\mathrm{rtss}(T, j, \beta)$*, inter-arrival real-time series* $\mathrm{rtss}_\triangle$ *is the difference function applied to subsequent time stamps:*

$$\forall 1 \leq l \leq L - 1 : \mathrm{rtss}_\triangle(T, j, \beta)[l] = \mathrm{rtss}[l+1] - \mathrm{rtss}[l]$$

with L being the length of the trace T.

We can define a similar function on the inter-arrival logical time series $ltss_\triangle$, which operates on the logical-time stamp sequence $ltss$.

DEFINITION 10 (Channel valuation). *Given an inter-arrival time series $ts_\triangle(T, j, \beta)$ (which is either* $\mathrm{rtss}_\triangle$ *or* $\mathrm{ltss}_\triangle$*), a channel $c_{j,k}$ has a value $c_{j,k} = ts_\triangle(T, j, \mathbb{P}_j.\rho_k)$.*

EXAMPLE 2. *We now construct* $\mathrm{rtss}_\triangle(T, j, \mathbb{P}_j.\rho_k)$ *with T as the trace shown in Example 1 and j being either 1 or 2 for the different columns.*

With $j = 1$, we create the signal for the PID *column and k specifying which concrete PID to model (e.g., $k = 2$ models the PID being 4 as 4 is the second member of $\mathbb{P}_1$).*

With $j = 2$, we create the signal for the Syscall *column and k specifying the concrete system call to model (e.g., $k = 1$ models* open *as it is the first member of $\mathbb{P}_2$).*

For different values of j and k, $c_{j,k}$ will look like the following:

j	$\mathbb{P}_j.\rho_k$	$rtss(T, j, \mathbb{P}_j.\rho_k)$	$c_{j,k} = rtss_\triangle(T, j, \mathbb{P}_j.\rho_k)$
1	3	(0,3,9,12,13)	(3,6,3,1)
1	4	(5,10,18,21)	(5,8,3)
2	`open`	(0,5,13)	(5,8)
2	`read`	(3,9,18,21)	(6,9,3)
2	`close`	(10,12)	(2)

3.2 Signal Processing Algorithm

Using the defined model, we can now specify what signal processing algorithms to apply and how. The previous section described a deterministic method for the first part of the anomaly detection technique that deals with trace-to-signal modeling. The output of this method, S, is a signal class that is the input of the second part of SiPTA. As discussed earlier, DFT is a tool to extract periodic features or more generally frequency properties of a time series. The main focus of our signal processing algorithm is DFT, how to perform it on the signal class of a trace, and how to scale the results.

DEFINITION 11 (FFT function). *Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{C}$ is the set of complex numbers. $\mathcal{F}$ be the function performing DFT using the fast Fourier's Transform (FFT) algorithm on its input on input positive-integer time series of length L. Then $\mathcal{F}$ is the FFT function on the input time series, i.e., the input positive integer vector of size L. Namely:*

$$\mathcal{F} : \mathbb{N}^L \rightarrow \mathbb{C}^L$$

In this work, the inputs to the FFT function are channels that are extracted by modeling the trace to signal classes.

For the vector, we require the functions *abs* and *sum*, which are the element-wise absolute value of the input vector and summation of the elements in the vector, respectively.

Using the time series of channels as inputs, we can now define how to compute the frequency series, for which we then normalize the area under the curve. For this normalized frequency series we can then define the frequency axis that our metrics and classifier will use.

DEFINITION 12 (Frequency Series). *For every $1 \leq j \leq M$, and every $1 \leq k \leq |\mathbb{P}_j|$, let $F_{j,k}$ be the frequency series associated with channel $c_{j,k}$ of length $L_{j,k}$. The frequency series $F_{j,k}$ is defined as:*

$$F_{j,k} = \text{abs}(\mathcal{F}(c_{j,k}))$$

DEFINITION 13 (Normalized Frequency Series). *For every $1 \leq j \leq M$, and every $1 \leq k \leq |\mathbb{P}_j|$, let $f_{j,k}$ be the normalized frequency series associated with channel $c_{j,k}$. Let $F_{j,k}$ be the frequency series of the channel $c_{j,k}$ of length $L_{j,k}$ defined as:*

$$f_{j,k} = \frac{F_{j,k}}{\sum_{l=2}^{L_{j,k}} (F_{j,k}[l]) / (L_{j,k} - 1)}$$

The normalization ignores the first element ($F_{j,k}[1]$), because we ignore the DC value of the frequency series for normalization.

DEFINITION 14 (Frequency Axis). *Let $f_{j,k}$ be a normalized frequency series of length $L_{j,k}$. The following defines frequency axis $\omega_{j,k}$ vector of length $L_{j,k}$ associated with $f_{j,k}$:*

$$\omega_{j,k}[l] = \frac{l-1}{L_{j,k}} \text{ with } 1 \leq l \leq L_{j,k} \tag{2}$$

So, $\omega_{j,k}[l]$ is the normalized frequency [23] associated with $f_{j,k}[l]$.

EXAMPLE 3. *Let $c_{1,1}$ and $c_{1,2}$ be the first two channels in Example 2, namely $c_{1,1} = (3, 6, 3, 1)$, and $c_{1,2} = (5, 8, 3)$. In this case, $L_{1,1} = 4$ and $L_{1,2} = 3$ are respectively the lengths of these channels. According to Definition 12 and 13, the values of define $F_{1,1}$, $f_{1,1}$, and $\omega_{1,1}$ are as following:*

$$\mathcal{F}(c_{1,1}) = (13, -5i, -1, 5i)$$
$$F_{1,1} = abs(\mathcal{F}(c_{1,1})) = (13, 5, 1, 5)$$
$$\frac{\sum_{l=2}^{L_{1,1}} (F_{1,1}[l])}{L_{1,1}} = \frac{11}{3}$$
$$f_{1,1} = \frac{F_{1,1}}{11/3} \approx (3.55, 1.36, 0.27, 1.36)$$
$$\omega_{1,1} = (0, 0.25, 0.5, 0.75)$$

Also, the values for $F_{1,2}$, $f_{1,2}$, and $\omega_{1,2}$ are as following:

$$\mathcal{F}(c_{1,2}) = (16, -0.5 - 4.3301i, -0.5 + 4.3301i)$$
$$F_{1,2} = abs(\mathcal{F}(c_{1,2})) \approx (16.0000, 4.36, 4.36)$$
$$f_{1,2} \approx (3.67, 1, 1)$$
$$\omega_{1,1} \approx (0, 0.33, 0.67)$$

It is worth noting that these examples are for the purpose of demonstration and therefore the values of $L_{j,k}$ are arbitrarily small. In real traces, values of $L_{j,k}$ are usually magnitudes larger. Figure 2 depicts a normalized frequency series (i.e., spectrum) of a channel corresponding to the THREAD-THREPLY parameter of a non-anomalous trace. Higher values in the spectrum close to $\omega \approx 0.25$ indicate a significant periodic behavior around this normalized frequency relative to other periodic behaviors. It means that the channel has a periodic feature with an approximate sequence length of $\frac{1}{\omega} \approx 1/0.25 = 4$.

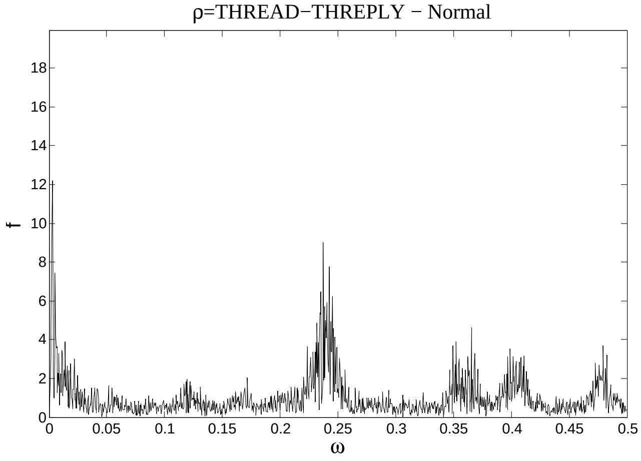

Figure 2: Spectrum of THREAD-THREPLY for a normal case

3.3 Metrics

Based on the spectra, we can compute metrics to distinguish between normal and anomalous traces. Consider a system that behaves normally, i.e., it does not show an anomaly. In this case, the system produces *normal traces* which are representative for the specification. Normal traces will result in *normal spectra* after applying our signal processing algorithm. Naturally, normal spectra of each parameter $\mathbb{P}_j.\rho_k$ for different normal traces result in similar periodic features. We expect that, in case an anomaly occurs which results in an anomalous trace, then this anomaly will have measurable impact on the spectrum.

For example consider the parameter THREAD-THREPLY in a trace obtained from one of our experiments. THREAD and THREPLY are individual parameters, and THREAD-THREPLY refers to the parameter obtained by concatenating them. Figure 2 shows the spectrum of the channel modeling this parameter in a normal trace. Figure 3 shows the spectrum of the same parameter with an anomaly present. Obviously, the two spectra differ and the second one can be labeled as an anomalous trace (assuming that the first shows a normal trace). For example the periodic behavior near $\omega \approx 0.25$ vanished in the anomalous trace.

Several metrics can help detecting periodic features. So far, SiPTA implements the following metrics: *peak-frequency* and *DC-significance*.

DEFINITION 15 (Peak-Frequency Metric). *Let $f_{j,k}$ and $\omega_{j,k}$ of lengths $L_{j,k}$ represent the spectrum of the parameter $\mathbb{P}_j.\rho_k$. Also let $0 \leq \omega_{j,k}^{\text{cut}} \leq 0.5$ be the minimum frequency cut-off representing a high-pass filter, and let $l^{\text{cut}} = \min\{l | \omega_{j,k}[l] \geq \omega_{j,k}^{\text{cut}}\}$ be the lowest index which its corresponding frequency is higher than $\omega_{j,k}^{\text{cut}}$. Basically, l^{cut} is the smallest index where $\omega_{j,k}[l^{\text{cut}}] > \omega_{j,k}^{\text{cut}}$. Similarly, let $l^{\text{half}} = \max\{l | \omega_{j,k}[l] \leq 0.5\}$ indicate the greatest index, where $\omega_{j,k}[l] \leq 0.5$. In this case, let $l^{\max} = \text{argmax}_{f_{j,k}[l]}(l^{\text{cut}} \leq l \leq l^{\text{half}})$ be the index for which the maximum magnitude in the spectrum of normalized frequency falls between $\omega_{j,k}^{\text{cut}}$ and $\omega_{j,k}^{\text{half}}$. In this case the metric peak-freq consists of the two values $f_{j,k}[l^{\max}]$, and*

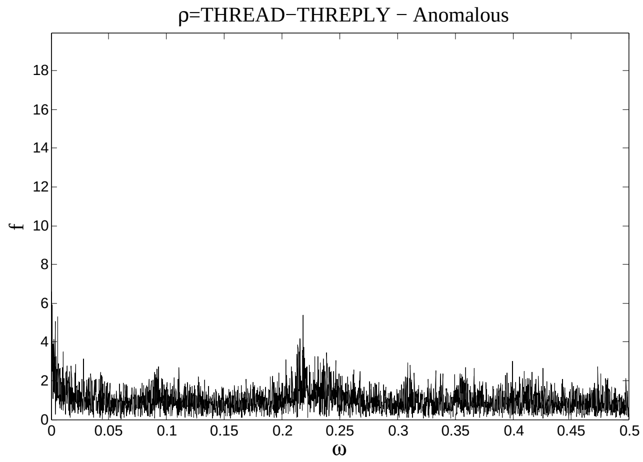

Figure 3: Spectrum of `THREAD-THREPLY` for an anomalous case

$\omega_{j,k}[l^{\max}]$. *Formally:*

$$P_{j,k} = f_{j,k}[l^{\max}]$$
$$F_{j,k} = \omega_{j,k}[l^{\max}]$$

where $P_{j,k}$ is the valuation for metric peak part of the metric and $F_{j,k}$ is the valuation for the frequency part of the metric. Basically, this valuation points to the point in the spectrum which has the highest magnitude while it is ignoring the values that has frequencies near 0*, i.e., DC. This metric represents the most significant periodic feature of the parameter.*

DEFINITION 16 (DC-Significance Metric). *Let $f_{j,k}$ be the normalized frequency domain of extracted with respect to the parameter $\mathbb{P}_j.\rho_k$ of lengths $L_{j,k}$. In this case, $DC_{j,k}$ representing the DC-significanceof the parameter $\rho_{j,k}$ is defined as below:*

$$DC_{j,k} = \frac{f_{j,k}[1]}{L_{j,k}}$$

In the case of inter-arrival time series, this metric points that how regularly the parameter $\rho_{j,k}$ occurred.

EXAMPLE 4 (Metrics). *Let $\rho =$* `THREAD-THREPLY`*. The valuation of peak-frequency and DC-significance metric on the two spectra of the two traces depicted in Figure 2 (normal trace), and Figure 3 (anomalous trace) are as following, where clean represent normal and dirty represent anomalous traces.*

Metric	Clean	Dirty
P	9.037	5.378
F	0.2374	0.2181
DC	0.0156	0.0081

As we can see from Example 4, there is a significant difference between peak and DC-significance metrics for normal and anomalous traces.

3.4 Classification

As mentioned earlier, *classification* consists of two phases, the training phase and the testing phase (also referred to as analysis phase). In the training phase, the classifier analyzes the values of each metric of each parameter of each training trace and calculates a mean value for each of them. For each trace in the analysis phase, SiPTA calculates the difference of each metric value with the computed mean. SiPTA does this for every channel and finally averages them with respect to some weight values. If the result is greater than a specified threshold, then the trace being evaluated is identified as anomalous, otherwise it represents a normal execution scenario.

To formalize different steps of classification, we should be able to distinguish between the notation for the training and testing traces. We do so by using superscripts. Therefore, a trace $T^{\mathfrak{t},\mathfrak{i}}$ represents the i-th training trace. Similarly, $T^{\mathfrak{a},\mathfrak{i}}$ represents the i-th testing trace. In a similar way, the length of the trace $T^{\mathfrak{t},\mathfrak{i}}$ will be denoted by $L^{\mathfrak{t},\mathfrak{i}}$, the value of DC-significance metric for a parameter $\mathbb{P}_j.\rho_k$ for valid j and k is denoted by $DC^{\mathfrak{t},\mathfrak{i}}_{j,k}$, and so on.

DEFINITION 17 (Training and Analysis Sets). *Consider an arbitrary parameter set sequence $\mathcal{P} = \langle \mathbb{P}_1, \ldots, \mathbb{P}_M \rangle$ with M parameter sets. Let $\mathbb{T}^{\mathfrak{t}} = \{T^{\mathfrak{t},1}, \ldots, T^{\mathfrak{t},\mathfrak{n}_{\mathfrak{t}}}\}$ and $\mathbb{T}^{\mathfrak{a}} = \{T^{\mathfrak{a},1}, \ldots, T^{\mathfrak{a},\mathfrak{n}_{\mathfrak{a}}}\}$ be the sets of training and analysis traces, each of which are defined over $\mathcal{P}$, where $\mathfrak{n}_{\mathfrak{t}}$ is the number of training and $\mathfrak{n}_{\mathfrak{a}}$ is the number of analysis traces.*

DEFINITION 18 (Metric Mean). *Let $\mathbb{T}^{\mathfrak{t}}$ be a training set of length $\mathfrak{n}_{\mathfrak{t}}$. For all $1 \le \mathfrak{i} \le \mathfrak{n}_{\mathfrak{t}}$, $1 \le j \le M$, and $1 \le k \le |\mathbb{P}_j|$, assume that $m^{\mathfrak{i}}_{j,k}$ is an arbitrary metric such as DC-significance, or peak-frequency value, for parameter $\mathbb{P}_j.\rho_k$ in trace $T^{\mathfrak{i}}$. In this case, metric mean of parameter $\mathbb{P}_j.\rho_k$, $\overline{m}_{j,k}$ is defined as following:*

$$\overline{m}_{j,k} = \frac{\sum_{\mathfrak{i}=1}^{\mathfrak{n}_{\mathfrak{t}}} m^{\mathfrak{i}}_{j,k}}{\mathfrak{n}_{\mathfrak{t}}}$$

Basically, it is the mean value of that metric from all the traces in training set.

DEFINITION 19 (Scores). *Let $T^{\mathfrak{i}}$ be a trace either in the training or analysis sets. Also, assume the parameters $\mathbb{P}_j.\rho_k$ are assigned to normalized weights $w_{j,k}$, that is, $\sum_{j=1}^{M}\sum_{k=1}^{|\mathbb{P}_j|} w_{j,k} = 1$, and all weights are positive. Now, the score assigned to $T^{\mathfrak{a},\mathfrak{i}}$ is the weighted average of squares of normalized differences of the metrics to the metric mean. Put formally:*

$$\widehat{m}^{\mathfrak{i}} = \sum_{j=1}^{M}\sum_{k=1}^{|\mathbb{P}_j|} (m^{\mathfrak{i}}_{j,k} - \overline{m}_{j,k})^2 \times w_{j,k}$$

DEFINITION 20 (Overall Scores). *In order to merge the scores of different metrics to compute an overall score, we calculate the average of the individual metric-scores. For example, for a trace $T^{\mathfrak{i}}$, the overall score* (score$^{\mathfrak{i}}$) *over the DC-significance, and peak-frequency metric, will be the following:*

$$\text{score}^{\mathfrak{i}} = \frac{\widehat{DC}^{\mathfrak{i}} + \widehat{P}^{\mathfrak{i}} + \widehat{F}^{\mathfrak{i}}}{3}$$

The average presented in Definition 20 can be replaced with a weighted average that signifies more important metrics. The reason that we did not do a weighted average, is that it we did not have an estimate of which metric is more significant.

DEFINITION 21 (Maximum Training Score). *This is defined as the maximum overall score over all traces in the training set.*

$$\text{scoreMax} = \max(\text{score}^{\mathfrak{i}})\ \forall i \textit{ such that } T^{t,i} \textit{ is a training trace}$$

DEFINITION 22 (Classification). *Let r be a multiplication factor which in turn defines the threshold factor of* scoreMax $\times$ r*. And also let* score$^{\mathfrak{a},\mathfrak{i}}$ *be the score for $T^{\mathfrak{a},\mathfrak{i}}$ from the analysis set. Then the following function determines which class ($C^{\mathfrak{a},\mathfrak{i}}$) this particular trace belongs to:*

$$C^{\mathfrak{a},\mathfrak{i}} = \begin{cases} \text{false} & \textit{if } \text{score}^{\mathfrak{a},\mathfrak{i}} \leq \text{scoreMax} \times r \\ \text{true} & \textit{otherwise} \end{cases}$$

We can vary the factor r, to define new threshold levels.

EXAMPLE 5 (Scoring and Classification). *For a studied case, the training set had 5 members and analysis set had 10 members which contained 5 normal and 5 anomalous traces. Traces 1 to 5 construct the training set. Traces 6 to 10 are the normal traces in the analysis set, and traces 11 to 15 are anomalous traces in analysis set. Figure 4 gives the score values for these traces.*

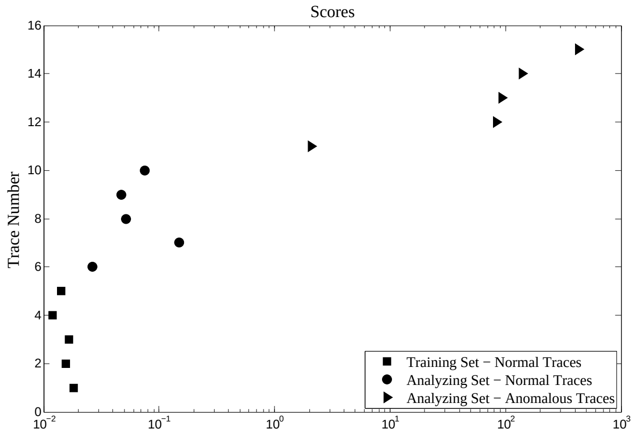

Figure 4: Classification scores for a system

In this example ScoreMax $= 0.0183$ *and with a threshold value r between 9 and 100, the classifier will provide a perfect classification of anomalous and normal traces.*

Figure 4 also shows that the scores for normal training traces are clearly more than these scores for clean analyzing traces. The reason is that expansion of the defintion $\widehat{DC}^{\mathfrak{i}}$, $\widehat{P}^{\mathfrak{i}}$, and $\widehat{F}^{\mathfrak{i}}$ in the score$^{\mathfrak{i}}$ *for a real example, it will be the weighted mean of squares of many (in order of* 1000*) of metrics. On the other hand the square function magnifies the difference from score mean which in turn results in high expected score for clean analyzing set compaired to the training set.*

4. Experimental Evaluation

While SiPTA in theory is well suited to the problem domain of periodic systems, we followed up our theory work with an experimental evaluation to provide quantitative results. The following section describes the conducted experiments used to evaluate the proposed technique, the evaluation process, and the results.

4.1 Experimental Setup and Workload

Our experiments use the QNX Neutrino 6.4 real-time operating system. While the framework is independent from system traces, we used QNX, because we have three distinct applications from which we can gather traces: a hexacopter application [4], a QNX CAR infotainment unit, and Blackberry phones running QNX. This allows us to gather system traces from very different execution contexts and properly evaluate SiPTA.

To collect data, we used the QNX kernel logging facility for our experiments. The kernel logging facility provides trace capabilities through QNX tracelogger. A trace collected from tracelogger contains a chronological order of system events such as system calls, message passes, interrupts, I/O etc. Every trace entry corresponds to a system event and a set of additional related information, such as the source/destination of a message pass. A sample trace is shown in the following snippet:

```
TIMESTAMP, CPU, EVENT,      PID, PROC, Details
t1,        1,   PROCCREATE, 1,   A,    PPID: 0 ...
t2,        1,   THCREATE,   1,   A,    ...
t3,        2,   INT_ENTR,   1,   A,    ...
t4,        2,   INT_EXIT,   1,   A,    ...
t5,        1,   MSG_SND,    1,   A,    To: B ...
t6,        1,   MSG_RECV,   3,   B,    From: A ...
.....
tn,        1,   KER_CALL,   1,   A,    SIGKILL ...
```

The above snippet shows the life-cycle of a process as recorded by the system. The process A, once created, spawns a thread, addresses an interrupt, sends message to some other process B before being terminated.

Our experiment uses only a subset of all the information that is contained in a trace. Specifically, our comparisons only consider the following event attributes: *class, event, time, pid, and process name*. This is consistent with related work [10] which reduces the used attributes to similar lists.

The experimental workload consists of different execution scenarios of the available systems. Each set contains normal and anomalous traces. The traces are not shared between the scenarios. Each scenario runs between 10 to 20 seconds. Each trace contains all logging information for a single scenario, and within one scenario each trace is approximately the same length in terms of run time. For a given scenario, we split the set of traces arbitrarily into a training subset and a testing subset where a trace belongs to only one of those subsets. Depending on the detection technique, the training set can contain just normal traces or it can contain a combination of normal and anomalous traces (e.g., we need anomalous traces as part of the training set for the comparison with Neural Networks). Similarly, the testing set can contain a set of normal traces, a set of anomalous traces, or a combination of both.

The experimental setup and workload is specific for embedded systems. In particular, we picked four embedded devices (details later in the section), and used some test scenarios to collect traces. As most events in embedded systems are periodic in nature, our test cases attempt to cover general cases that exhibit periodicity. Each scenario contained two sets of traces: $\mathbb{T}^t$ and $\mathbb{T}^a$. Of these sets, $\mathbb{T}^t$ trains the system with relevant information to be able to identify anomalous traces from the set $\mathbb{T}^a$. Recall that an anomalous trace is one that does not conform with the expected behavior learnt from the training traces.

- *Hexacopter*: The first scenario uses traces from an unmanned aerial vehicle (UAV) platform, which implements non-trivial software and systems control [4]. The platform runs on beaglebone white with ARM Cortex-A8 processor and comprises a networked system of hardware and software components. The hexacopter is field tested through several mission-critical applications including iceberg monitoring on the open sea and creating infrared maps over critical Canadian Solar infrastructure. For our experiment, we created two test scenarios. In both scenarios, the training traces correspond to the hexacopter running normally. The classes of anomalous traces involved, for instance, a periodic recursive listing on the file system[1] and getting stuck in a tight loop.
- *QNX CAR infotainment device*: The second set of scenarios uses QNX CAR [5], which is the leading in-car infotainment system. The device was running QNX Neutrino on an *i.MX6Q* (Sabre lite) board with an ARM Cortex A9 quad-core proces-

[1] `while [ 1 ]; do sleep 2; ls -lR /; done`

sor. We created the following scenarios on this platform:

Normal trace	Anomalies
Idle	Induced network traffic, user inputs
Play MP3	Seeking, fast-forward, different song
Play Video	Seeking, fast-forward, different video
Run *top* command	Induced network traffic, more shell tasks

- *QNX-based BlackBerry Z10 phone*: The third set of scenarios uses the BlackBerry phone. These phones run a modified version of QNX Neutrino, however, the tracelogging facilities are still available. We created the following set of scenarios on this platform:

Normal trace	Anomalies
Record video	Zooming and toggling the camera light on/off
Play youtube video	Playing different video, toggling HD on/off
Play game	Change sound, using different controls
Run flash applet	Reload page, leave page

- *Embedded target*: This scenario uses a non-commercial demonstrator platform for QNX Neutrino [6]. The hardware runs on the *i.MX6Q* (sabre lite) similar to our QNX CAR case study. For this scenario, the platform executes a sound alarm application displaying audio frequencies recorded through the microphone. The normal behavior runs the system in a quiet setup. The anomalies include producing different loud sounds nearby.
- *Variable delay traces*: Finally, we ran SiPTA on a set of traces which were generated on QNX Car platform by varying the delay between the events. We emulated this behavior in the traces by inserting delays between two instances of high system activity. For example, a scenario in which the device pings the network every other second, and another in which the interval is increased to 5 seconds. Considering the slow traces as normal and the faster ones as anomalies, SiPTA was able to correctly identify and appropriately label the traces as we show in Section 5.

SiPTA has varying parameters that we should set for experiments. These parameters are the channels and signals that SiPTA uses, time stamps, and the classification weights $w_{j,k}$. In this experiment setup, the trace-to-signal modeling phase extracts the signal class as a concatenation of the CLASS, EVENT, and PID columns. Also, the time stamps are real-time stamps. Classification scores are calculated using the following weight values:

$$w_{j,k} = \frac{\bar{m}_{j,k} / \mathrm{dev}_{\mathrm{i}}(m^{\mathrm{i}}_{j,k})}{\sum_{j=1}^{M} \sum_{k=1}^{|\mathbb{P}_j|} \left(\bar{m}_{j,k} / \mathrm{dev}_{\mathrm{i}}(m^{\mathrm{i}}_{j,k}) \right)} \tag{3}$$

where $\mathrm{dev}_{\mathrm{i}}(m^{\mathrm{i}}_{j,k})$ is the standard deviation of the metric for parameter $\mathbb{P}_j.\rho_k$ This formula gives more weight to parameters that show less deviation in training phase.

4.2 Evaluation Criteria

To measure the effectiveness of SiPTA, we need to compare the results to different existing techniques. The prevalent metric for comparison in anomaly detection is receiver operating characteristics curve (ROC) analysis [14].

ROC Analysis

ROC analysis is a common technique in research to compare different classifiers based on their performance [22, 29]. ROC analysis explains the trade-off between the true positive rate (TPR), plotted on the y-axis, and the false positive rate (FPR), plotted on the x-axis. To compare different classifiers, the common approach [14] is to consider the classifier with the higher area under the ROC curve as the better classifier.

Figure 5 shows an ROC curve for one of the experiment runs. The algorithm calculates the values for a point *(FPR, TPR)* using Equations 4 and 5 where a classifier threshold interprets the input probabilities into a binary output of 0 (negative) or 1 (positive).

$$\mathrm{TPR} = \frac{\text{Positives correctly classified}}{\text{Total Positives}} \tag{4}$$

$$\mathrm{FPR} = \frac{\text{Negatives incorrectly classified}}{\text{Total Negatives}} \tag{5}$$

To obtain all points for the ROC curve when using Markov Model, the algorithm varies the threshold over a range of all input probabilities to obtain the corresponding points (FPR, TPR) plotted in the figure. In SiPTA, this corresponds to varying the value r.

In addition to the area under the ROC curve, some important characteristics of the curve help with the analysis of the classifier performance. For example, point $(0, 1)$ indicates perfect classification while the region under the dotted line TPR $=$ FPR indicates that the classification is worse than making a random guess.

4.3 Comparison to Alternative Approaches

To evaluate SiPTA, we compare it to other approaches implementing alternative concepts. For example, one alternative uses a stochastic approach, which assumes that anomalies change the probabilistic characteristics of event transitions in a sequence of system events. The most popular technique for this approach is the first-order Markov Model technique [11]. Another alternative that we use for comparison is Neural Networks [9]. The two techniques were chosen due to their popularity in the domain and thus a wider applicability for existing work. The disadvantage of using Neural Networks however is the inability to use the ROC analysis to compare its performance to the other techniques, as discussed later in this section.

4.3.1 Markov Model Technique

Markov Model is a discrete-time stochastic process used to study the probability of the change of a random variable value. First-order Markov Models are commonly used to study the probabilistic characteristics of a single transition between two events within the trace sequence [16, 27–29].

To compare to the performance of SiPTA, we implemented the anomaly detection engine shown in Figure 1 using first-order Markov Model. For an input trace sequence, an *event* represents a Markov Model state so that the Markov Model will describe the probability of occurrence of a transition between an *event* and its first successor. Following the work-flow in Figure 1, the preprocessor splits the trace into sub-traces based on the *process name* and extracts only the *event name* and *event class* attributes. For each sub-trace, the Markov Model calculates a transition probability matrix [27], which indicates the probability of transition between any trace entries. The averaged transition probability matrices calculated for the training set describe the normal behavior of the system.

In the testing phase, the classifier compares the transition probability matrix of the test trace and the normal behavior matrix to decide if the test trace is normal or anomalous. We assign an anomaly flag for each transition in the test trace that occurs with a probability value that lies outside a defined region around the mean value of probabilities that describe the normal behavior of the system. For each experiment, we performed several experiment runs using different sizes for that region to select the region size that yields the best ROC curve. For the final binary classification, the percentage of anomalous transitions within the trace indicates, whether the trace is anomalous or not. Varying a threshold over the range of the percentages, ranging from 0% to 100%, yields points of ROC curve as described earlier.

For the sake of simplicity of the implementation, transition probability matrices consider only the transitions in a randomly selected normal trace during the training phase instead of considering all possible transitions combinations. This consideration reduces the calculations by excluding the transitions that rarely occur and have no effect on the final classification result.

4.3.2 Neural Networks Technique

Artificial Neural Networks (NN) are massively connected networks of computational nodes or neurons. The nodes are usually organized into layers (input, output, and hidden layers) with weighted connections between them [17]. As the network learns, it updates the weights on the connections to improve classification.

For the purpose of comparison, the Kohonen self-organizing network (KSON) was used. The network uses unsupervised learning algorithms to cluster inputs into groups with similar characteristics. The learning is called unsupervised because the output characteristics of the network are determined internally and locally by the network itself, without any data on desired outputs. The nodes distribute themselves across the input space to recognize groups of similar input vectors, while the output nodes compete among themselves to be fired one at a time in response to a particular input vector [17]. Thus, due to this competitive learning, similar input vectors activate physically close output nodes. We want to take advantage of this characteristic of KSON to classify the traces.

The input vectors for the network are an encoded representation of the events in a trace. To generate the encoding we extract event names from the trace and then count the number of occurrences of each event. The count for each event is then scaled by dividing it by the total number of logged events in the trace. The input vector is thus a collection of event to count mappings for the trace.

The training sets for the network need to contain both clean and anomalous traces. When the network is trained with only clean traces, it is not able to classify anomalous traces as part of a different cluster during testing. Thus, unlike other approaches, Neural Networks imposes constraints on the training set. During the testing phase, the network determines the cluster that the trace belongs to and thus classifies the trace. The classification is typically discrete, with the output being either 0 (clean) or 1 (anomalous), however the value can be within that interval in case of more uncertainty.

The difficulty with using Neural Networks technique for comparison with the other approaches is the lack of a classifier threshold. The classification takes place within the internal structure of the network using specialized learning algorithms. We can thus alter the structure of the network, but cannot alter any thresholds that influence the cluster that the network will choose. As a consequence, we can only report the detection rate for this technique but cannot perform an ROC analysis.

5. Results

Table 1 shows the detection rates (TPR) and false-alarm rates (FPR) for each of the three approaches, namely SiPTA, Markov Model and Neural Networks. Contrary to Neural Networks, SiPTA and Markov Model implement a binary classifier. This allows us to use ROC curves to compare their performance, but a similar comparison cannot be done with Neural Networks.

As mentioned in Section 4.2, the points (FPR, TPR) asymptotically represent the ROC curve for each approach. Figures 5, 6, and 7 show the ROC curves for the hexacopter, QNX-Car run *top* command, and variable ping speed experiments, respectively. The remaining experiments yielded ROC curves similar to Figure 5 which show near perfect classification for both SiPTA and Markov Model.

ROC curves clearly demonstrate the trade-off between the detection rate and the false-alarm rate for a binary classifier. ROC curve points closer to $(0, 1)$ indicate better results with 100% detection rate and 0% false alarm rate. Although such points are most desirable, for our comparisons in Table 1, we favor the detection rate over false-alarm rate. The reason behind this preference is that for most cases the penalty of false negatives outweighs false positives.

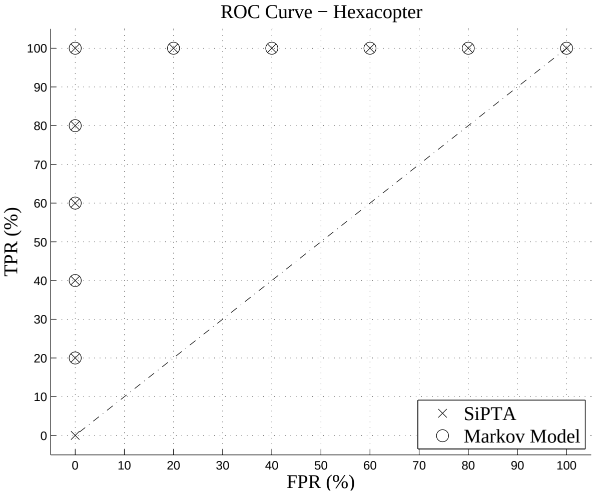

Figure 5: ROC curve for hexacopter

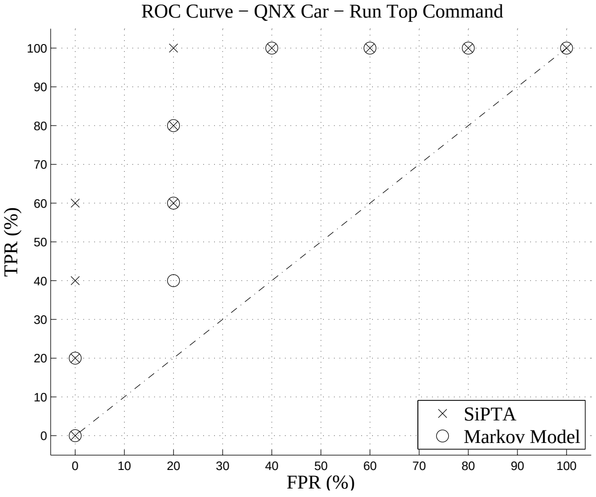

Figure 6: ROC curve for QNX car-run *top* command

6. Discussion

SiPTA outperforms all other approaches. In every studied case, our approach yields better results than other studied approaches. For instance, as Figure 7 indicates, SiPTA yields perfect classification results for the variable speed ping scenario while other techniques do not. One conjecture for why SiPTA worked is that a

	SiPTA		Markov Model		Neural Networks	
Case studies	Detection Rate	False-Alarm Rate	Detection Rate	False-Alarm Rate	Detection Rate	False-Alarm Rate
Hexacopter	100%	0%	100%	0%	91%	0%
QNX-Car Idle	100%	0%	100%	0%	91%	0%
QNX-Car Play MP3	100%	0%	100%	0%	45%	0%
QNX-Car Play Video	100%	0%	100%	0%	83%	0%
QNX-Car Run *top* command	100%	20%	80%	20%	26%	0%
Variable speed ping	100%	0%	80%	0%	75%	0%

Table 1: Results summary

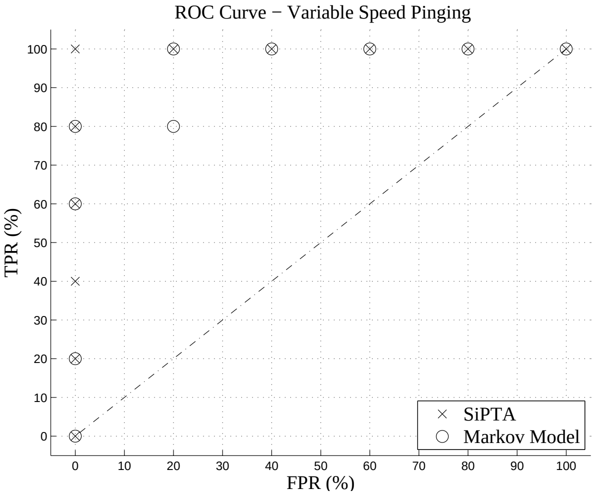

Figure 7: ROC curve for QNX car-variable speed ping

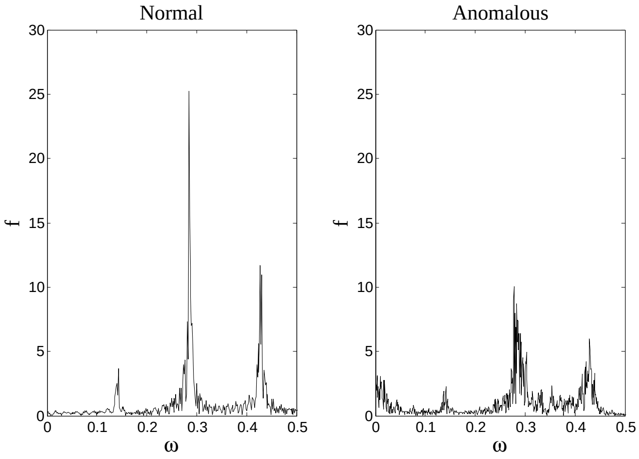

Figure 8: Spectra of parameter `THREAD-THREADY-2367525`

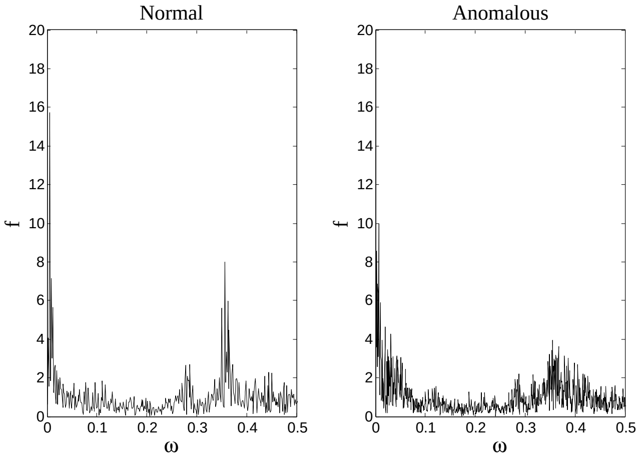

Figure 9: Spectra of parameter `THREAD-THREADY-1`

change in the ping frequency introduces more irregular behavior in the trace (or less depending on the direction of the rate change). With this irregularity, more noise will appear on the spectra of the channels. As these spectra are normalized, this noise grows in significance as the peaks show less magnitude. Because for this case, we consider higher ping speed as anomaly, the peak-frequency and DC-significance metric to have lower values in anomalous case. For example, Figure 8 shows two spectra for the parameter `THREAD-THREADY-2367525`; one plot for a normal and the other for an anomalous case. Figure 9 also shows two spectra, in this case for the parameter `THREAD-THREADY-1`. The distinction between the peak-frequency values for the normal and anomalous cases is visible and obvious.

Markov Models fail on certain scenarios. The Markov Model results indicate that the technique does not work for all experiments as SiPTA does. The Markov Model technique calculates the probabilities of transitions between events where the classifier aims to find any irregularities among those probabilities. Although the technique seems to work reasonably well and is currently the dominantly used one [10], it failed to handle certain scenarios. We conjecture that Markov Model fails for these scenarios, because the anomaly has an insignificant effect on the transition probabilities. This is due to events changing their inter-arrival time without affecting their transitions probabilities, which represents a whole class of anomalies that SiPTA can detect, however, Markov Model cannot.

Neural Networks are ill-suited for trace-based anomaly detection. Neural Networks is an established technique that is used for detecting anomalies, however, due to its limitations, it is less suitable for this given problem. The results for Neural Networks indicate that the technique classifies anomalous traces only for some of the scenarios. Neural Networks are only able to classify anomalous traces unless trained with at least one anomaly similar to the one occurring. Table 1 shows that Neural Networks have a 0% false alarm rate throughout the experiments, because when the network is not trained with anomalous traces, then it will tend to default to classifying traces as normal. So, if the occurring anomalous trace is not similar enough to any anomalous trace used in training, then the network will classify the trace as normal. Unlike SiPTA and Markov Model, Neural Networks require a different training set comprising both normal and anomalous traces.

The concepts are widely applicable. It can be observed that SiPTA has an inherently modular structure. Each component, namely trace-to-signal modeling, signal processing algorithm, metrics, and classification technique can be modified to suit application specific scenarios as long as the modules complement each other. For example, one can use some alternative method to model trace-to-signal mappings other than assigning channels to each parameter. Similarly, one can plug-in another signal processing algorithm that exploits some other domain knowledge.

Threats to validity. Our framework and SiPTA base on the assumption that anomalies show changes in the periodicity of events in the trace; otherwise such anomalies will pass undetected. Furthermore, similar to all other approach, SiPTA assumes that the trace contains evidence of the anomaly. Our approach still needs supervised learning from a labeled trace set, similar to Markov Models. While SiPTA was able to outperform all other studied approaches and we created a comprehensive set of scenarios and traces, naturally more evidence is necessary to gain confidence that the system will consistently outperform other approaches also in other settings.

7. Conclusion

Identifying an incorrect behavior is crucial for safety-critical embedded systems. In this work we demonstrated that, with an appropriate application of signal processing algorithms on execution traces, one can identify an incorrect system behavior. We demonstrated the feasibility of such an approach by implementing these algorithms into SiPTA. In our experiments, we performed a holistic evaluation of SiPTA by running it on execution traces from varied execution scenarios. To demonstrate the effectiveness of SiPTA, we compared it with state of art techniques of Markov Model and Neural Networks. The results indicate that SiPTA outperforms all the studied contemporary approaches.

References

[1] Audit data from MIT Lincolin lab. URL http://www.ll.mit.edu/mission/.

[2] QNX Neutrino RTOS. URL http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html.

[3] System call dataset from University of New Mexico. URL http://www.cs.unm.edu/∼immsec/data-sets.htm.

[4] Unmanned Aerial Vehicle (UAV). URL https://uwaterloo.ca/embedded-software-group/projects/unmanned-aerial-vehicle-uav-exemplar.

[5] QNX CAR Platform for Infotainment, . URL http://www.qnx.com/products/qnxcar/.

[6] QNX Accelerator Kits, . URL http://www.qnx.com/products/reference-design/acceleratorkits.html.

[7] ISO-26262, Road vehicles – Functional safety, 2011.

[8] DO-178C, Software Considerations in Airborne Systems and Equipment Certification, 2012.

[9] A. K. Ghosh and A. Schwartzbard. A Study in Using Neural Networks for Anomaly and Misuse Detection. In *Proc. 8th USENIX Security Symposium*, pages 23–36. USENIX, 1999.

[10] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[11] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection for Discrete Sequences: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[12] X. Cheng, K. Xie, and D. Wang. Network Traffic Anomaly Detection Based on Self-Similarity Using HHT and Wavelet Transform. In *Fifth International Conference on Information Assurance and Security, IAS*, volume 1, pages 710–713. IEEE, 2009.

[13] G. Creech and J. Hu. A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns. 2013.

[14] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[15] J. Gao, G. Hu, X. Yao, and R. K. C. Chang. Anomaly Detection of Network Trafc Based on Wavelet Packet. In *Asia-Pacific Conference on Communications. APCC*, pages 1–5. IEEE, 2006.

[16] S. Jha, K. MC. Tan, and R. A. Maxion. Markov Chains, Classifiers, and Intrusion Detection. In *csfw*, volume 1. Citeseer, 2001.

[17] F.O. Karray and C. De Silva. *Soft Computing and Intelligent Systems Design: Theory, Tools and Applications*, volume 1. Pearson Education Limited, 2004.

[18] Leveson, N. G. and Turner, C. S. An Investigation of the Therac-25 Accidents. *Computer*, 26(7):18–41, 1993.

[19] Lions, J.-L. Ariane 5 flight 501 failure, 1996.

[20] J. W. S. Liu. Real-Time Systems, 2000.

[21] W. Lu and A. A. Ghorbani. Network Anomaly Detection Based on Wavelet Analysis. *EURASIP Journal on Advances in Signal Processing*, 2009, 2009.

[22] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture. A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules. 2013.

[23] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing*, volume 2. Prentice-hall Englewood Cliffs, 1989.

[24] Pimentel, A. D. and Hertzbetger, L. O. and Lieverse, P. and Van Der Wolf, P. and Deprettere, E. F. Exploring Embedded-Systems Architectures with Artemis. *Computer*, 34(11):57–63, 2001.

[25] S. Rawat and C. S. Sastry. Network Intrusion Detection Using Wavelet Analysis. In *Intelligent Information Technology*, pages 224–232. Springer, 2005.

[26] M. Salagean and I. Firoiu. Anomaly Detection of Network Traffic Based on Analytical Discrete Wavelet Transform. In *8th International Conference on Communications (COMM)*, pages 49–52. IEEE, 2010.

[27] N. Ye and X. Li. A Markov Chain Model of Temporal Behavior for Anomaly Detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, pages 171–174. Oakland: IEEE, 2000.

[28] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu. Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(4):266–274, 2001.

[29] N. Ye, Y. Zhang, and C. M. Borror. Robustness of the Markov-Chain Model for Cyber-Attack Detection. *IEEE Transactions on Reliability*, 53(1):116–123, 2004.

[30] M. Zhou and S. D. Lang. Mining Frequency Content of Network Traffic for Intrusion Detection. In *Proceedings of the IASTED International Conference on Communication, Network, and Information Security*, 2003.