Anomaly Detection Using Inter-Arrival Curves for Real-time Systems

Mahmoud Salem
University of Waterloo, Canada
m4salem@uwaterloo.ca

Mark Crowley
University of Waterloo, Canada
mcrowley@uwaterloo.ca

Sebastian Fischmeister
University of Waterloo, Canada
sfischme@uwaterloo.ca

***Abstract*—Real-time embedded systems are a significant class of applications, poised to grow even further as automated vehicles and the Internet of Things become a reality. An important problem for these systems is to detect anomalies during operation. Anomaly detection is a form of classification, which can be driven by data collected from the system at execution time.**

We propose inter-arrival curves as a novel analytic modelling technique for discrete event traces. Our approach relates to the existing technique of arrival curves and expands the technique to anomaly detection. Inter-arrival curves analyze the behaviour of events within a trace by providing upper and lower bounds to their inter-arrival occurrence. We exploit inter-arrival curves in a classification framework that detects deviations within these bounds for anomaly detection. Also, we show how inter-arrival curves act as good features to extract recurrent behaviour that these systems often exhibit. We demonstrate the feasibility and viability of the fully implemented approach with an industrial automotive case study (CAN traces) as well as a deployed aerospace case study (RTOS kernel traces).

I. Introduction

The design and implementation of embedded systems are complex tasks that involve developing heterogeneous sub-components [1], often these systems must meet specified customer requirements and standards such as ISO-26262 for automotive functional safety or DO-178C for airborne systems. Analysis of event traces may provide an approach for studying the conformity of embedded systems behaviour to specified requirements. Such an approach to post-mortem analysis is well-suited to the study of embedded systems, as these systems typically generate event traces as part of the normal system design thus eliminating the need for special equipment that could affect system behaviour during analysis.

Several challenges face the trace analysis approach. One challenge is the high computational expense of the analysis of large amounts of data within an event trace [2]. In addition, the features extracted should allow for human interpretation to make it easier to comprehend the system behaviour, because trace analysis techniques consider the system under scrutiny as a black box without much knowledge of its complex internals. Hence, the need for methods to extract features specific to real-time systems allowing for efficient and effective reasoning about these systems behaviour.

Since embedded systems are bespoke and often implement recurrent behaviour, we expect the event traces generated by these systems to reflect such recurrence. For example, consider an event trace from a remote temperature measurement device with the following normal operation scenario: sample the temperature reading once a second, filter the sensor data, then send the data over a communication network to a base station. That device would generate a trace of events that reflect every stage of this scenario repeatedly, e.g., *read_sensor*, *filter_data*, *send_msg*, and *ack_msg*. As a result, extracting the recurrent behaviour from a system trace would provide a black-box understanding for the system operation. This would enable reasoning about any deviations from the model due to sensor failures or loss of communication, for example. Different classes of applications implement similar recurrent operation such as control applications, sensor fusion applications, etc.

We introduce *inter-arrival curves* as a specific form of arrival curves [3] to model the system behaviour using lower and upper bounds of inter-arrival occurrences of events within a trace. The model allows for the analysis of the system behaviour with respect to an expected system behaviour. Arrival curves were introduced as part of Network Calculus, and are used to provide bounds to data flow in integrated services networks. Real-time Calculus uses a variant of arrival curves for performance analysis [4], [5] and analytical modelling for event streams [6]. Our work is the first to generalize the concept of arrival curves via using inter-arrival curves to characterize embedded systems for statistical learning. Previous work has explored monitoring network messages for anomaly detection applications [7] and studying system correctness using online runtime monitoring [8], [9].

Problem Statement and Our Approach

Informal problem statement: *Given a set of event traces generated by a well-specified system in a given execution scenario, check whether a new trace from the same system originates from the same execution scenario.*

The problem statement defines a semi-supervised classification problem. Our work targets the problem through a framework whose building blocks are shown in Figure 1. Using the given training set of event traces, also known as normal traces, the output of the framework is a decision whether a test trace is conforming to that training data. The framework uses inter-arrival curves for extracting high-level features from the event traces, then builds a training model using these features to reason about the corresponding features extracted from the test trace.

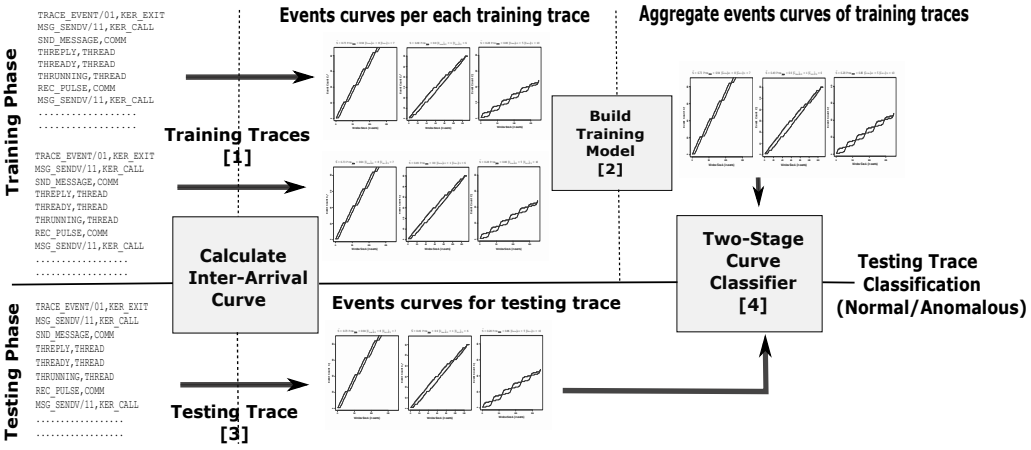


Fig. 1: Anomaly Detection Framework

Event traces can have various formats that require different trace analysis techniques to obtain the relevant information for analysis purposes [10]. Our work applies to event traces composed of discrete event streams such as streams of function calls from a call stack or network messages sent over a communication bus. We use inter-arrival curves to analyze traces composed of process events streams generated by a real-time system kernel during execution.

The contributions of this work are as follows:

- Introduce inter-arrival curves as a novel modelling technique for analysis of event traces.
- Derive higher-level features and quantifiable shape-based metrics from inter-arrival curves.
- Apply inter-arrival curves to a classification framework for anomaly detection.
- Validate empirically the suitability of inter-arrival curves for extracting recurrent behaviour.

II. Related Work

Network Calculus and Real-time Calculus provide the basis for arrival curves. Network Calculus [3] provides the theory for requirement specification that applies cumulative functions in time to describe data flow in the networks domain. Network Calculus uses arrival curves to provide guarantees to data flows in integrated services networks by defining bounds on the number of bits seen in the flow within a defined time interval.

Real-time Calculus [11] introduced a framework to extend arrival curves to be more widely applicable for performance analysis and analytical modelling of arbitrary event streams generated by real-time systems. Real-Time Calculus uses interval bound cumulative functions, where the functions are cumulative over time intervals, i.e., $f(\Delta t)$ instead $f(t)$ in Network Calculus. Further work on event streams using Real-Time Calculus aimed to reduce the exhaustiveness of that analytic approach. [4] uses a coarser-grained approach that abstracts event streams by defining coarse events as a collection of fine events before proceeding to calculate arrival curves of RTC. [12] used a different approach which considers events that occur within a defined time step without considering their exact occurrence time within that time step. More closely related to our work, Event Count Curves (ECC) introduced in [6] describes the occurrence of event types in a structured event stream in contrast to arrival curves that concerns with the timing of those occurrences. ECC curves consider an interval Δ of all event occurrences to define lower and upper bounds of different observed event counts.

A survey [10] summarized the research on anomaly detection using a sequence of events which can also defined as ordered series of events. Our work fits into the category of sliding window-based anomaly detection techniques. The main related research work uses hidden markov models (HMM) [13], rule inference [14] to build finite state automata (FSA) [15]. Related to the sliding window technique used by our work, episode mining [16] extracts collections of events that are partially ordered and occurring relatively close to each other within a defined window. Episode mining is used for anomaly detection [17] and software specification [18].

Inter-Arrival Curves vs Arrival Curves

We differentiate our inter-arrival curves from arrival curves provided in Network Calculus and Real-Time Calculus through the following key differences:

First, we use *cumulative functions $f(\Delta)$ over sliding windows of Δ discrete events instead of time intervals*. Second, *any sliding window must start by an event of the monitored type ε*. In other words, we only consider the arrival behaviour of events of type ε after the occurrence of an event of similar type, i.e., inter-arrival behaviour of events of type ε.

III. Inter-Arrival Curves for Event Traces

In this section, we present the definitions that form the basis of inter-arrival curves and provide a sample computation for the curves using a synthetic trace.

Definition 1: (Event) An event e is a tuple of system-defined values, denoted as $e = \langle v_1, v_2, \ldots \rangle$.
An event type, denoted as ε, belongs to a finite set of unique values generated by a given system.

Note that the value of event e does not include a timestamp, as inter-arrival curves do not consider the occurrence timing of the discrete events in contrary to arrival-curves.

Definition 2: (Trace) A trace $\widehat{T}$ is an ordered sequence of events e_i generated by a given system, where i is an index associated with the event in the sequence. Formally, $\widehat{T} = \{e_1, e_2, \ldots, e_{|\widehat{T}|}\}$ where $|\widehat{T}|$ is the number of events in a trace $\widehat{T}$, i.e., the length of the trace.

Definition 3: (Trace Mapping Operator) To count the multiple occurrences of a single event, we map the trace to just two symbolic events. The operator $\uparrow$ maps a trace $\widehat{T}$ from Definition 2 to a trace T of two symbolic event types ϵ and $\bar{\epsilon}$ by replacing every event e in $\widehat{T}$ as follows:

$$\widehat{T} \uparrow \varepsilon \mapsto T, \text{ where } e = \begin{cases} \epsilon, & \text{if } e = \varepsilon \\ \bar{\epsilon}, & \text{if } e \neq \varepsilon \end{cases}, \forall e \in \widehat{T} \qquad (1)$$

Example 1: Consider the following trace example $\widehat{T} = \{aabcaabbcaaabbacccaa\}$, there exists $|\widehat{T}| = 20$ events of event types $\varepsilon \in \{a, b, c\}$.

Given the sample trace from Example 1, the corresponding mapped trace T for an event type c as a result of applying the mapping operation $\widehat{T} \uparrow c$ is shown in Figure 2.

$$\widehat{T} = \{\ a\ a\ b\ c\ a\ a\ b\ b\ c\ a\ a\ a\ b\ b\ a\ c\ c\ c\ a\ a\ \}$$
$$T = \{\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}\ \underbrace{\epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}}_{\text{Count}=1}\ \bar{\epsilon}\ \underbrace{\epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}}_{\text{Count}=1}\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}\ \underbrace{\epsilon\ \epsilon\ \epsilon\ \bar{\epsilon}}_{\text{Count}=3}\ \underbrace{\bar{\epsilon}}_{\text{Count}=2}\ \underbrace{\bar{\epsilon}}_{\text{Count}=1^*}\ \}$$
$$\text{Using } T = \widehat{T} \uparrow c\ ,\ C_{min}(T,4) = 1 \text{ and } C_{max}(T,4) = 3$$

Fig. 2: Trace Mapping and Inter-Arrival Curve Calculation

Definition 4: (Inter-Arrival Curve) An inter-arrival calculation for an event type ϵ denoted by $C_f(T, \Delta) \rightarrow \mathbb{R}$ applies a specified function f to the occurrence count C of events of type ϵ within sliding windows of lengths $\Delta \in \mathbb{N}_{>0}$. The sliding windows always start with events of symbolic type ϵ.

We use Definition 4 to define two specific curves of interest which define the lower and upper bounds on the inter-arrival behaviour of an event type ϵ.

Definition 5: (Max/Min Inter-Arrival Curves) A maximum inter-arrival curve is defined with $f = max$, where max provides the maximum counts for occurrences of ϵ within the sliding windows described by Definition 4. Similarly, a minimum inter-arrival curve uses $f = min$, where min provides the corresponding minimum counts.

The following formulas show how to obtain a point on the maximum and minimum curves corresponding to a window duration Δ:

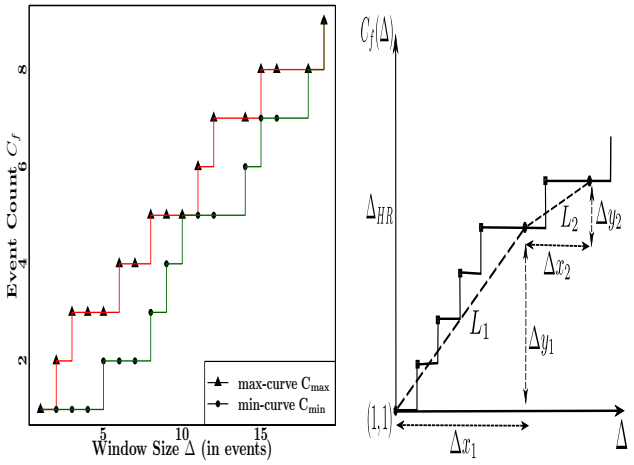
$$C_{\max}(T, \Delta) = max(C(T, \Delta)) \qquad (2)$$
$$C_{\min}(T, \Delta) = min(C(T, \Delta)) \qquad (3)$$

Having the sliding window positioned at ϵ optimizes the computation of $C_{\max}$ and $C_{\min}$, because the minimum and maximum count values will always be the same as the values obtained when sliding the window through all events ϵ and $\bar{\epsilon}$ of a given trace T.

Figure 2 shows an example calculation of $C_{\max}$ and $C_{\min}$ for the mapped trace T using a window duration Δ of 4 events, i.e, $C_f(T, \Delta = 4)\ for\ f \in \{max, min\}$.

Repeating this procedure on a range of Δ values yields the maximum and minimum inter-arrival curves. Figure 3a shows both $C_{\max}$ and $C_{\min}$ curves computed on the full range of $\Delta \in [1, |T|]$ for $\varepsilon = a$ using the trace $\widehat{T}$ of Example 1.

Note that the origin point of any $C_{\max}$ or $C_{\min}$ inter-arrival curve for an event type ε is always $(1, 1)$, which corresponds to the existence of at least one instance of that event within a given trace where the sliding window is positioned as mentioned in Definition 4.



(a) Min-Max Curves $\varepsilon = a$ (b) Steadiness Slope Metric

Fig. 3: Inter-Arrival Curves and Metrics

Definition 6: (Difference Inter-Arrival Curve C_{diff}) We define a third curve of interest, denoted as C_{diff}, whose values represent the difference between $C_{\max}$ and $C_{\min}$ at similar window durations Δ for a given trace T as follows:

$$C_{\text{diff}}(T, \Delta) = C_{\max}(T, \Delta) - C_{\min}(T, \Delta) \qquad (4)$$

The C_{diff} curve has some points of interest which allow for describing recurrent behaviours of real-time systems as we show in Section V. However, for any C_{diff} curve we have $C_{\text{diff}} = 0$ at both $\Delta = 0$ and $\Delta_{\max} = |T|$, since $C_{\max}(T, 1) = C_{\min}(T, 1) = 1$ and $C_{\max}(T, |T|) = C_{\min}(T, |T|) = C(T, |T|)$ respectively. Otherwise, the curve values belong to $\mathbb{N}_{\geq 0}$, because $C_{\max}(T, \Delta) \geq C_{\min}(T, \Delta)$ for all values of Δ. Formally,

$$C_{\text{diff}} = \begin{cases} 0, & \text{if } \Delta = 1 \\ \mathbb{N}_{\geq 0}, & \text{if } 1 < \Delta < |T| \\ 0, & \text{if } \Delta_{\max} = |T| \end{cases} \qquad (5)$$

In the next section, we introduce a set of single-curve and multiple-curves metrics to quantify and reason about the shape of inter-arrival curves. In Section V, we show how these metrics provide explanatory power for defining higher-level features to describe the behaviour of real-time systems.

IV. Metrics for Inter-Arrival Curves

The properties and metrics defined rely on the assumption that $C_f(T, \Delta + 1) \leq C_f(T, \Delta) + 1$. This is valid for f being max and min functions, because the minimum duration difference between two consecutive Δ is one event. Remember that the inter-arrival curve uses event arrivals on the x-axis and not time like standard arrival curves. This assumption leads to the following theorem.

Theorem 1: C_{max} and C_{min} are monotonically increasing with respect to window duration Δ.

Proof: Maximum and minimum inter-arrival curves use cumulative computations over increasing window durations Δ, the corresponding $C_{\text{max}}(T, \Delta)$ and $C_{\text{min}}(T, \Delta)$ values are either the same or increasing.

Using Theorem 1 and Definition 5, one can describe both C_{max} and C_{min} as a set of steady intervals (i.e., plateaus) and intervals of monotonically increasing counts C_f.

A. Single-Curve Metrics

The following metrics can be obtained from a single inter-arrival curve (i.e., a min- or max-curve).

Longest Interval of Strong Monotonic C_{max} Increase: For a max-curve C_{max}, there exist intervals of strong monotonic increase.

Theorem 2: The longest interval of strong monotonic increase, denoted by Δ_{HR}, for C_{max} exists at the beginning of the curve, starting at $(1, 1)$ and ending at the point having smallest Δ satisfying $C_{\text{max}}(T, \Delta) = C_{\text{max}}(T, \Delta + 1)$.

In other words, Δ_{HR} shows the longest train of events ϵ in mapped trace T. This can be shown on Figure 3a where $\Delta_{HR} = 3$ for event a as a result of having the stream $\{aaa\}$ in trace $\widehat{T}$ from Example 1. Note: Δ_{HR} can be represented on the y-axis as shown in Figure 3b as the maximum increase in C_{max} per increase in Δ is 1.

Longest Plateau of a Min-Curve C_{min}: We define a plateau or steady interval of an inter-arrival curve as an interval of successive Δ values having the same $C_f(T, \Delta)$.

Theorem 3: Longest plateau of a min-curve C_{min}, denoted as Δ_{FS}, exists at the beginning of the curve starting by point $(1, 1)$ and ending at the point having smallest Δ satisfying $C_{\text{min}}(T, \Delta) = 2$.

In other words, the Δ_{FS} value on the x-axis indicates the maximum window of separation between any two events ϵ within a mapped trace T. This can be shown on Figure 3a where $\Delta_{FS} = 5$ for event a as a result of having the streams $\{abbca\}, \{accca\}$.

Steadiness Slope of Inter-Arrival Curve: We define a metric $\bar{S}$ that describes how C_f increases with the increase of Δ by studying the steady intervals or plateaus of that curve. The steadiness slope $\bar{S}$ calculates the mean of the slopes of all virtual lines L_i that connects the last point of each two successive plateaus as shown in Figure 3b where i refers to the index of the plateau considered for calculation. The slope of a single virtual line L_i, denoted as S_{L_i}, having the start point at Δ_s and the end point at Δ_e where $\Delta_e > \Delta_s$ can be calculated using the following equation:

$$Slope\ S_{L_i} = \frac{\Delta y}{\Delta x} = \frac{C_f(T, \Delta_e) - C_f(T, \Delta_s)}{\Delta_e - \Delta_s} \tag{6}$$

To calculate the mean of slopes $\bar{S}$ of all virtual lines L_i defined over n plateaus, we use Equation 7:

$$\bar{S} = \frac{\sum\limits_{i} S_{L_i}}{n},\ where\ \bar{S} \in\ (0, 1). \tag{7}$$

The value of $\bar{S}$ for event $\varepsilon = a$ in Example 1 can be obtained by applying Equations 6 and 7 as follows: $\bar{S} = \frac{1}{5} * \{\frac{2}{4} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}\} = 0.42$.

This metric differs from the longest plateau and strongest monotonic increase as the steadiness slope $\bar{S}$ describes the occurrence behaviour of the system events over an interval by describing the density of the occurrence, i.e., count of symbolic event ϵ relative to the window duration Δ. For example, a higher value for mean slope $\bar{S}$ refers to a higher increase in the count $C_f(T, \Delta)$ as Δ increases. This indicates a higher event density within a trace as the length of the sliding window increases. The steadiness slope metric enables the comparison of inter-arrival curves calculated for different events within the same trace or calculated for the same event using different traces.

The Area under Inter-Arrival Curve: The next metric calculates the area under an inter-arrival curve C_f, denoted as Λ. Formally as follows:

$$\Lambda(C_f) = \sum_{\Delta=1}^{\Delta_{max}} C_f(T, \Delta),\ where\ \Delta_{max} \leq |T|. \tag{8}$$

To calculate this area, Equation 8 considers a virtual grid enclosed between the y-axis and the inter-arrival curve over the range $\Delta \in [1, \Delta_{max}]$, then counts the squares having an area equal to one unit on the x-axis by one unit on the y-axis. Our work does not use absolute values of Λ as a single-curve metric. Instead, the Λ metric allows us to relate multiple curves to each other by defining the following multiple-curve metrics whose calculation involves more than one curve.

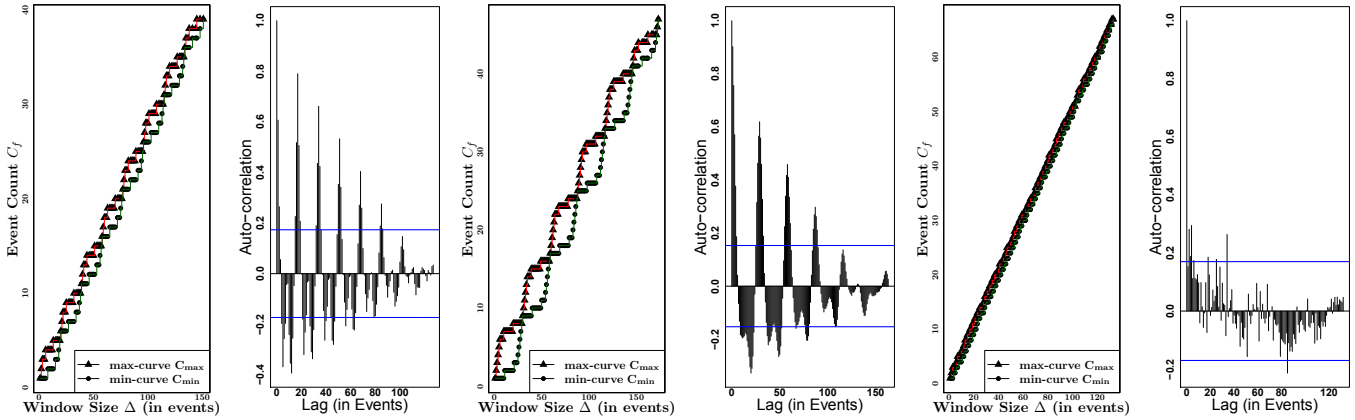
B. Multiple-Curves Metric

We introduce a multiple-curves metric to describe system behaviour using more than one inter-arrival curve. We then employ a variant of this metric to anomaly detection purposes in Section V.

The Proximity of Multiple Inter-Arrival Curves: We define a metric $Prox$ that relates different curves to each other by calculating the ratio of their area under curve values. A variant of the $Prox$ metric, denoted as $Prox_{min/max}$, calculates the proximity of C_{max} to C_{min} obtained from the same trace T as follows:

$$Prox_{min/max} = \frac{\Lambda(C_{\text{min}})}{\Lambda(C_{\text{max}})}\ ,\ Prox \in\ (0, 1] \tag{9}$$

$Prox_{min/max}$ describes the variation of inter-arrival behaviour of a given event in a trace by monitoring the variation between the minimum and maximum counts of events obtained



(a) $RPT1$ Pattern (b) $RPT2$ Pattern (c) No Pattern

Fig. 4: C_{max}, C_{min} Inter-Arrival Curves and Auto-Correlation of C_{diff}

using instances of sliding windows of different durations. As shown in Figure 2, $Prox_{min/max}$ quantifies the proximity of both curves to each other, where a higher ratio that tends to 1 indicates a minimum curve that is closer to the maximum curve. The more variation in the event occurrence in the trace, the lower the ratio will be as the min- and max-curves diverge away from each other. In Section V, we show how another variant of the $Prox$ metric calculated using same types of curves C_f, i.e. either C_{max} or C_{min} obtained from different traces, can be used to reason about the conformance of a system with the expected real-time behaviour.

V. Inter-Arrival Curves as High-Level Features to Reason about Real-Time Systems

In this section, we apply our work to real-time systems through using the curves as high-level features to describe the inter-arrival behaviour of generated events and to extract recurrent behaviour that these systems often exhibit. In addition, we provide a framework that uses multidimensional features to allow reasoning about the conformity of the behaviour expressed by an unknown trace to a system behaviour expressed by a previously collected set of known traces.

A. Extracting Recurrent Behaviour using $\mathbf{C}_{diff}$

To reason about the existence of recurrent behaviour of a real-time system, our approach aims to extract intervals of repeating modes of operation and characterize the inter-arrival behaviour of events within those intervals. For example, an application that can be described to have a recurrent single-mode of operation is a coffee machine having a 'brewing' mode and otherwise an 'idle' mode. We conjecture that finding points having $C_{\text{max}} \simeq C_{\text{min}}$, i.e., $C_{\text{diff}} \simeq 0$, separated by roughly the same distance, denoted as Δ_p, indicates that the event trace has a specific pattern where sliding windows of duration Δ_p yield roughly the same C_f counts. In other words, our conjecture is that a low variance σ of distances Δ separating points of $C_{\text{diff}} \simeq 0$ signifies such applications having recurrent single-mode of operation. The following set of Equations 10 shows how to calculate that variance σ of distances:

$$\begin{aligned}
&\zeta_\Delta = \{\Delta' | C_{\text{diff}}(\Delta', T) \simeq 0 \wedge \Delta' \in \mathbb{N}_{>0}\} \\
&\text{diff}(\zeta_\Delta) = \\
&\quad \{\delta | \delta = \Delta'_j - \Delta'_i, \Delta'_j = \inf\{\zeta_\Delta > \Delta'_i\} \text{for all} \Delta'_i \in \zeta_\Delta\} \\
&\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \text{ with } x = \text{diff}(\zeta_\Delta), \mu \text{ as mean of } x
\end{aligned} \tag{10}$$

Following this conjecture, we employ the *auto-correlation function (acf)* [19] to reason about the existence of recurrent patterns within a C_{diff} curve. Auto-correlation is widely used in signal processing to detect repeating patterns [20]. The auto-correlation function when applied to a curve shows the mutual relation of the curve with itself versus increasing time lag, where finding statistically significant auto-correlation values separated by approximately equal lag distances indicates a repeating pattern [21].

In our work, the auto-correlation function shows the correlation of the C_{diff} curve versus increasing window duration Δ to detect repeating patterns within the set ζ_Δ, where a C_{diff} curve having points of $C_{\text{diff}} \simeq 0$ at distances $\simeq \Delta_p$ of low variance σ is expected to yield an autocorrelation curve of statistically significant values separated by distances $\simeq \Delta_p$.

Example 2: Consider a trace $T1$ composed of a repeating pattern $RPT1$ of 19 events, $RPT1 = \{abbbbbbabbbbbbabbaa\}$ and trace $T2$ composed of a recurrent pattern $RPT2$ of 29 events which have an additional mode of operation, $RPT2 = \{aaaababbba\ abbbbbbabbbbbbabbaa\}$.

Figure 4a and Figure 4b show the corresponding C_{min}, C_{max} curves for traces $T1$, $T2$ and the auto-correlation of the corresponding C_{diff} curves. The auto-correlation result in Figure 4a shows a decaying sinusoidal-like curve, which indicates the existence of a repeating pattern within the C_{diff} curve with a cycle $\Delta_p \simeq 20$, i.e., the length of pattern $RPT1$.

Figure 4b shows a cycle of $\Delta_P \simeq 30$ as pattern $RPT2$ which has more events than the pattern $RPT1$. In practice, a change in the length of the recurrent pattern might indicate that a system repeats a different mode of operation or repeats a sequential order of several modes.

To show the result of applying our approach to traces whose events do not have recurrent behaviour, we randomly introduce events to disturb the patterns $RPT1$ and $RPT2$. As a result, the auto-correlation result shown in Figure 4c did not have statistically significant values. The significance is indicated by the horizontal confidence intervals around ± 0.2 shown in Figure 4c, where having autocorrelation values only within these bounds indicate that C_{diff} shows no correlation versus increasing lag [21]. Note that the choice of max window size Δ_{max} is crucial to detect the recurrence period Δ_p, e.g., Δ_{max} must be at least a multiple of Δ_p.

The results obtained using the synthetic trace of Example 2 shows how inter-arrival curves are potentially good features for describing recurrent behaviour within event traces. In Section VI, we show how this approach can be applied to event traces generated from deployed real-time systems.

B. Anomaly Detection Using $\mathbf{C}_{min}$ and $\mathbf{C}_{max}$

We present the framework in Figure 1 to exploit inter-arrival curves as high-level features suited for describing the behaviour of real-time systems. We conjecture that anomalies in an event trace would affect any of the metrics described in Section IV, i.e., change the steadiness slopes or the proximity of C_{min} and C_{max}. Therefore, we use these metrics to detect and quantify such anomalies for the purposes of anomaly detection.

Behavioural anomalies can cause changes to either C_{max}, C_{min}, or both of them. Anomalies that cause generation of more events of type ε within similar trace intervals will probably lead to higher values for C_{max} of that event type compared to a normal trace. An example of such an anomaly can be caused by running into a failure scenario for an embedded system. Another type of anomaly are ones that affect events of another type than ε leading to fewer counts in C_{min}. In other words, the density of events of type ε within sliding windows might decrease due to the increased occurrence of other event types within these windows. A practical example resulting in this anomaly can be due to the occurrence of unexpected event types in a trace (e.g., interrupts due to faults, error message events, or new user events). Lastly, anomalies might disrupt the distribution of multiple event types within the entire trace altogether and as a result, will affect both C_{min} and C_{max} curves for these event types.

We exploit inter-arrival curves in anomaly detection of real-time systems by adopting a generic semi-supervised classification framework as described in [2]. Figure 1 shows the building blocks for the framework. The framework builds a model using a set of event traces for a known expected behaviour, and then inspects a set of unknown traces for being either anomalous or not to that model.

Calculating Inter-Arrival Curves: During the training phase, the output from this building block is a set of pairs of min- and max-curve for each event type ε per each trace in the known traces set. For more robust results, we only calculate inter-arrival curves for event types that contribute more than a defined percentage of events within a trace. This threshold is a tunable framework parameter discussed in Section VI.

Building the Training Model: The second block shown in Figure 1 builds a model using the input pairs of C_{min} and C_{max} computed per each event type ε using traces of the training set. To achieve this, we aggregate the similar inter-arrival curves calculated using the same function f applied to the same event type ε to obtain a corresponding aggregated inter-arrival curve. The followed approach allows for incremental update of the model, where having new normal traces can add to the model by aggregating the curves obtained with the previously calculated model.

We experimented with various aggregation methods. The aggregation method that best represented the set of min- and max-curves was calculating a mean curve ($\bar{C}_f$) and confidence intervals for C_{min} and C_{max} independently. One main advantage of this technique is that it is robust against outliers, e.g., inter-arrival curves being unexpectedly different from the corresponding curves calculated using traces from the same traces set. We denote the mean curve as $\bar{C}_f$ and the confidence interval curves as C_f^- and C_f^+ where we compute a student's t-test confidence interval [22] using a window-by-window computation. The data sample for confidence interval calculation consists of all counts of events obtained from the training traces using the same function f corresponding to the same window duration Δ.

The output from the second block in the framework is a model composed of six inter-arrival curves per each event type ε as follows: the mean curves $\bar{C}_{min}$ and $\bar{C}_{max}$ along with the corresponding confidence interval curves as C_{min}^-, C_{min}^+, C_{max}^-, and C_{max}^+.

Trace Classification: Using inter-arrival curves allows pinpointing the specific events that caused the anomalous behaviour by specifying events ε whose C_{min} and C_{max} curves deviate from the built model. To achieve this, the last building block in the framework involves a binary classifier which evaluates whether the features obtained from the unknown trace conforms to the corresponding features obtained from the set of known traces. If the curve fails the test, the classifier then quantifies the deviation of that curve from the model for further classification stage.

Figure 5 shows an example from our experiments that we detail in Section VI. The figure visualizes the curves that represent the aggregated model for an event type ε along with the corresponding curves calculated using two unknown traces. The visualization in Figure 5a, opposed to Figure 5b, shows that curves conform to the obtained model, as the testing curves are close to the corresponding curves in the aggregated training model. Note that in our work, we consider an event behaviour to be anomalous to the model as long as one or more curves, i.e., C_{max} or C_{min}, are anomalous.

- Stage I: Detecting Curve Deviation.

As a result of Theorem 1, the technique in Example 3 shows how to represent the inter-arrival curves as a frequency distribution to describe the curve shape for statistical testing.

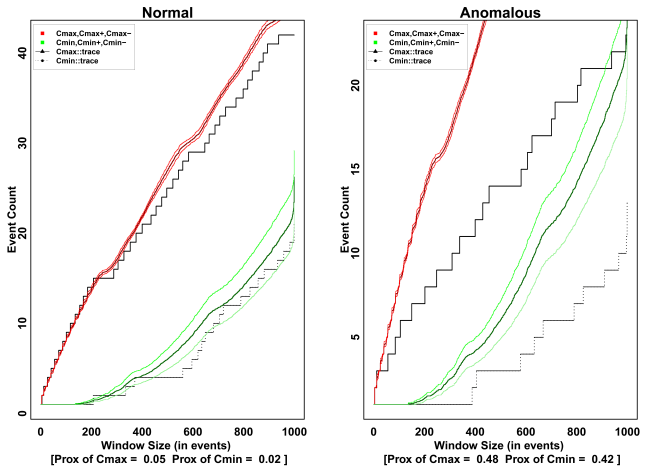
Example 3: Consider a max curve with values $C_{\max} = \{1, 2, 2, 3, 4, 4\}$ calculated using Δ_{max} of 6 events. We partition the curve to a set of intervals of unique $C_{\max} \in \{1, 2, 3, 4\}$ with corresponding Δ intervals $\in \{1, 2, 1, 2\}$.

To automate the curve similarity procedure, we apply the Wilcoxon-Mann-Whitney test, also known as the Mann-Whitney U test [23], which is a non-parametric statistical similarity test with a null hypothesis that two samples come from the same population. In our work, the test checks whether an inter-arrival curve C_f of an event type ε obtained from an unknown trace has the same distribution of a corresponding curve obtained from the aggregate model.

- Stage II: Measuring Deviation from Training Model.

To quantify a detected deviation in Stage I, we use the $Prox$ metric to quantify the proximity of an inter-arrival curve calculated using a test trace to a corresponding curve from the aggregate model. The corresponding curve from the model can either be the mean curve $\bar{C}_f$, the confidence interval curve C_f^-, or C_f^+.To choose that corresponding curve, we perform the $Prox$ metric calculation using the three curves and pick the nearest curve to C_f, i.e., picking the highest $Prox$ value to C_f. To finalize the analysis, a tunable threshold on the $Prox$ metric controls the classifier decision as will be discussed in Section VI.

The inter-arrival curve of an event type will be considered normal, if it passes the second stage of classification even if it failed the statistical test of first stage. Otherwise, the curve is declared to be anomalous by both stages. As the second stage requires more tuning parameters, we make this sufficient stage optional in our work. However, this stage can be essential when the model is over-fitting the training set of known traces.



(a) Normal Curves (b) Anomalous Curves

Fig. 5: `THREAD_THRUNNING` using two unknown traces

Figure 5a shows $Prox$ values of 0.06 and 0.02 for max- and min-curves respectively, hence the trace is declared normal using a $Prox$ threshold of 0.10. Curves of Figure 5b show $Prox$ values of 0.48 and 0.42 for max- and min-curves respectively, hence the trace is declared anomalous using the same threshold.

To obtain the final analysis result for the unknown trace, we use a straightforward voting technique that uses the classifier results obtained using a defined set of event types ε within that trace. In the next section, we discuss the significance of using different values for the tunable framework parameters on experiments results.

VI. Experimental Evaluation

In this section, we use event traces generated from deployed real-time systems of two industrial case studies to show the feasibility of the applications proposed in Section V.

UAV Case Study: We use kernel event traces generated from a unmanned aerial vehicle (UAV) running the real-time operating system QNX Neutrino 6.4. The UAV was developed at the University of Waterloo, received the Special Flight Operating Certificate (SFOC), and flew real mapping and payload-drop missions in Nova Scotia and Ontario. The trace snippet in Figure 6 shows the event attributes used in our experiments.

```
class   ,event           ,pname
PROCESS ,PROCCREATE_NAME ,proc/boot/procnto-instr
THREAD  ,THREADY         ,proc/boot/procnto-instr
THREAD  ,THRECEIVE       ,proc/boot/devc-seromap
.....   ,.....           ,........
```

Fig. 6: QNX Trace Snippet

The snippet is generated using the *tracelogger* and *traceprinter* utilities available in QNX Neutrino. In the experiments, an event type ε has a unique value that combines the values from the three attributes described in the snippet as an `event` from an event `class` that is generated by the kernel while running a specific process `pname`. We generated 254 UAV traces where each trace consists of a stream of roughly 10K events. These traces are generated from four anomalous execution scenarios. One scenario implements a task that interferes with system tasks by running a while-loop to consume CPU time, two scenarios implement a job executed every few seconds where the task is scheduled using two different scheduling algorithms (e.g., FIFO and sporadic scheduling), and the last scenario corresponds to a normal execution behaviour but does not conform to the training traces.

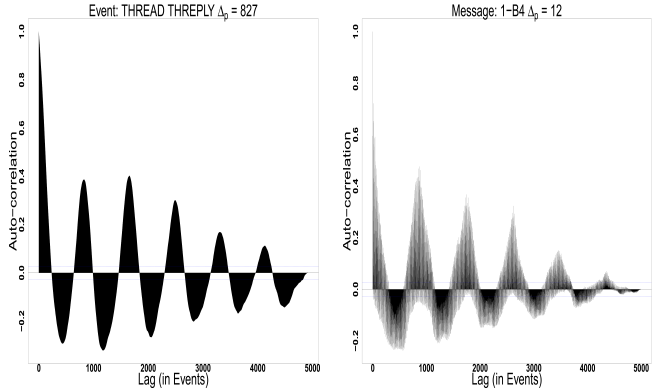
CAN Case Study: We captured 167 traces of CAN messages from a production vehicle during different driving scenarios. Each trace consists of roughly 10K CAN messages which represents 10 seconds of message capturing during the following driving scenarios: starting and stopping the engine, accelerating and decelerating between speeds of 0 and 40 km/h, and making lane changes to the left and to the right. On average, a trace has messages of 42 different CAN IDs, we combine the bus ID with the CAN message ID to obtain

an event type ε, e.g., a CAN message on bus 1 with ID $2C4$ has a type of $12C4$.

A. Extracting Recurrent Behaviour

We now provide the initial empirical evaluation for the conjecture presented in Section V-A using traces from the CAN case study, and then further evaluation using a subset of traces from the UAV case study.

Since CAN messages sent over a vehicle communication bus are often periodic, the aim of this case study is to validate our conjecture on traces known to exhibit recurrent behaviour. Using $\Delta_{max} = 5000$ messages to extract recurrent behaviour of CAN messages IDs occurring frequently, the experiments show that 10 out of 15 frequently occurring event types exhibit strong recurrent behaviour within the collected traces. Such recurrent behaviour is indicated by a decaying sinusoidal acf output of C_{diff} as discussed in the previous section.



(a) *UAV* Recurrent Pattern (b) *CAN* Recurrent Pattern

Fig. 7: Extracting Recurrent Behaviour Experiments

As the UAV case study uses QNX event traces generated by a real-time system, we expect the traces to show signs of recurrent behaviour. We use a subset of roughly 60 traces which consist of a stream of 10K events each to represent the three different execution scenarios: *normal*, *full-while*, and *fifo-ls*. The *normal* execution scenario shows that 19 out of 22 event types have recurrent behaviour in the generated traces. The *fifo-ls* scenario shows similar numbers of event types having recurrent behaviour, however such behaviour occurred with different intervals Δ_p. The *full-while* scenario shows on average of 7 event types to have recurrent behaviour meaning that several events lost the recurrent behaviour due to the highly utilized CPU behaviour affecting the behavioural patterns of the system.

Figure 7 provides a sample of the result of autocorrelation function applied to the C_{diff} curves obtained from both case studies where Figure 7a corresponds to the curve of a QNX event type `THREAD_THREPLY` while Figure 7b corresponds to the curve of a CAN message `B4` from the vehicle CAN bus ID `1`. Both curves shows the recurrent behaviour within the generated trace that can be characterized using the recurrence windows $\Delta_p = 827$ events and $\Delta_p = 12$ messages respectively.

The number of events that showed recurrent behaviour in both case studies highlights the applicability of using inter-arrival curves for the purpose of characterizing the recurrent behaviour of real-time systems using their event traces. The promising technique can be applied similarly to generic streams of recorded events as function calls, system tasks, etc.

B. Anomaly Detection Analysis

To evaluate the anomaly detection framework discussed in Section V-B, we use a modelling set of roughly 100 known traces of an expected behaviour from the UAV and a set of roughly 40 unknown traces divided equally between normal and anomalous traces for each experiment. Note that the normal traces used in the testing phase were different from the ones used in the training phase.

The experiments show the results of classifying the unknown traces as normal or anomalous where we evaluate the classification performance using the receiver operating characteristic (ROC) curve [24]. An ROC curve plots the True Positive Rate (`TPR`) values on the y-axis versus the False Positive Rate (`FPR`) values on the x-axis which are obtained by varying the binary classifier threshold values. The `TPR` and `FPR` are calculated using the following equations:

$$\text{TPR} = \frac{\text{True Positives Detected by Classifer}}{\text{Actual Positives in Testing Set}} \tag{11}$$

$$\text{FPR} = \frac{\text{False Positives Flagged by Classifier}}{\text{Actual Negatives in Testing Set}} \tag{12}$$

An ROC with a higher area under the curve indicates a better classifier where the point on the top-left corner $(0\%, 100\%)$ indicates a perfect classification result. The line $\texttt{TPR} = \texttt{FPR}$ corresponds to a classifier which is as good as a random classifier.

Tuning Framework Parameters. The implemented framework has some tuning parameters that can be used to achieve better classification results. Such tuning operation also ensures that the model does not overfit the set of normal traces. We explain how to use the prior knowledge to tune the following framework parameters along with experiments that show their effect on the classification results: voting threshold on anomalous curves count denoted as ν_a, the significance percentage of an event type ε within a trace denoted as $S\%$, and the maximum events window duration denoted as Δ_{max}.

The voting threshold ν_a specifies the minimum number of anomalous inter-arrival curves computed using a trace so that the binary classifier declares the trace as anomalous. Tuning the threshold controls how strict the classification is. For example, a strict classifier with $\nu_a = 1$ will consider a trace to be anomalous if only a single inter-arrival curve turned out to be anomalous. In practice, this setting would lead to high false alarms as some event types within a normal trace might still yield inconsistent inter-arrival curves, so a low value for ν_a would raise more false alarms and lead to a higher false positive rate.

The significance percentage parameter, $S_{\%}$, specifies the minimum proportion of events of the same type in a trace before that type is considered significant. Choosing a value for $S_{\%}$ requires preliminary analysis of training traces to study the occurrence frequency of the different event types, that is to choose $S_{\%}$ values that yield an appropriate count of event types contributing to the classification procedure in the framework. We want a value of $S_{\%}$ high enough to capture just enough significant event types to accurately distinguish traces with fundamental differences using their inter-arrival curves. However, using more event types ε in the training model might overfit the training data and hence degrade the classifier performance.

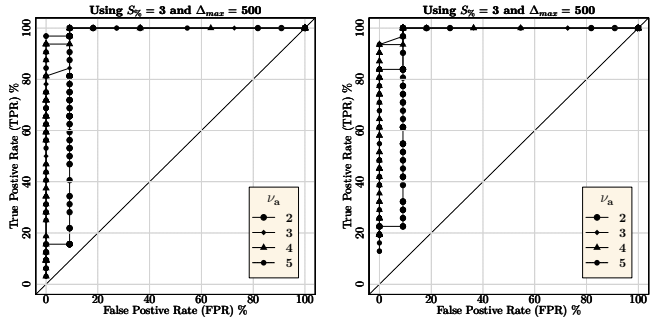
The last parameter, Δ_{max}, specifies the maximum duration for a sliding window of events. Increasing Δ_{max} might allow detection of anomalies whose disruption effect might go undetected using shorter window sizes. The deviation in Figure 5b between the result obtained using a testing trace with respect to the training model is significant after $\Delta \gtrsim 200$ events, so using a $\Delta_{max} = 200$ events will classify the curve of that event type to be normal however using a $\Delta_{max} = 1000$ events will correctly classify it as anomalous. It is important to note that such detection comes at higher computation cost. That is why we consider it to be the last resort while tuning parameters for better classification results. However, increasing Δ_{max} might have an over-fitting effect on the training model which raises the FPR during testing. For our case studies, experimental results represented by ROC curves show that using $\Delta_{max} = 500$ events achieves classification results better than using $\Delta_{max} = 1000$ events.

Table I shows the best classification results obtained for the different anomalous scenarios after tuning the previously mentioned parameters. Since results of scenarios *fifo-ls* and *different-normal* achieved only less than perfect classification results, we show the best classification results indicated by the ROC curves in Figure 8. The full dataset of results generated by our algorithms are available publicly [25]. The values in the figure corresponds to the tuned Δ_{max} and $S_{\%}$ while varying ν_a over a range of $[2, 5]$. Using other values for the tunable parameters yielded poor classification results, i.e., ROC having less area under the curve.

Scenario	TPR	FPR	Framework Parameters Values
full-while	100%	0%	$\Delta_{max} = 500$, $S_{\%} = 3$, $\nu_a = 3$
different-normal	100%	0%	$\Delta_{max} = 500$, $S_{\%} = 1$, $\nu_a = 4$
sporadic-ls	97%	0%	$\Delta_{max} = 500$, $S_{\%} = 3$, $\nu_a = 5$
fifo-ls	94%	0%	$\Delta_{max} = 500$, $S_{\%} = 3$, $\nu_a = 3$

TABLE I: Classification Results

The results in Table I and Figure 8 show the feasibility of the proposed framework and its applicability to real-time systems deployed for industrial purposes. The lack of publicly available datasets hinders the process of comparing the different feature extraction techniques to each other [26]. To address that, we are working on making our framework implementation publicly available and compatible with generic event streams from different sources. However, one main difficulty specific to cyber-physical systems is that these systems all have their own format for event traces which is not the case with research techniques developed for more common traces, e.g., Java execution traces.



(a) *sporadic* Scenario (b) *fifo-ls* Scenario

Fig. 8: Effect of Tuning Count Threshold ν_a on Classification

Complexity and Parallelism. Sliding window techniques are computationally exhaustive with complexity O($|T|^2$) where $|T|$ is the number of events within a trace T [2]. However, that complexity can be reduced using pruning techniques that do not necessarily perform all possible computations [27]. Similarly in our framework, we reduce computations needed to calculate inter-arrival curves for a reduced set of significant event types controlled by $S_{\%}$.

We implemented our technique using the R programming language to calculate the C_{min} and C_{max} curves and the auto-correlation of the C_{diff} curve. We obtain the ROC curve values of FPR, TPR using *ROCR* R library [28]. Implementing inter-arrival curves using a sliding windows technique is well-suited for parallelism because the computations performed on different windows are completely independent. For the parallel computations, we used the *doMC* R library [29] to benefit from our experimentation machine having 32 CPU cores spread on four processors with an 8-core Intel Xeon E5-2630 V3 2.4GHz each. Although the computation times achieved using that approach were reasonable, we implemented inter-arrival curves via a more scalable Cartesian product approach that is more computationally efficient however, consumes more memory. The computation of the curves and the analysis for classification takes $\sim$ 2 to 3 seconds using a trace of 10K events and $\Delta_{max} = 1000$ events.

VII. Discussion

The experimental results in Section VI show the technical feasibility and viability of using inter-arrival curves for the proposed applications. The ROC curves show how the introduced framework can detect anomalies in the behaviour of real-time systems.The mentioned computation time demonstrates the practicality of using discrete event traces, which despite being less informative still capable of modelling the system behaviour for anomaly detection purposes.

The autocorrelation function applied to C_{diff} verifies the existence of recurrent behaviour of interval Δ_p within their event traces. The approach requires further work to extract characteristic sub-traces for online anomaly detection purposes. Further research work can adopt different curve representation and model aggregation functions in addition to more curve similarity techniques for better classification results.

We highlight that inter-arrival curves are not capable of detecting anomalies that affect events timing because the behavioural model does not consider event timestamps. Also, inter-arrival curves will not be able to detect anomalies that have a minor disruption to the event trace which is not enough to deviate the system from its well-specified behaviour.

VIII. Conclusion

Following the success of well-known arrival curves in performance analysis of real-time systems, we introduced inter-arrival curves as a special form of arrival curves to be used as an analytical modelling technique using event traces along with quantifiable metrics for reasoning. We integrated inter-arrival curves into an anomaly detection framework and provided initial evidence on the suitability of inter-arrival curves for detecting recurrent system behaviour using event traces. The experiments validate our work using industrial case studies from two different application domains.

References

[1] A. Pimentel, L. Hertzbetger, P. Lieverse, P. van der Wolf, and E. Deprettere, "Exploring embedded-systems architectures with artemis," *Computer*, vol. 34, no. 11, pp. 57–63, Nov 2001.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 5, pp. 823–839, 2012.

[3] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[4] K. Altisen, Y. Liu, and M. Moy, "Performance evaluation of components using a granularity-based interface between real-time calculus and timed automata," *CoRR*, vol. abs/1004.2637, 2010. [Online]. Available: http://arxiv.org/abs/1004.2637

[5] L. T. X. Phan, "Towards a safe compositional real-time scheduling theory for cyber-physical systems," *AVICPS 2013*, p. 21, 2013.

[6] S. Perathoner, T. Rein, L. Thiele, K. Lampka, and J. Rox, "Modeling structured event streams in system level performance analysis," in *ACM Sigplan Notices*, vol. 45. ACM, 2010, pp. 37–46.

[7] A. Bouillard, A. Junier, and B. Ronot, "Hidden anomaly detection in telecommunication networks," in *Proceedings of the 8th International Conference on Network and Service Management*. International Federation for Information Processing, 2012, pp. 82–90.

[8] K. Huang, G. Chen, C. Buckl, and A. Knoll, "Conforming the runtime inputs for hard real-time embedded systems," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 430–436.

[9] B. Hu, K. Huang, G. Chen, and A. Knoll, "Evaluation of runtime monitoring methods for real-time event streams," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 582–587.

[10] V. Chandola, V. Mithal, and V. Kumar, "Comparative evaluation of anomaly detection techniques for sequence data," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 743–748.

[11] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs." in *DATE*, vol. 3. Citeseer, 2003, p. 10190.

[12] K. Banerjee and P. Dasgupta, "Acceptance and random generation of event sequences under real time calculus constraints," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 254.

[13] Y. Qiao, X. Xin, Y. Bin, and S. Ge, "Anomaly intrusion detection method based on hmm," *Electronics Letters*, vol. 38, no. 13, p. 1, 2002.

[14] X. Li, J. Han, S. Kim, and H. Gonzalez, "Roam: Rule-and motif-based anomaly detection in massive moving object data sets." in *SDM*, vol. 7. SIAM, 2007, pp. 273–284.

[15] S. Salvador, P. Chan, and J. Brodie, "Learning states and rules for time series anomaly detection." in *FLAIRS Conference*, 2004, pp. 306–311.

[16] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data mining and knowledge discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[17] D. Lo, S.-C. Khoo, and C. Liu, "Efficient mining of recurrent rules from a sequence database," in *Database Systems for Advanced Applications*. Springer, 2008, pp. 67–83.

[18] M.-Y. Su, "Discovery and prevention of attack episodes by frequent episodes mining and finite state machines," *Journal of Network and Computer Applications*, vol. 33, no. 2, pp. 156–167, 2010.

[19] G. E. Box and G. M. Jenkins, *Time series analysis: forecasting and control, revised ed*. Holden-Day, 1976.

[20] S. Laxman and P. S. Sastry, "A survey of temporal data mining," *Sadhana*, vol. 31, no. 2, pp. 173–198, 2006.

[21] M. Natrella, "Nist/sematech e-handbook of statistical methods," 2010.

[22] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[23] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[24] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[25] M. Salem, M. Crowley, and S. Fischmeister, *Dataset for Anomaly Detection Using Inter-Arrival Curves for Real-time Systems*. [Online]. Available: http://dx.doi.org/10.5281/zenodo.51472

[26] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[27] A. Ghoting, S. Parthasarathy, and M. E. Otey, "Fast mining of distance-based outliers in high-dimensional datasets," *Data Mining and Knowledge Discovery*, vol. 16, no. 3, pp. 349–364, 2008.

[28] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer, "Rocr: visualizing classifier performance in r," *Bioinformatics*, vol. 21, no. 20, p. 7881, 2005. [Online]. Available: http://rocr.bioinf.mpi-sb.mpg.de

[29] R. Analytics, *doMC: Foreach parallel adaptor for the multicore package*. [Online]. Available: http://CRAN.R-project.org/package=doMC