Mining Event Traces from Real-time Systems for Anomaly Detection

by

Mahmoud Salem

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Nathan Fisher Associate Professor, Wayne State University College of Engineering
Supervisor(s):	Sebastian Fischmeister Associate Professor, University of Waterloo Department of Electrical and Computer Engineering
	Mark Crowley Assistant Professor, University of Waterloo Department of Electrical and Computer Engineering
Internal Member:	Lin Tan Associate Professor, University of Waterloo Department of Electrical and Computer Engineering
Internal Member:	Rodolfo Pellizzoni Associate Professor, University of Waterloo Department of Electrical and Computer Engineering
External Member:	Matthias Schonlau Professor, University of Waterloo Department of Statistics and Actuarial Science

Author's Declaraction

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contribution

In what follows is a list of publications which I have co-authored and used their content and ideas in this dissertation. The use of the content from the listed publications in this dissertation has been approved by all co-authors.

- Mahmoud Salem, Mark Crowley, and Sebastian Fischmeister. Anomaly Detection using Inter-arrival Curves for Real-time Systems. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016
  - Defined and Implemented arrival curves for anomaly detection;
  - Designed and performed the evaluation methods;
  - Wrote the paper.
- Gonzalo Carvajal, Mahmoud Salem, Nirmal Benann, and Sebastian Fischmeister. Enabling Rapid Construction of Arrival Curves from Execution Traces. *IEEE Design & Test*, 2017
  - Performed the performance evaluation of the algorithm;
  - Contributed in writing the related work and evaluation sections.
- Mohammad Mehdi Zeinali Zadeh, Mahmoud Salem, Neeraj Kumar, Greta Cutulenco, and Sebastian Fischmeister. SiPTA: Signal Processing for Trace-based Anomaly Detection. In *Proceedings of the 14th International Conference on Embedded Software (EMSOFT)*, 2014
  - Prepared an anomaly detection survey which contributes to the thesis related work;
  - Working on SiPTA helped formulate the TRACMIN framework in this thesis;
  - Wrote portions of the paper.

Abstract

Real-time systems are a significant class of applications, poised to grow even further as autonomous vehicles and the Internet of Things (IoT) become a reality. The computation and communication tasks of the underlying embedded systems must comply with strict timing and safety requirements as undetected defects in these systems may lead to catastrophic failures. The runtime behavior of these systems is prone to uncertainties arising from dynamic workloads and extra-functional conditions that affect both the software and hardware over the course of their deployment, e.g., unscheduled firmware updates, communication channel saturation, power-saving mode switches, or external malicious attacks. The operation in such unpredictable environments prevents the detection of anomalous behavior using traditional formal modeling and analysis techniques as they generally consider worst-case analysis and tend to be overly conservative.

To overcome these limitations, and primarily motivated by the increasing availability of generated traces from real-time embedded systems, this thesis presents TRACMIN - Trace Mining using Arrival Curves - which is an anomaly detection approach that empirically constructs arrival curves from event traces to capture the recurrent behavior and intrinsic features of a given real-time system. The thesis uses TRACMIN to fill the gap between formal analysis techniques of real-time systems and trace mining approaches that lack expressive, human-readable, and scalable methods. The thesis presents definitions, metrics, and tools to employ statistical learning techniques to cluster and classify traces generated from different modes of normal operation versus anomalous traces. Experimenting with multiple datasets from deployed real-time embedded systems facing performance degradation and hardware misconfiguration anomalies demonstrates the feasibility and viability of our approaches on timestamped event traces generated from an industrial real-time operating system.

Acknowledging the high computation expense for constructing empirical arrival curves, the thesis provides a rapid algorithm to achieve desirable scalability on lengthy traces paving the way for adoption in research and industry. Finally, the thesis presents a robustness analysis for the arrival curves models by employing theories of demand-bound functions from the scheduling domain. The analysis provides bounds on how much disruption a real-time system modeled using our approach can tolerate before being declared anomalous, which is crucial for specification and certification purposes. In conclusion, TRACMIN combines empirical and theoretical methods to provide a concrete anomaly detection framework that uses robust models of arrival curves scalably constructed from event traces to detect anomalies that affect the recurrent behavior of a real-time system.

Acknowledgements

I am sincerely thankful to my supervisor, Professor Sebastian Fischmeister, for the opportunities and continuous guidance. His way of supervision enriched my skills through performing research, presenting demos, teaching, and collaborating with the industry. His support during my toughest times will never be forgotten.

Thanks to my co-supervisor, Professor Mark Crowley, for his valuable guidance that expanded my research scope to machine learning. I would like to thank Professor Gonzalo Carvajal for his collaboration that evolved the research ideas and his continuous constructive reviews. I would like to thank my committee members: Professor Lin Tan, Professor Matthias Schonlau, Professor Nathan Fisher, and Professor Rodolfo Pellizzoni for their feedback and for taking the time to serve on the examination committee.

I would like to thank my friends in the Real-time Embedded Software Group whom I met throughout this unforgettable time especially, Hany Kashif, Ramy Medhat, and the "Embedded Soccer" team. Thanks to my friends Ahmed Raslan, AbdelMaguid Tawakol, and Amany El-Gouhary for making the working days enjoyable.

There are no words that can describe my gratitude to my parents, Omnia and Ashraf. My late mother always pushed me to pursue my dreams with unparalleled support and love; her legacy continues to surround me in every achievement. Nothing motivated me to work towards my degree more than the inspiring career of my father and his valuable advice.

I would like to thank my lovely wife, Mahitab, for her constant support and patience. Thank you for encouraging me throughout the entire journey. Finally, I would like to thank my son, Karim, for always bringing joy to my days.

Dedication

To my parents, Omnia and Ashraf.
To my sister, Nihal.
To my wife, Mahitab.
To my son, Karim.

Table of Contents

List of Tables

List of Figures

List of Symbols

$\boldsymbol{C_{\text{max}}}$ Maximum Arrival Curve: A curve that captures the maximum of event counts observed in all windows of a given size sliding throughout a trace.

$\boldsymbol{C_{\text{min}}}$ Minimum Arrival Curve: A curve that captures the minimum of event counts observed in all windows of a given size sliding throughout a trace.

$\boldsymbol{C_{\text{diff}}}$ Difference Arrival Curve: A curve that represent the difference between C_{max} and C_{min}.

$\boldsymbol{\bar{S}}$ The mean of slopes of virtual lines connecting the plateaus in an arrival curve.

$\boldsymbol{\Lambda_f(T)}$ The area under an arrival curve described by a function f for a trace T.

$\mathbb{P}$ The proximity of multiple arrival curves as the ratio of the areas under both curves.

$\boldsymbol{C^-_{\text{min}}, C^+_{\text{min}}, C^-_{\text{max}}, C^+_{\text{max}}}$. Confidence Interval Curves for C_{max} and C_{min}.

$\boldsymbol{\bar{C}_f}$ The mean of a given set of arrival curves of a given function f.

`TPR`, `FPR` True positive rate and False positive rate for a given classifier.

$\boldsymbol{\nu_a}$ Voting threshold on the number of anomalous events.

$\boldsymbol{S_{\%}}$ The percentage of which an event is considered significant in a trace.

$\boldsymbol{\Delta_{max}}$ Maximum window size of the sliding window of computation.

$\boldsymbol{\Delta_p}$ Recurrence period captured in an arrival curve C_{diff}.

$\boldsymbol{dbf(t)}$ The demand-bound function for a given task.

$\boldsymbol{\sigma}$ Variation in the demand of a given task due to some system alteration.

$\boldsymbol{\alpha}$ Decrease in period of a given task due to some system alteration.

$\boldsymbol{\beta}$ Increase in execution time of a given task due to some system alteration.

$\boldsymbol{\alpha_u}, \boldsymbol{\alpha_l}$ Upper and Lower bounds on the decrease in period of a given task due to some system alteration.

$\boldsymbol{\beta_u}, \boldsymbol{\beta_l}$ Upper and Lower bounds on increase in execution time of a given task due to some system alteration.

$\boldsymbol{T_K}$ A task whose demand corresponds to an approximated demand-bound function.

$\boldsymbol{T'_K}$ An altered task whose demand corresponds to an approximated demand-bound function with some variation.

$\boldsymbol{T}$ A trace: corresponds to a sequence of timestamp and attributes corresponding to a given system.

$\boldsymbol{TS}$ A trace sequence: corresponds to a sequence of timestamps only.

$\boldsymbol{\phi^l}, \boldsymbol{\phi^u}$ Actual upper and lower arrival curves of a given trace sequence.

$\hat{\boldsymbol{\phi}}^{\boldsymbol{l}}, \hat{\boldsymbol{\phi}}^{\boldsymbol{u}}$ Quantized upper and lower arrival curves of a given trace sequence.

Chapter 1

Introduction

The introduction of the Internet-of-Things (IoT) in industrial settings is accelerating innovation towards complex Cyber-Physical Systems (CPS) such as advanced driving assistance systems (ADAS), connected and unmanned vehicles, smart process control, among others [5, 29]. Many of these applications rely on distributed embedded systems that perform real-time operations, and thus they must be carefully designed and validated to comply with strict timing and safety specifications [54, 77].

Even for well-tested deployed systems, undetected errors may lead to catastrophic failures similar to the failures of Therac-25 device [48] and Ariane 5 flight [51]. Making it more challenging, modern real-time systems are becoming increasingly complex, and their runtime behavior is prone to uncertainties arising from dynamic workloads and changes in their underlying software and hardware over the course of their deployment. For example, a platform executing a real-time application may suffer a degradation in processor performance when switching to a low-power mode, or it may sporadically increase its processor demand when accommodating for running some failure scenario. Therefore, embedded systems typically used in safety-critical 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 [1] for automotive functional safety and DO-178C [2] for airborne systems).

The ever-increasing complexity and inter-connectivity of modern embedded systems are setting major challenges for the verification of their functional and non-functional properties (e.g., timing constraints). For decades, designers have used mathematical models of systems and formal methods to reason about real-time performance metrics at early design phases. Traditional formal approaches work well for simple closed systems whose worst-case behavior can be mapped to generic mathematical representations, but they are inadequate for characterizing complex

interconnected systems whose behavior depends on unpredictable interactions between highly-heterogeneous components. The limitations of formal methods have triggered a renewed interest in techniques based on runtime empirical analysis, which rely on the collection and processing of actual execution traces to capture non-trivial recurrent behavioral patterns that cannot be predicted at design time. As it becomes clearer that verification of modern real-time systems cannot purely rely on broad generic abstractions and theoretical analysis, it is more important than ever to promote evidence-based discussions and drive new research towards novel data-driven techniques for analyzing more representative system-specific data.

Traditional formal methods for exhaustive analysis are relatively mature and have become a standard practice in the industry; however, as they only target worst-case analysis, they tend to be overly conservative and do not provide information about the robustness of the system to specific variations in the parameters of composing task sets. As modern real-time systems become more heterogeneous and interconnected, it is important to understand in more detail how variations in individual system parameters affect the demand and timing behavior of the system, in order to guarantee correctness while maximizing utilization and performance. An extensive body of research discussed formal analytical approaches to model and inspect real-time systems before deployment. However, these methods tend to be pessimistic and can not capture the evolving design changes that a system would encounter.

To overcome this problem, trace mining approaches offer a solution as it provides a non-invasive method that analyzes the traces readily generated during the repeating executions of the systems. In such systems, trace-based anomaly detection can act as a monitoring mechanism and invoke modules responsible for prevention and recovery from failures. Trace mining as a subfield of data mining has been of research interest because the problems facing that techniques have its solutions using statistical and computational approaches [26, 89]. Massive datasets generated continuously motivated the use of data mining techniques to extract useful information for a wide range of applications. In essence, trace-based anomaly detection aims at detecting execution patterns that do not conform to the normal functioning of the system. Detection of anomalous patterns can indicate software bugs or malfunctions which is convenient, as such an analysis treats the system under scrutiny as a black box and does not require knowledge of its internals. The approach 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. Detection of anomalies can be done both, online (during runtime) 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.

Bespoke real-time embedded devices have intrinsic properties that make generic anomaly detection approaches ill-suited for this domain [92]. On the contrary, fine-grained models capture

the specific behavior of a given cyber-physical system by using domain-specific knowledge to improve classification and root-cause analysis. Most trace mining approaches are not specific to real-time systems as they do not capture the recurrent nature of the bespoke real-time systems. Our work aims to find tools or develop methodologies that are best suited for the traces of real-time systems. Also, the features extracted should allow for human interpretation to make it easier to comprehend the system behavior, 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 behavior.

Several challenges face the trace analysis approach. One main challenge of trace-based anomaly detection is how to identify an incorrect behavior without raising too many false alarms. Anomaly detection tends to reduce false positives. However a false negative is a serious problem because it would imply that a fault might go without inspection. Related work in a comprehensive survey [18] 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. Although the empirical evaluation of the model is the common technique to evaluate the accuracy of the model, statistical measures that provide confidence in the evaluation results are crucial for the industrial adoption of anomaly detection, however, there is a gap in the research work on this measures. Also, a common challenge for trace analysis approaches is the high computational expense of the analysis of large amounts of data within an event trace [19].

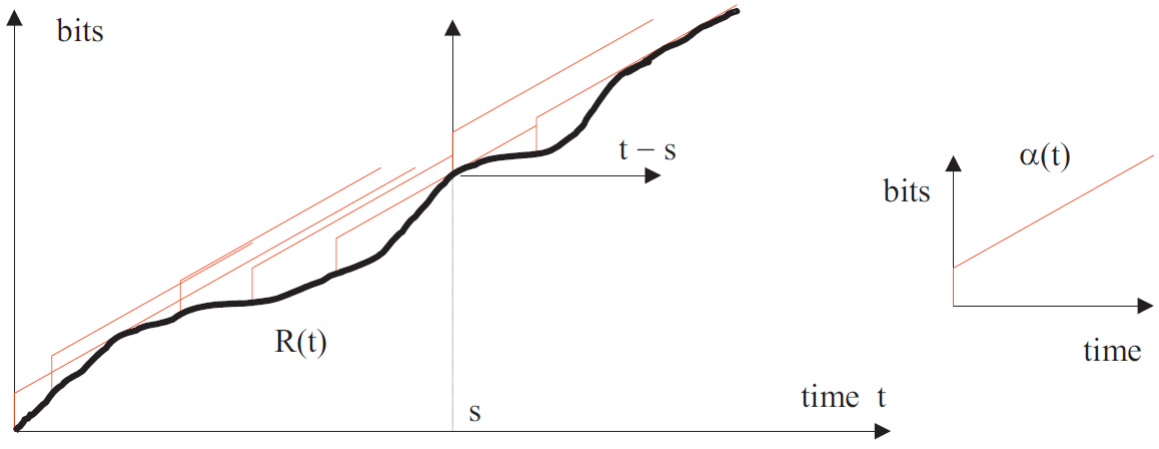


Figure 1.1: Example for an arrival curve [45]

Our conjecture that arrival curves are suited to mine recurrent behavior from generated event streams from real-time systems. Arrival curves have been widely used in service-level agreements to characterize the arrival of data streams in the networks domain. Figure 1.1 shows an example for an arrival curve as defined in [45] to constrain the arrival of bits for a given network setting. The right-hand side figure shows a bound that should be continuously enforced to the cumulative function $R(t)$. In the context of event streams, arrival curves can be described as functions of the *interval time domain* that provide upper and lower limits to the number of events that can occur in a system within any given time interval of length Δt [75].

In our work, we consider arrival curves as high-level summarization for the recurrent behavior of a real-time system computed using event traces. Through statistical learning approaches, i.e., classification and clustering, to classify normal and anomalous traces after separating them based on the underlying modes of operation. We validate our approach by experimenting with multiple datasets from real-time systems having recurrent behavior.

Thesis Statement

The thesis aims to quantitatively evaluate whether arrival curves constructed from event traces provide good features for trace mining to effectively and efficiently characterize the recurrent behavior of real-time systems for anomaly detection purposes.

Thesis Contribution

We summarize the thesis contributions in the main following points:

- The thesis introduces empirical arrival curves to fill the gap between formal analysis of real-time systems and trace mining approaches that do not capture the intrinsic features within event traces generated by those systems.
- Using the definitions and metrics of empirical arrival curves, we introduce TRACMIN framework that employs statistical learning techniques to the trace-based curves.
- We assess whether empirical arrival curves employed in TRACMIN can achieve accurate trace classification for the different modes of operation for a given real-time system and can perform anomaly detection within the recurrent behavior captured by event traces. Additionally, we show how TRACMIN can be extended to on-the-fly anomaly detection purposes.

- To ensure the deployment of our methods, a robustness analysis for the offline arrival curves models which employs theories of demand-bound functions from the scheduling domain. The analysis provides bounds on how much disruption real-time systems modeled using empirical arrival curves can tolerate before being declared anomalous. Examining the bounds on permissible system variations would allow designers to asses the robustness of a real-time system model to dynamic operational conditions.
- We overcome the expensive computation of empirical arrival curves by providing a rapid algorithm for their construction and presenting the necessary analysis to demonstrate the scalability of the algorithm on lengthy traces paving the way for the work in this thesis to be fully adopted in research and industry.

Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides an overview of the related work on trace mining approaches for anomaly detection, in addition to, the background of arrival curves and how it relates to real-time systems. Chapter 3 defines empirical arrival curves constructed from timestamped event data along with metrics that characterize the curves. We also discuss the semantics of those curves and metrics in reality. Chapter 4 introduces TRACMIN which is the anomaly detection framework we use to exploit the definitions of empirical arrival curves. The chapter provides evaluation methods for TRACMIN using an industrial case-study. Chapter 4 extends the application of the framework to clustering traces originating from different modes of operation, to the deployment in a streaming framework for on-the-fly anomaly detection, and mining recurrent patterns from real-time systems traces. Chapter 6 provides a novel method for robustness assessment to the models obtained by TRACMIN providing a quantitative model evaluation and bounds on accuracy. Chapter 7 discusses the traditional algorithms used to construct empirical arrival curves and presents a scalable implementation for a rapid algorithm that enables the wider adoption of the framework into research and industry. Finally, Chapter 8 concludes the thesis by discussing the contributions, limitations, and possible extensions of TRACMIN.

Chapter 2

Related Work

First, it is important to differentiate between *anomaly detection* which aims to detect a change in behavior for a given system, and *intrusion detection* which is a common approach for finding an instance or more of unexpected events within an execution trace. The main objective of the framework and approaches presented in this thesis is to accurately and efficiently detect anomalous behavior in real-time systems by processing event data generated during execution.

2.1 Anomaly Detection using Event Data

To be able to well-define the scope of the work presented in this thesis, we summarize the main points of a widely used comprehensive review on anomaly detection [19].

1- Nature of Input Data: The data used in an anomaly detection procedure can be binary, categorical, or continuous. These data instances can have one of many relations, such as being sequence data, spatial data, spatio-temporal, etc. The thesis is mainly concerned with timestamped event data from real-time operating systems which can be classified as categorical sequence data. We note that sequential data streams and time series are both sequence data, i.e., a sequence of ordered events. However, a time series can relate different observations to time and observe the trend and seasonality. According to this definition, we cannot consider the event traces generated from real-time systems as time series where the time cannot present an intrinsic feature as defined by Box and Jenkins [10].

2- Types of Anomaly: We already mentioned that our work is concerned with anomalies; the survey mentions three main types of anomalies: point, contextual, and collective anomalies. Point anomaly is an individual data instance that does not conform to the rest of the data. Contextual

anomalies are defined over a specific context, i.e., fraud in credit card activity. The collective anomalies are the types of interest in this thesis, which is defined as a collection of data that are anomalous with respect to some normal data.

3- Availability of Data Labels: The choice of a specific method of anomaly detection is influenced by the availability of labels. Supervised anomaly detection techniques build models using data labels for both normal and anomalous data points, semi-supervised anomaly detection build model for the normal data labels and aim to detect anomalies in unknown datasets, and unsupervised anomaly detection techniques make an assumption about the normal data as labels are not readily available. The work presented in this thesis assumes the normal data labels are available in order to build the model that describes the nominal behavior, then aims to classify unknown traces. The labels for anomalous data are only used for evaluating the accuracy of the approaches. We note that the labeling process is an error-prone task that might require human-knowledge in the case that such anomalies are easily achievable, e.g., not rare like traces originating from catastrophic failures.

4- Output of Anomaly Detection: Upon performing anomaly detection, we obtain a label for the trace either to be normal or anomalous. Optionally, an anomaly score can be provided to describe how it conforms to the normal behavior. A survey [28] categorizes the output of an anomaly detector as either a single-class or multi-class assignment, where for the single-class the classification of the data stream yields a result of being normal or not normal. For a multi-class approach, the data stream can be classified to one of many normal sub-classes or one of many anomalous sub-classes. The authors highlight the existence of techniques that classify some data streams as unknown, then later perform offline classification with human-aided support from a domain specialist to label such streams as either normal or anomalous. In our work, we use the single-class approach where a trace can be declared either normal or anomalous.

The rest of the survey paper is beyond the scope of the work presented in this thesis; however, the survey briefly discusses how sequential anomalies, similar to our interest, can be handled. One of the methods is to model the sequences, i.e., traces. Techniques to modeling traces include Markovian modeling [90] to study the probabilistic characteristics of event transitions. The work 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. Our work can put in the category of the sequence or trace modeling approaches, also called *Trace Mining*; We provide a detailed overview of its state-of-the-art.

2.2 Trace Mining Overview

Trace mining makes use of the increasing availability of system-specific execution traces as it offers new opportunities for capturing, modeling, and analyzing non-trivial behavioral patterns that cannot be predicted at design time. For trace mining, an entity introduced to the existing infrastructure processes the data blindly in order to build profiles and flag anomalous behavior to the end user. Related works report successful applications of data-driven modeling techniques in anomaly detection [32, 33, 55], security [91], resource management [43, 63], among other applications. However, there is still work to do on validating the effectiveness of data-driven analysis methods in real-world scenarios.

A related review [26] categorizes the approaches for **offline mining of data streams**, which correspond to event traces in our work. The authors categorize mining data streams techniques from the perspective of data handling into two categories; either by processing a subset of the data stream or by summarizing the whole stream. The former approach uses techniques such as sampling and load shedding. The later one uses synopsis of data structure or trace aggregation which fits our work since we summarize the trace data for analysis, i.e., represent a trace into some form before analysis. Traditional basic summarization techniques include histograms or frequency analysis; alternatively, trace sampling uses probabilistic functions to process data points selectively. Similarly to trace sampling, load shedding drops sequences of data streams entirely. Such approaches for sampling and shedding might not be a good choice for anomaly detection as the process of data elimination should be aware of anomalies.

A survey [20] summarized the research on mining data streams for anomaly detection. The main related research work uses hidden markov models (HMM) [70], rule inference [49] to build finite state automata (FSA) [79]. We discuss the basics of our trace mining approach in Chapter 3 which fits into the category of sliding window-based anomaly detection techniques. Related to the sliding window technique used by our work, episode mining [58] 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 [53] and software specification [85].

Online detection of anomalies in data streams using models extracted offline is one of the actively researched problems [19]. A survey [97] on anomaly detection for wireless sensor networks, which is related to our work, discusses problems similar to what we tackle in this thesis. They highlight the problem of online anomaly detection as one of the currently required research problems in the field. One example mentions the need for online techniques that meet the general requirements of scalability, being unsupervised, and allowing automatic parameter configuration.

As we discuss later in Chapter 5 an online variant of our anomaly detection work, one advantage of online anomaly detection is allowing to employ corrective measures right after an

anomaly is detected. The survey highlights the suitability of sequence-based techniques for online anomaly detection, as it allows for the assignment of scores to windows of data on arrival. For example, in contrast to the work presented in this thesis, trace similarity and frequency-based approaches are not applicable because as they need an entire trace for analysis.

The authors in [65] on data stream clustering and classification discusses how the traditional data mining algorithms can be adapted for the problem data stream mining. They highlight examples for the constraints of data stream mining as the necessity of being a single-pass algorithm having a real-time response. Single-pass constraint comes from the fact of expensiveness of the I/O operations with respect to the memory operation, however, as we propose later, the constraint can be relaxed by having a short-term memory that can be utilized when needed to improve the analysis. Another related constraint is the bounded memory available, which comes from the fact that the data generated can be considered infinite.

2.3 Arrival Curves & Real-time Systems

Arrival curves, originally introduced in Network-Calculus [45], are used to analyze Quality-of-Service (QoS) in packet-switched networks with the focus on modeling arrival traffic and deriving performance guarantees. The curves are widely used as part of service-level agreements (SLA) [71], which are contracts between a user and a provider to specify the expected behavior for a provided service. A variant of network calculus, named Real-Time Calculus (RTC), uses arrival curves for modeling temporal workloads and performing an exhaustive analysis of traditional real-time systems [75, 88].

Arrival curves are functions of interval time domain that provide upper and lower limits to the number of events that can occur in a system within any given time interval of length Δt [41]. We could conceptually obtain arrival curves from a timestamped trace by sliding a window of arbitrary length Δt over the time axis, keeping track of the maximum and the minimum number of events enclosed within the window while scanning the entire trace. However, for traditional analytical methods, designers do not construct arrival curves from finite traces but rather derive them from high-level mathematical models of the target system.

The underlying idea is to obtain guaranteed performance metrics by analyzing generic behavioral patterns that will bound any possible curve that could be observed during operation. [16] describes how an upper-bound arrival curve can be constructed analytically by extracting parameters such as the largest packet or long-term arrival rate from packets trace. Similarly, a lower-bound arrival curve can be constructed by analyzing the gaps in the packets trace.

To describe such curves, a commonly used form of arrival curves is *affine arrival curves* [22], where an arrival curve can be represented as a burst b and arrival rate r. The burst b represents the size of the maximum amount of network data that can arrive at once, while r represents the long-term arrival rate of the data. Another specification called *TSpec* [81] bounds arrival curves using a combination of two affine arrival curves boundaries. One boundary represents the maximum traffic behavior while the other boundary maps the long-term or the average behavior. In this case, the TSpec is the minimum of both boundaries. We expand on these definitions as we employ both characterizations in Chapter 3 and Chapter 5 to describe the arrival curves for anomaly detection purposes.

In traditional real-time systems, designers can assume that events occur following an application model with a known period, jitter, and delay (PJD models) [30]. Designers then map these models to generic templates of arrival curves representing worst-case scenarios, which facilitates derivation of guaranteed performance metrics using formal methods even before system deployment [75]. As such, designers approximate extreme scenarios using simple event patterns whose equations are amenable for formal analysis. This approach is ill-suited for characterizing complex real-time systems since such templates are abstract by definition.

Closely related to this thesis work, Event Count Curves (ECC) introduced in [67] describes the occurrence of event types in a structured event stream. In other words, the curves are constructed through computation of events counts rather than inferring parameters about the trace. We refer to that kind of arrival curves as *empirical arrival curves*. We conjecture that the mentioned body of work on arrival curves can be adopted to empirically constructed curves that would describe the behavior of real-time systems.

Authors in [13] acknowledge the lack of applications that deploy bounding approaches like empirical arrival curves for real-time analysis problems despite the similarities they share with SLA problems. This thesis considers the similarity between a system behaving normally as a provider who is meeting with some specified service-level agreement, where we characterize the system specifications from event traces collected under normal behavior and assess the conformity of the system to these specifications during deployment. We use the approach of modeling arrival traffic but through the characterization of event traces for anomaly detection.

Chapter 3

Defining Empirical Arrival Curves From Event Traces

In this chapter, we introduce the definitions and metrics for empirical arrival curves. The presented metrics along with their semantics will provide the basis needed to exploit the curves for characterizing the recurrent behavior of real-time systems throughout the thesis.

3.1 Definitions

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.*

Definition 2. *(Trace) A trace $\widehat{T}$ is an ordered sequence of timestamped events e_{t_i} generated by a given system, where t_i is a unique timestamp having an associated index i within the sequence. Formally, $\widehat{T} = \{e_{t_1}, e_{t_2}, \ldots, e_{t_{|\widehat{T}|}}\}$, where $|\widehat{T}|$ is the length of the trace.*

We refer to a trace that follows Definition 2 as a *timestamped* trace, however, if a timestamp associated with an event represents the index or the event position within a trace, we refer to this trace as an *Indexed* trace. To generalize our definitions, the following definitions and metrics consider a trace to be timestamped.

To compute arrival curves for a specified event type ε, we extract the trace events e_{t_i} that match ε using Definition 3.

Definition 3. *(Trace Extraction Operator) The operator $\uparrow$ takes a subset of trace $\widehat{T}$ from Definition 2 to create a trace T_ε of an event type ε as follows:*

$$\widehat{T} \uparrow \varepsilon \mapsto T_\varepsilon, \text{ where } T_\varepsilon \subset \widehat{T}, \text{ such that } e = \varepsilon, \forall e \in T_\varepsilon \tag{3.1}$$

Example 1. *(Timestamped Trace Example) Consider the following trace example* $\widehat{T} = \{a(t_1), a(t_2), b(t_3), c(t_4), a(t_5), a(t_6), b(t_7), b(t_8), c(t_9), a(t_{10}), a(t_{11}), a(t_{12}), b(t_{13}), b(t_{14}), a(t_{15}), c(t_{16}), c(t_{17}), c(t_{18}), a(t_{19}), a(t_{20})\}$*, there exists* $|\widehat{T}| = 20$ *events of event types* $\varepsilon \in \{a, b, c\}$.

Example 2. *(Indexed Trace Example) The indexed trace corresponding to the trace in Example 1 is as follows* $\widehat{T} = \{a_1, a_2, b_3, c_4, a_5, a_6, b_7, b_8, c_9, a_{10}, a_{11}, a_{12}, b_{13}, b_{14}, a_{15}, c_{16}, c_{17}, c_{18}, a_{19}, a_{20}\}$.

Given the sample trace from Example 1, the corresponding mapped timestamped trace T_a for an event type a as a result of applying the mapping operation $\widehat{T} \uparrow a$ is $T_a = \{a(t_1), a(t_2), a(t_5), a(t_6), a(t_{10}), a(t_{11}), a(t_{12}), a(t_{15}), a(t_{19}), a(t_{20})\}$. Similarly for Example 2, $\widehat{T} = \{a_1, a_2, a_5, a_6, a_{10}, a_{11}, a_{12}, a_{15}, a_{19}, a_{20}\}$. A trace T_ε contains a stream of indexed or timestamped events of a unique type ε making it analogous to a timestamped stream of bits on a network channel, where the timestamp is either an actual clock time associated with the event or its index in the trace. Consequently, we adapt the concepts of network calculus to real-time systems by defining empirical arrival curves.

Definition 4. *(Empirical Arrival Curve for ε) An arrival curve calculation for an event type ε denoted by $C_f(T_\varepsilon, \Delta t) \to \mathbb{R}$ applies a specified function f to the occurrence count C of events of type ε within all sliding windows of time interval $\Delta t \in \mathbb{R}^+$ contained in the time interval $[t_i, t_{|\widehat{T}|}]$ and positioned to start at e_{t_i}.*

We highlight that for indexed traces, the sliding window interval is an interval of event counts instead of a time interval. Now, we use Definition 4 to define two specific curves of interest which define the lower and upper bounds on the arrival behavior of an event type ϵ.

Definition 5. *(Max/Min Arrival Curves) A maximum arrival curve C_{max} uses a function f that provides the maximum counts for occurrences of ϵ within the sliding windows described by Definition 4. Similarly, a minimum arrival curve C_{min} uses a function f that provides the corresponding minimum counts.*

Applying Definition 5 to a given trace T_ε using a range of Δt values yields the maximum and minimum arrival curves of an event type ε. Throughout the thesis, the function f in Definition 5 uses the widely known SymTA/S [75] assumptions for computing the maximum and minimum event counts within a window, where the minimum computation considers the interval $(0, \Delta t]$ and the maximum computation considers the interval $[0, \Delta t)$. We discuss the details of computation in Chapter 7.

The curves C_{max} and C_{min} form an envelope that characterizes the arrival behavior of an event type within a trace which is analogous to the envelope used in SLA agreements for constraining the arrival of bit streams in a given network. We demonstrate the use of the previous definitions given the sample trace from Example 1, the corresponding mapped trace T_a for an event type a as a result of applying the mapping operation $\widehat{T} \uparrow a$ is shown in Figure 3.1. The dotted window represents an incomplete sliding window for which we discard the C_{min} computation. Figure 3.2 shows both C_{max} and C_{min} curves computed on the full range of $\Delta t \in [0, t_{|T_a|} - t_1]$ for $\varepsilon = a$ using the indexed trace $\widehat{T}$ of Example 1. We note that the red and green curves in Figure 3.2 are for visualization purposes, since the computation of C_{max} and C_{min} yields discrete points specified at the defined window sizes on the x-axis.

$$\widehat{T} = \{\ \mathrm{a}\ \mathrm{a}\ \mathrm{b}\ \mathrm{c}\ \mathrm{a}\ \mathrm{a}\ \mathrm{b}\ \mathrm{b}\ \mathrm{c}\ \mathrm{a}\ \mathrm{a}\ \mathrm{a}\ \mathrm{b}\ \mathrm{b}\ \mathrm{a}\ \mathrm{c}\ \mathrm{c}\ \mathrm{c}\ \mathrm{a}\ \mathrm{a}\ \}$$
$$T = \{\ \underbrace{\epsilon\ \epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \epsilon\ \epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}}\ \epsilon\ \epsilon\ \epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \epsilon\ \bar{\epsilon}\ \bar{\epsilon}\ \bar{\epsilon}\ \underbrace{\epsilon\ \epsilon}\ \}$$

Figure 3.1: Trace mapping example

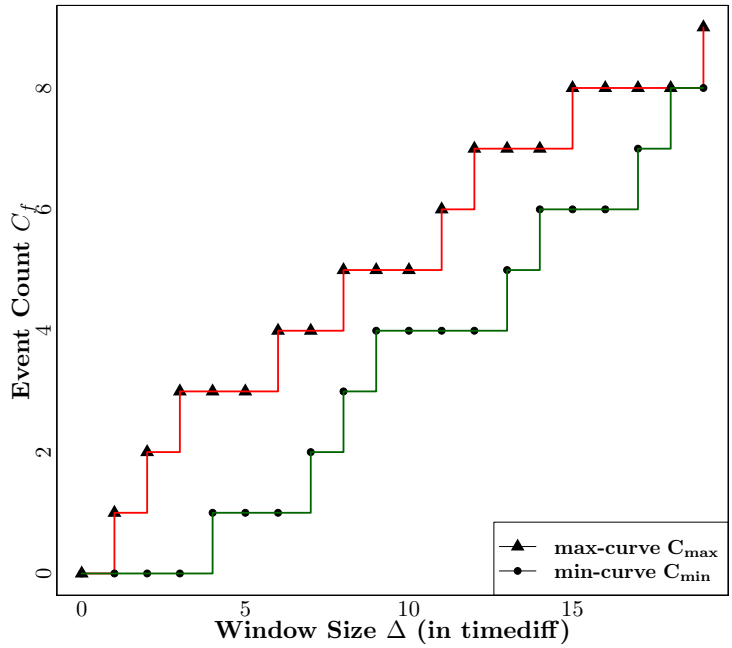


Figure 3.2: Min-Max curves for $\varepsilon = a$

The definitions of $C_{\max}$ and $C_{\min}$ leads to two observations that can be used to characterize the behavior of a system, the first observation:

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

Proof: Maximum and minimum arrival curves use cumulative computations over increasing window durations Δ, the corresponding $C_{\max}(T, \Delta)$ and $C_{\min}(T, \Delta)$ values are either the same or increasing. Only given the assumption that $C_f(T, \Delta t) \leq C_f(T, \Delta t + \Delta t)$ which is valid for f being max and min functions.

Now we define a plateau or steady interval of an arrival curve as an interval of successive Δt values having the same $C_f(T, \Delta t)$. The definition of the plateau leads to the second observation:

Theorem 2. *Longest plateau of a min-curve C_{min} exists at the beginning of the curve starting by point $(0, 0)$ and ending at the point having smallest Δt satisfying $C_{min}(T, \Delta t) = 1$.*

In other words, we can use the x-axis to indicate the maximum window of separation between any two events ϵ within a mapped trace T.

Definition 6. *(Difference 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 Δt for a given trace T as follows:*

$$C_{diff}(T, \Delta t) = C_{max}(T, \Delta t) - C_{min}(T, \Delta t) \tag{3.2}$$

The C_{diff} curve has some points of interest. For example, for any C_{diff} curve we have $C_{\text{diff}} = 0$ at both $\Delta t = 0$ and $\Delta_{\max} t = t_{|\widehat{T}|} - t_1$, since $C_{\max}(T, 0) = C_{\min}(T, 0) = 0$ and $C_{\max}(T, \Delta_{max} t) = C_{\min}(T, \Delta_{max} t) = |T|$ respectively. Otherwise, the curve values belong to $\mathbb{N}_{\geq 0}$, because $C_{\max}(T, \Delta t) \geq C_{\min}(T, \Delta t)$ for all values of Δt. Formally,

$$C_{\text{diff}} = \begin{cases} 0, & \text{if } \Delta t = 0 \\ \mathbb{N}_{\geq 0}, & \text{if } 0 < \Delta t < t_{|\widehat{T}|} - t_1 \\ 0, & \text{if } \Delta t = \Delta_{max} t \end{cases} \tag{3.3}$$

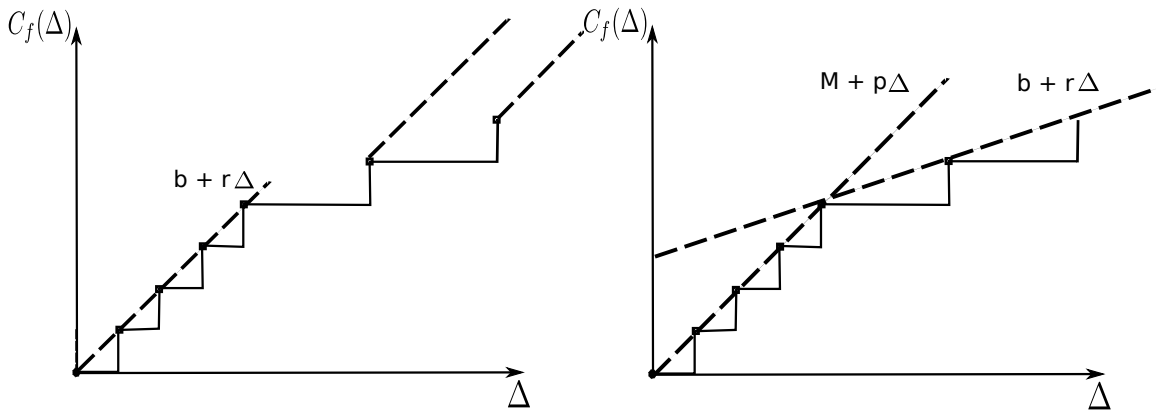
We use this observation to describe the recurrent behaviors captured in the event traces generated by real-time systems in Chapter 5.

3.2 Metrics

The previous definitions allow us to introduce a couple of metrics that characterize the shape of arrival curves obtained from timestamped event streams in addition to existing metrics from the literature. In the later chapters, we exploit those metrics to provide explanatory power for defining higher-level features which analyze the behavior of real-time systems.

3.2.1 Steadiness of an Arrival Curve

As discussed in Chapter 2, there exists different approaches to specify the bounds on traffic profiles in arrival curves. We use two of the common characterizations which are the $(burst, rate)$-model and the *TSpec*-model. Figure 3.3 presents a visualization to both approaches on an empirical arrival curves example.



(a) Affine parameters visualization

(b) TSPEC parameters visualization

Figure 3.3: Visualization for affine and TSpec specs of arrival curves

For the $(\mathbf{burst}, \mathbf{rate})$**-model**, an arrival curve can be bounded by a curve having a burst b and a long-term rate r, i.e., represented by a function $b + r\Delta$ as shown in Figure 3.3a. In practice, the arbitrary choice of an affine curve can favor over-approximation or under-approximation of an arrival curve through controlling the burst b and the rate r parameters. In the thesis, we use affine curves that start at the origin $(0, 0)$. As a result, in this thesis, we use a $(burst, rate)$-model that is equivalent to defining the long term rate r with $b = 0$.

The ***TSpec*-model** is a widely known characterization which uses a couple of $(burst, rate)$-models. One to describe the initial arrival behavior of the flow while the second describes the

average behavior. As shown in Figure 3.3b, the arrival curve can be bounded by the minimum of both curves $M + p\Delta$ and $b + r\Delta$, where p is the peak rate and M is the maximum burst. In this thesis, we employ a *TSpec*-model whose initial $(burst, rate)$-model passes by the origin.

In this thesis, we define a third metric **Steadiness Slope** $\bar{\mathbf{S}}$ that describes how C_f increases with the increase of Δt by studying all the steady intervals or plateaus of that curve. We relate the metric to the previously discussed $(burst, rate)$-model and *TSpec*-model in Chapter 5 where the metrics are used for anomaly detection purposes.

The steadiness slope $\bar{S}$ calculates the mean of the slopes of all virtual lines L_j that connects the last point of each two successive plateaus where j refers to the index of the plateau considered for calculation. The slope of a single virtual line L_j, denoted as S_{L_j}, having the start point at Δt_s and the end point at Δt_e where $\Delta t_e > \Delta t_s$ can be calculated using the following equation:

$$S_{L_j} = \frac{C_f(T_\varepsilon, \Delta t_e) - C_f(T_\varepsilon, \Delta t_s)}{\Delta t_e - \Delta t_s} \tag{3.4}$$

To calculate the mean of slopes $\bar{S}$ of all virtual lines L_i defined over n plateaus, we use Equation 3.5:

$$\bar{S} = \frac{\sum\limits_{i} S_{L_i}}{n},\ where\ \bar{S} \in\ (0, 1). \tag{3.5}$$

The value of $\bar{S}$ for event $\varepsilon = a$ in Example 1 can be obtained by applying Equations 3.4 and 3.5 as follows: $\bar{S} = \frac{1}{5} * \{\frac{2}{4} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}\} = 0.42$.

The steadiness slope $\bar{S}$ describes the occurrence behavior of the system events over an interval by describing the density of the occurrence, i.e., count of event ε relative to the window duration Δt. For example, a higher value for mean slope $\bar{S}$ refers to a higher increase in the count $C_f(T_\varepsilon, \Delta)$ as Δt 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 arrival curves calculated for different events within the same trace or calculated for the same event using different traces.

3.2.2 Proximity of Multiple Arrival Curves

The next metric calculates the area under an arrival curve C_f, denoted as Λ. However, 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. Therefore, we define a metric $\mathbb{P}$ that calculates the proximity of two arrival curves to each other by calculating the ratio of their area under curve values. Formally,

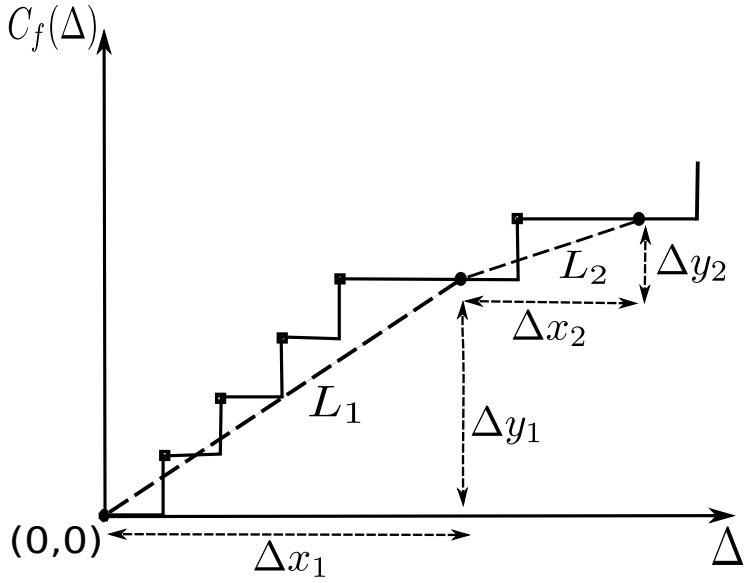


Figure 3.4: Steadiness slope metric visualization

$$\mathbb{P} = \frac{\Lambda_{f_1}(T_1)}{\Lambda_{f_2}(T_2)}, \text{ where } \Lambda_f(T) = \int C_f(T)\, d(\Delta t) \tag{3.6}$$

Equation 3.6 defines a generic proximity metric between two arrival curves obtained from two arbitrary traces. We employ multiple variants of this metric. One calculates the proximity of curves C_f which use the same function f, i.e. either C_{max} or C_{min} obtained from different traces, to reason about the conformance of a system with the expected real-time behavior in Chapter 4. The other variant of $\mathbb{P}$ can be used to characterize the variation between the C_{max} and C_{min} obtained using the same trace T_ε for clustering purposes in Chapter 5 as follows:

$$Prox_{min/max} = \frac{\Lambda(C_{\text{min}})}{\Lambda(C_{\text{max}})}\ ,\ Prox \in\ (0, 1] \tag{3.7}$$

$Prox_{min/max}$ describes the variation of arrival behavior of a given event in a trace by monitoring the variation between the minimum and maximum counts of events obtained using instances of sliding windows of different durations. For the example shown in Figure 3.2, $Prox_{min/max}$ can quantify 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.

Conclusion

In this chapter, we presented the basic definitions that enabled the first adaptation of network calculus concepts to timestamped event streams instead of network streams. We defined a set of metrics which presents novel characterizations that we conjecture would enable anomaly detection for cyber-physical systems analogously to the well-known application of defining service-level agreements in networks domain. We highlight that the definitions and metrics used in this section serve the purpose of tackling the thesis statement; however, the curves can be further described using more functions and metrics for different purposes.

Chapter 4

Anomaly Detection using Arrival Curves: TRACMIN

This chapter presents novel characterizations using the definitions and metrics of empirical arrival curves to enable anomaly detection for cyber-physical systems. The events arrival from these systems turns out to be analogous to the arrival of network data in the well-known application of defining service-level agreements for networks domain.

4.1 Approach

Behavioral 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.

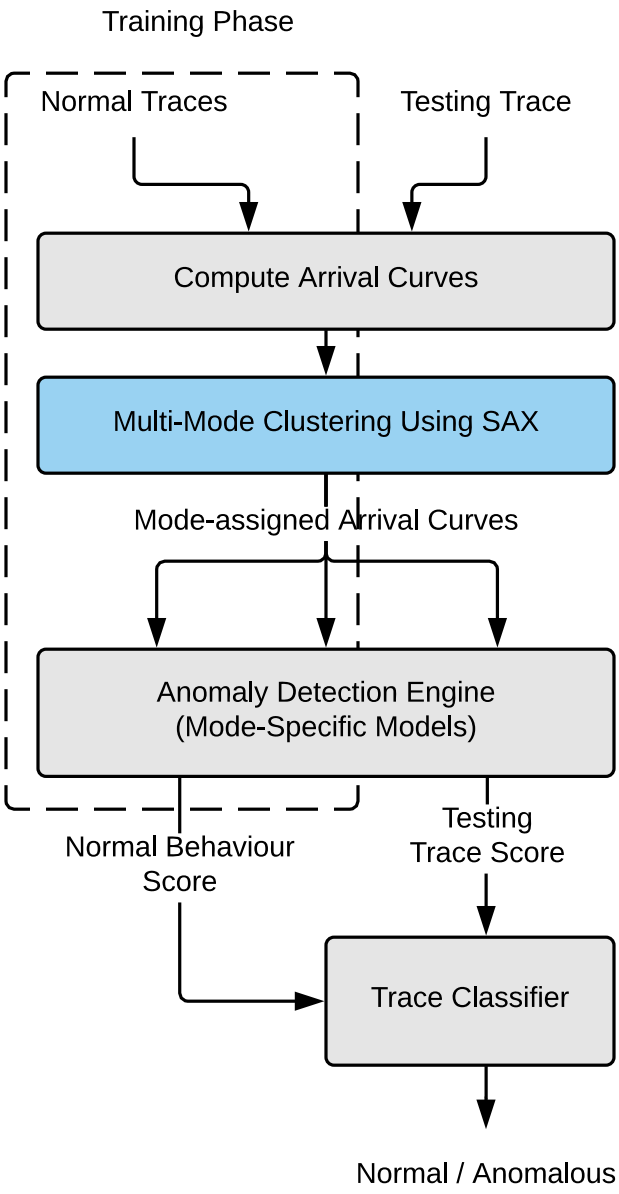


Figure 4.1: Trace mining framework using arrival curves - TRACMIN

We introduce a classification framework in Figure 4.1 which exploits arrival curves as high-level features suited for describing the behavior of real-time systems. We conjecture that behavioral anomalies in an event trace would affect any of the metrics described in Chapter 3, 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.

We adopt a generic semi-supervised classification framework as described in [19]. The framework builds a model using a set of event traces for a known expected behavior, and then inspects a set of unknown traces for being either anomalous or not to that model. Within the classification framework, we incorporate a novel clustering engine, that can also be used as a standalone framework that will be discussed in Chapter 5. The clustering framework employs arrival curves for the detection of operational modes by abstracting the generated traces from a real-time system. Figure 4.1 shows the building blocks for the framework as follows:

Calculating Arrival Curves

During the training phase, the output from this building block is a set of pairs of min- and max-curve, i.e., C_{min} and C_{max}, for each event type ε per each trace in the known traces set. For more robust results, we only calculate the 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 later in the chapter.

Building the Training Model

The second block shown in Figure 4.1 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 arrival curves calculated using the same function f applied to the same event type ε to obtain a corresponding aggregated 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 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., 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 [84] 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 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}. Figure 4.2 shows an example of the arrival curve model from a case study that we discuss later in the Chapter that uses timestamped traces, this is indicated by having the window size in nanoseconds on the x-axis of the figure.

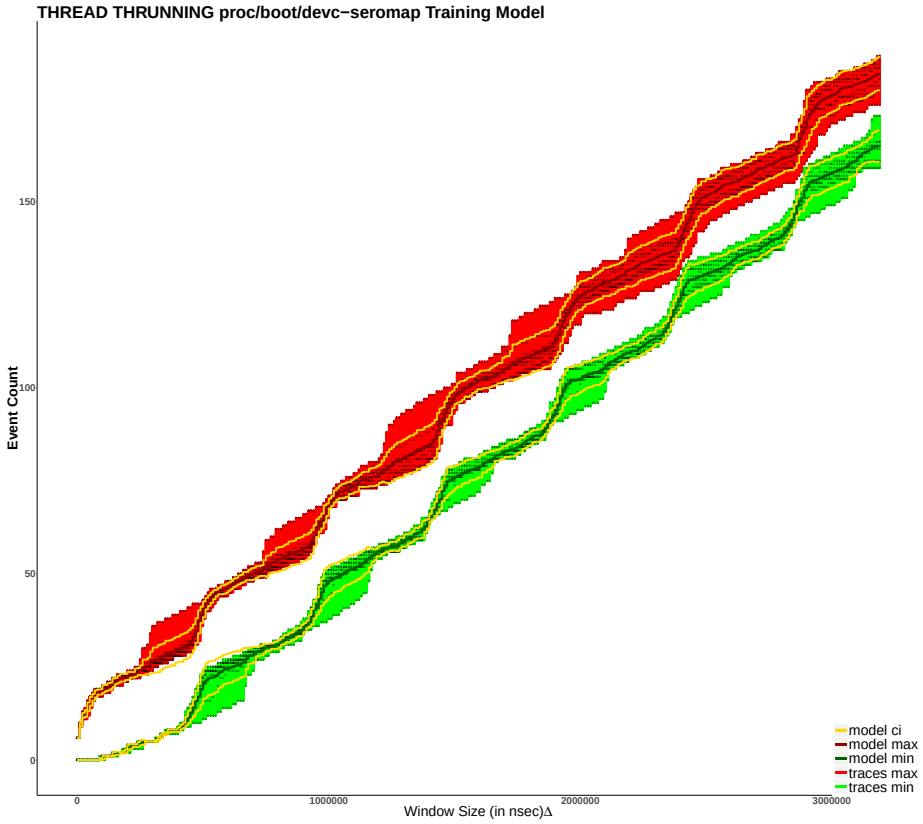


Figure 4.2: Arrival curves model - Aggregate of C_{max} and C_{min} with confidence interval curves. [timestamped traces]

Trace Classification

Using arrival curves allows pinpointing the specific events that caused the anomalous behavior 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 4.3 shows an example from a case study that uses indexed event traces as indicated by the window size in events count on the x-axis. 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 4.3a, opposed to Figure 4.3b, 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 behavior 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 in Chapter 3, the technique in Example 3 shows how to represent the arrival curves as a frequency distribution to describe the curve shape for statistical testing.

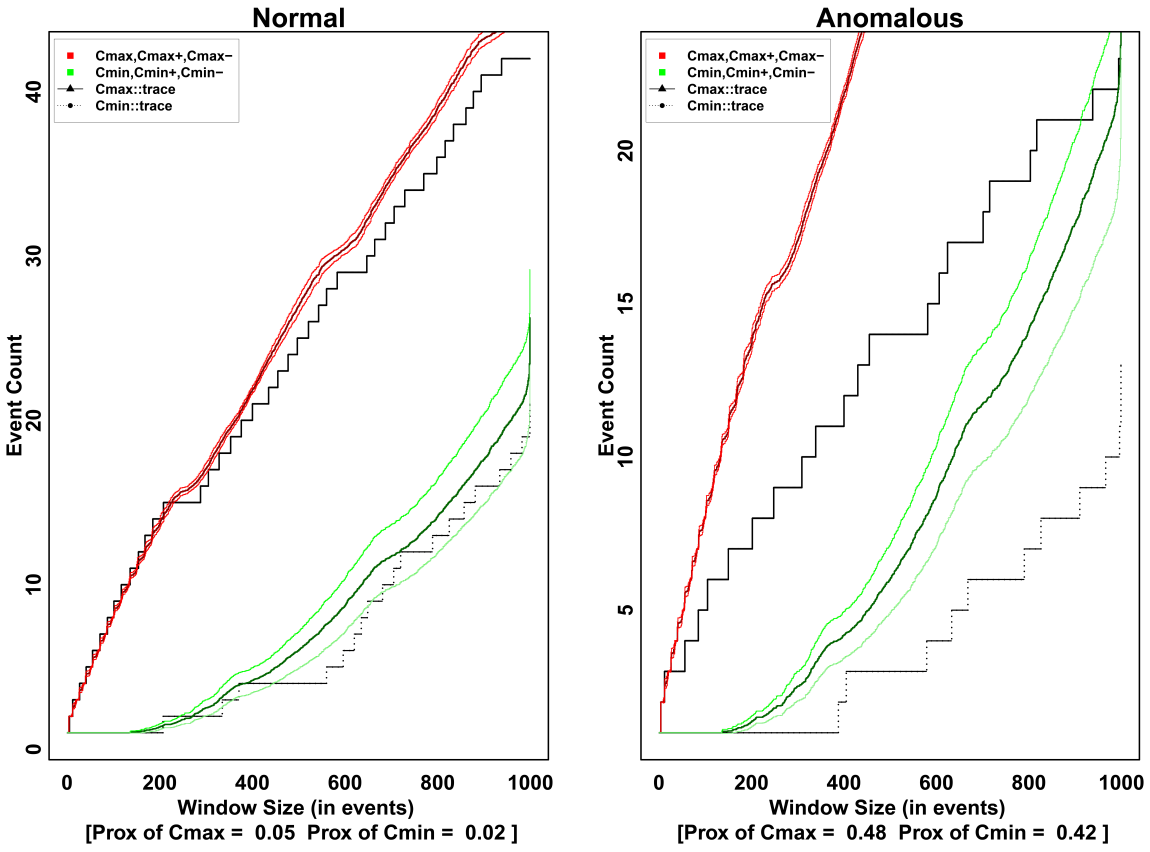
Example 3. *Consider a max curve with values* $C_{max} = \{1, 2, 2, 3, 4, 4\}$ *calculated using* Δ_{max} *of 6 time units. We partition the curve to a set of intervals of unique* $C_{max} \in \{1, 2, 3, 4\}$ *with corresponding* Δt *intervals* $\in \{1, 2, 1, 2\}$.

To automate the curve similarity procedure, we apply the Wilcoxon-Mann-Whitney test, also known as the Mann-Whitney U test [57], 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 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 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 the evaluation section.

The 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

Figure 4.3: Normal vs anomalous arrival curves for `THREAD_THRUNNING` [indexed traces]

Figure 4.3a 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 4.3b 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. We highlight that the curves shown in Figure 4.3 consider the indices of the events within a trace not their actual timestamps, this is a more abstracted use for the definitions in Chapter 3, we expand on this in the evaluation section.

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 as we deploy the anomaly detection framework on indexed and timestamped traces for evaluation purposes.

4.2 Evaluation Method

We evaluate the classification performance using the receiver operating characteristic (ROC) curve [25]. A given 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:

$$\text{TPR} = \frac{\text{True Positives Detected by Classifer}}{\text{Actual Positives in Testing Set}} \tag{4.1}$$

$$\text{FPR} = \frac{\text{False Positives Flagged by Classifier}}{\text{Actual Negatives in Testing Set}} \tag{4.2}$$

A 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. Whenever the perfect classification result cannot be achieved, a better classifier will have a higher area under the ROC curve.

In practice, the application of the anomaly detector dictates whether false positives or false negatives can be tolerated. For example, a medical device that generates many false alarms results in the patient stopping to trust the device, which is known as alarm fatigue [21]. In such systems, a lower false positive rate is more desirable. On the contrary, a lower false negative is desirable when a negative case can be fatal. For example, a false detection for an anomaly might be costly, but a false classifier miss might be catastrophic with fatalities.

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

The threshold specifies the minimum number of anomalous 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 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 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 $S_{\%}$

The parameter 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 arrival curves. However, using more event types ε in the training model might overfit the training data and hence degrade the classifier performance.

Maximum sliding window size Δ_{max}

Increasing Δ_{max} might allow detection of anomalies whose disruption effect might go undetected using shorter window sizes. The deviation in Figure 4.3b between the result obtained using a testing trace with respect to the training model is significant after $\Delta \gtrsim 200$ units, so using a $\Delta_{max} = 200$ units will classify the curve of that event type to be normal however using a $\Delta_{max} = 1000$ units 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. The units of Δ_{max} can be number of events or number of timeunits as we discuss using the presented case studies.

4.3 Experimental Evaluation

In this section, we use event traces generated from deployed real-time systems of two industrial case studies. One we use to demonstrate the applicability of the approach on indexed traces, while the other case study uses timestamped event traces.

4.3.1 Indexed Traces: UAV Case Study

We use kernel event traces generated from an 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 4.4 shows the event attributes used in our experiments.

```
Index, class           ,event           ,pname
1    , PROCESS         ,PROCCREATE_NAME ,proc/boot/procnto-instr
2    , THREAD          ,THREADY         ,proc/boot/procnto-instr
3    , INT_ENTR        ,0x00000044      ,NA
4    , INT_HANDLER_ENTR,0x00000044      ,NA
5    , INT_HANDLER_EXIT,0x00000044      ,NA
6    , INT_EXIT        ,0x00000044      ,NA
7    , COMM            ,SND_MESSAGE     ,proc/boot/qconn
8    , COMM            ,REC_MESSAGE     ,proc/boot/qconn
9    , THREAD          ,THSIGWAITINFO   ,proc/boot/devc-pty
.....           ,.....           ,.......
```

Figure 4.4: Indexed 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 behavior but does not conform to the training traces.

To evaluate the anomaly detection framework discussed in this chapter, we use a modeling set of roughly 100 known traces of an expected behavior from the UAV and a set of roughly 40 unknown traces divided equally between normal and anomalous traces for each experiment. The normal traces used in the testing phase were different from the ones used in the training phase.

Table 4.1 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 4.5. 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%	$\Delta_{max} = 500$, $S_{\%} = 3$, $\nu_a = 3$

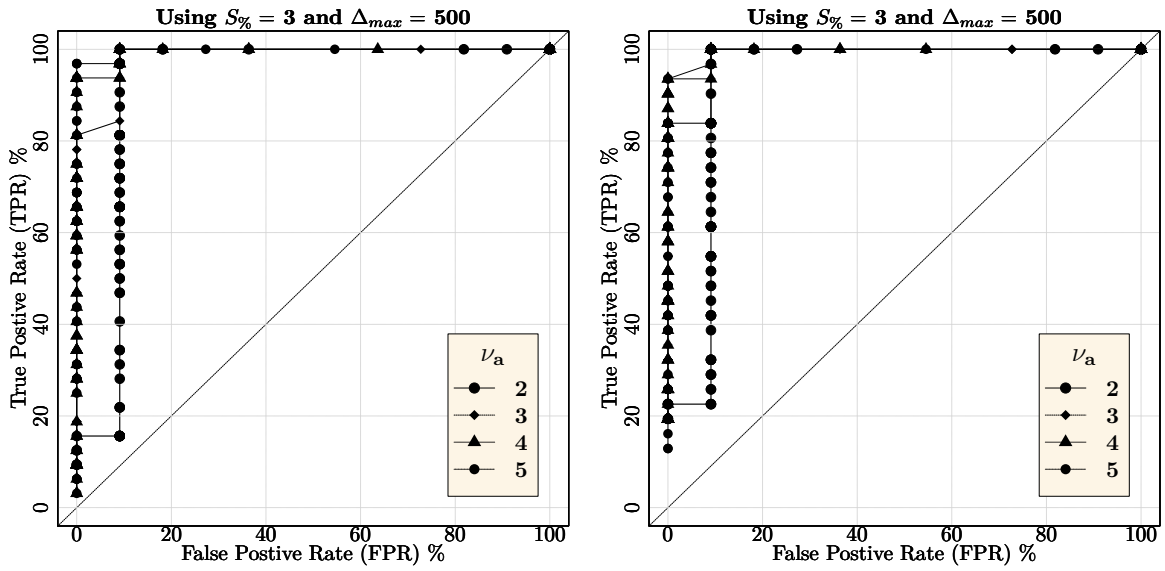
Table 4.1: Indexed Traces Classification Results

The results in Table 4.1 and Figure 4.5 show the feasibility of the proposed framework on indexed traces, we now discuss another case study where we consider the timestamps of the events as originally proposed in the definitions of Chapter 3.

4.3.2 Timestamped Traces: SSPS Dataset

We generate traces using a device under test (DUT) that mimics a data collection system which executes recurrent processes that sequentially sense, process, and send some sensor information over a communication network. The Sequential Sense Process Send Dataset (SSPS) dataset comprises a set of event traces generated using the QNX tracelogger utility. The dataset involves variants of data generation hardware, data storage media, and different configuration for the underlying DUT operations. The experiment architecture uses computational servers with CPU and GPU capabilities to process the SSPS dataset traces stored in a PostgreSQL database.

The SSPS dataset specified data collection configuration generates traces from the BeagleBone hardware platform for the three modes of operations. We select traces that represents multiple runs from the Beaglebone board collecting 5000 data blocks from the pseudo-device *hd0*. The objective of the anomaly detection framework is to correctly classify a given trace as normal



(a) *sporadic* Scenario

(b) *fifo-ls* Scenario

Figure 4.5: Effect of tuning the count threshold ν_a on indexed traces classification

or anomalous, in addition to providing a quantifiable anomaly score. To represent performance anomalies, we deploy the data collection system on two embedded boards with distinguishable hardware capabilities which are BeagleboneBlack and ARM SabreLite embedded boards. System configuration anomalies vary the source of data from *hd0* and *random* pseudo-devices and the count of data blocks processed.

In this dataset, we use the timestamps associated with the events in the QNX trace. So as we discussed in Chapter 3, the Δ parameter is measured in time units instead of window counts, which also applies to the tunable parameter Δ_{max}.

The classification results in Table 4.2 and Table 4.3 show one dataset to be more challenging than the other. This can be verified by domain knowledge, where the difference between the hardware capabilities of the Beaglebone and Sabrelite makes the classification using timing information a less challenging task, i.e., (TPR,FPR) = (100%,3%), compared to classifying the different configurations within the same BeagleBone board, i.e.,(TPR,FPR) = (85%,45%). The results in both tables are not as good as the results obtained in the UAV case study even for the best tuning parameters used; We improve this result by introducing a multi-mode clustering approach in Chapter 5 which builds a mode-specific model to better describe the underlying behavior of the system.

	Hardware Variants: BeagleBone vs SabreLite	
	Trace Count: (Train, Normal, Anomalous)	(TPR,FPR)
Without Clustering	(132,67,69)	(100%,3%)

Table 4.2: Timestamped Traces Classification Results: Hardware Variants

	Misconfiguration: *hd0* 2500 vs *random* 5000	
	Trace Count: (Train, Normal, Anomalous)	(TPR,FPR)
Without Clustering	(57,55,55)	(85%,45%)

Table 4.3: Timestamped Traces Classification Results: Hardware Misconfiguration

4.4 Conclusion

In this chapter, we used arrival curves as high-level features in a framework that use multi-dimensional features to allow reasoning about the conformity of the behavior expressed by an unknown trace to a system behavior defined by a previously collected set of known traces. Experimenting with multiple datasets from a real-time system demonstrates the feasibility and viability of our approaches on both indexed and timestamped traces generated from an industrial real-time operating system experiencing anomalous behavior.

Chapter 5

Extending the Applications of TRACMIN

In this chapter, we introduce more applications for empirical arrival curves. First, we improve the classification results obtained in Chapter 4 for the SSPS case-study. We achieve this through a novel clustering approach that translates empirical arrival curves into a set of strings before performing traditional string distance evaluation. Second, we extend the application of TRACMIN to the domain of streaming traces, i.e., on-the-fly anomaly detection. Finally, we build on some of the definitions introduced in Chapter 3 to use arrival curves for mining recurrent behavior which is intrinsic to real-time systems.

5.1 Multi-Mode Clustering in TRACMIN

The observation we encountered when evaluating the traces that involved multiple modes of operation when using a single-mode based approach for classification aligns with the discussion in a recent survey [65]. The survey presents some techniques that compose a normal behavior of a given system as a set of normal classes. For example, having a set of features that describe some distinct system behaviors all corresponding to an expected or normal behavior, i.e., an aggregate curve for system startup, others for different execution phases such as communication, file transfer, sleep, etc. Combining all normal behaviors into an aggregate curve can compromise the accuracy of the model when classifying unknown behavior as we experienced in Chapter 4.

The framework in Figure 5.1 shows the main building blocks for a mode-clustering approach using arrival curves. For a given anomaly detector, the primary goal is to correctly classify a given trace to be either normal or anomalous; however, the clustering engine makes sure that the traces that undergo the classification process originate from the same mode of operation. The reasoning

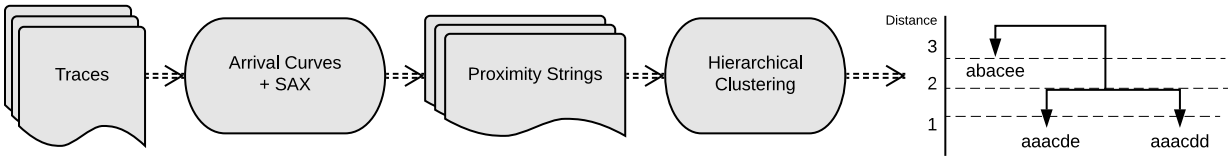


Figure 5.1: Clustering modes of operation using SAX strings of arrival curves

behind this approach is to minimize the number of traces which might be misclassified because they belong to a mode of operation that is not well represented in the normal model. In other words, having a clustering framework allows for building normal profiles for the different modes of operation so an unknown trace can be classified against the model corresponding to its mode of operation.

5.1.1 Approach

The proposed approach aims to cluster the arrival envelope enclosed between C_{max} and C_{min} based on one of the slope metrics applied to C_{max} and the proximity $\mathbb{P}$ of C_{min} to C_{max}. Given these two metrics, we can intuitively automate the process of differentiating between the arrival envelopes of an event ε. Specifically, we apply a choice from the multiple methods for computing a slope-based metric of a given curve discussed in Chapter 3, and we employ a variant of the proximity metric $\mathbb{P}$ which computes the ratio of the area under the C_{min} curve to the area under the C_{max} curve. This variant of $\mathbb{P}$ ranges between 0% and 100%.

We employ the Hierarchical Clustering method since it does not require specifying the number of clusters beforehand. In specific, we apply the function *hclust* in R [72]. The function applies the complete linkage clustering approach which is updated iteratively using the Lance–Williams dissimilarity formula [60]

A straightforward approach is to measure the edit distance between the traces themselves as performed in [24] where the authors apply distance measures to quantify the data loss within a trace compared to a normal trace by relying on hierarchical clustering for anomaly detection purposes. However, this approach can be challenging for traces of variable lengths and is not directly applicable to timestamped traces. We propose an approach which transforms **the proximity curve** $\mathbb{P}$ of C_{min} to C_{max} as a fixed-length character string using Symbolic Aggregation Approximation (SAX) [50].

The edit distance between those character strings serves as the input to the hierarchical clustering approach. The reason behind using SAX is that Euclidean distance between the

obtained two strings is a lower bound to the Euclidean distance between their corresponding proximity curves [50].

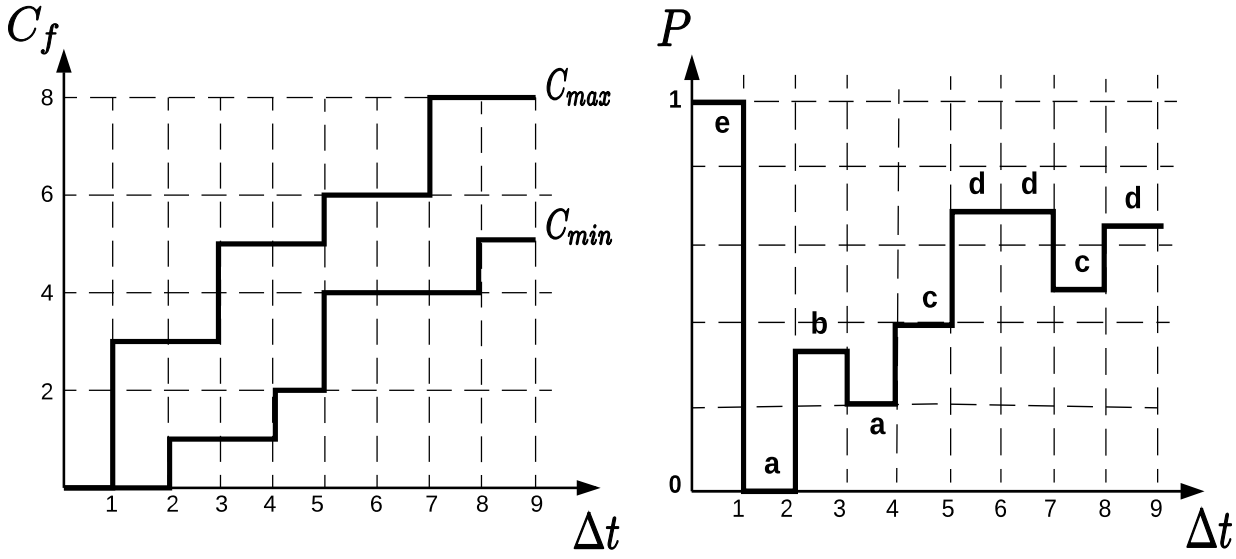


Figure 5.2: Applying SAX to proximity curve of $C_{\min}$ to $C_{\max}$

Example 4 demonstrates how SAX obtains a character string that symbolically represents the proximity curve. SAX applies Piecewise Aggregate Approximation (PAA) to the proximity curve $\mathbb{P}$ and assigns characters based on a Gaussian distribution of thresholds on the arrival counts (y-axis) for specified Δt interval values (x-axis). The selection of suitable threshold levels and Δt intervals can be performed throughout an iterative process for best results. Example 4 defines SAX breakpoints in the range [0,1] for an alphabet of 5 characters to describe the proximity levels on the y-axis. The length of the character string corresponds to an arbitrary number of 9 PAA segments.

Consequently, the proximity curve can be represented as a fixed-length character string using a defined alphabet. To build a distance matrix, we use the Levenshtein distance [47] to compute the edit distances between the SAX strings of the different proximity curves $\mathbb{P}$. The Levenshtein distance represents the edit distance between two character strings by counting the numbers of characters to be replaced until two strings are matched.

Example 4. *Figure 5.2 shows the arrival curves C_{max} and C_{min} along with the corresponding proximity curve $\mathbb{P}$ for a defined set of window intervals Δt. Applying SAX to the proximity curve $\mathbb{P}$ with the mentioned settings for alphabet size and PAA segments yields the character string* eabacddcd. *Assume we obtain another string* eaaaccddd *for another proximity curve, then the Levenshtein distance between the two strings is 3.*

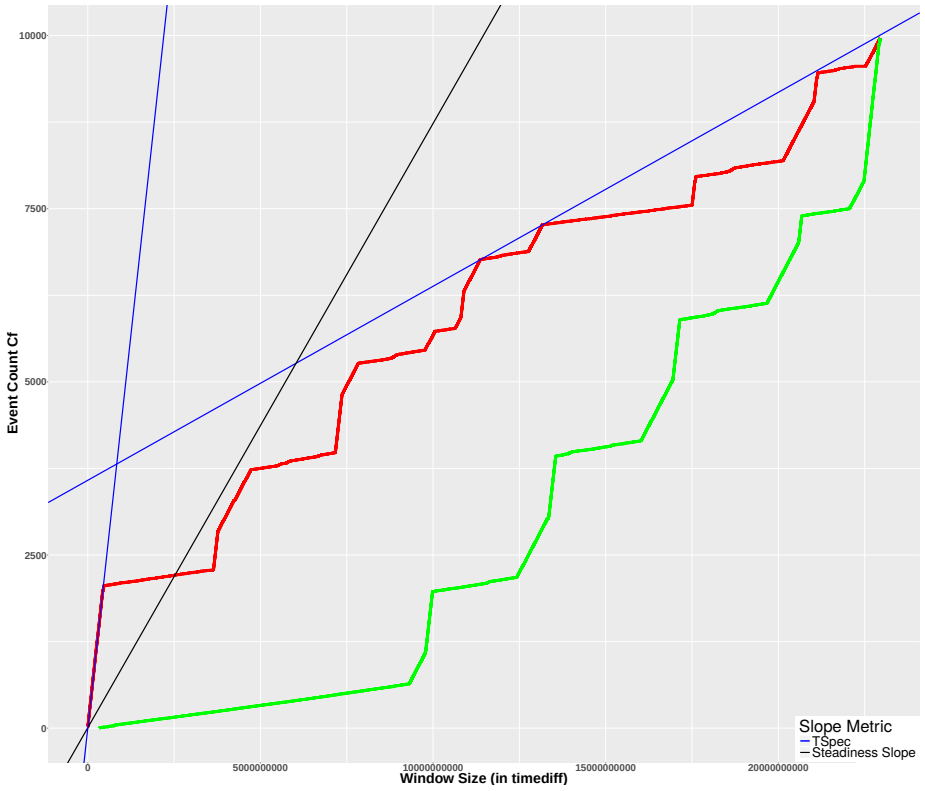


Figure 5.3: Visualization of the different slope metrics

To fully characterize the arrival curves envelope, we build another distance matrix using Euclidean distance between a the values of a chosen **slope metric**. Figure 5.3 visualizes the slope metrics discussed in Chapter 3. The affine curves of the *TSpec*-model that upper bounds $C_{\max}$ curve - in blue - are obtained using Quantile Regression from the R package *quantreg* [40]. The figure also shows the steadiness slope metric $\bar{S}$ - in black - which averages the slopes between the steady plateaus to encapsulate the overall shape of the arrival curve $C_{\max}$.

The aggregate distance matrix used for clustering a set of arrival curves is a weighted sum of the two distance matrices obtained for the slope-based metric and the proximity strings where the distance matrices are normalized because of their different scales. Finally, we apply the hierarchical clustering technique to the aggregate distance matrix to obtain a bottom-up dendrogram, which is a tree diagram structure whose height shows the distance between the desired number of clusters. We show the feasibility of the approach by presenting a dendrogram that successfully clusters traces arising from different modes of operations of a real-time system.

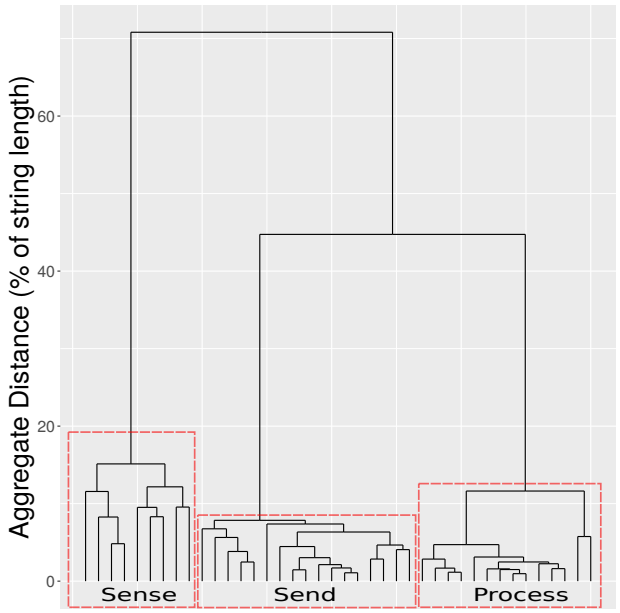


Figure 5.4: Trace clustering based on modes *Sense-Process-Send*

5.1.2 Experimental Evaluation

To evaluate the clustering framework, we use the SSPS dataset introduced in Chapter 4. We apply the framework in Figure 5.1 to compute arrival curves using a Δt resolution of $1e6(nsecs)$ and maximum trace interval of $2.5e6(nsecs)$ before splitting into ε which corresponds to a given process name, QNX class, and QNX event combined. The SAX method translates the proximity curve $\mathbb{P}$ of C_{min} to C_{max} using settings similar to the ones discussed in Example 4. We obtain a distance matrix that represents the Levenshtein distance between the SAX strings which is aggregated with the distance matrix that represents the Euclidean distance between the slope-based metric — the steadiness slope $\bar{S}$ in this case — using a weighted sum. Applying the hierarchical clustering method, we obtain the dendrogram in Figure 5.4 which correctly clusters the different modes of operations since there exists a separation that can produce three clusters, each having a single mode of operation. The y-axis of the dendrogram shows the aggregate distance as a percentage of the string length.

The experiment shows a standalone application for the proposed clustering framework. For anomaly detection purposes, the clustering approach might be sufficient to cluster normal traces against anomalous traces for less challenging datasets. However, for more challenging anomaly detection tasks, we now demonstrate the capability of the proposed anomaly detection framework for capturing behavioral anomalies within the same mode of operation.

In this case, the clustering framework can serve as a preprocessing engine that enables building mode-specific arrival curves models for classification purposes. We demonstrate how classification results can significantly improve by splitting the arrival curves models by modes of operation.

The SSPS dataset involves anomalous behavior such as device misconfiguration and performance degradation. By comparing different hardware setups executing a given workload or under different configuration, we can evaluate whether clustering the modes of operation would improve anomaly detection results. In a deployed system, an attacker would substitute a storage media device or alter the system configuration which should be sufficient for our anomaly detection platform to flag the anomaly by inspecting the logs using abstract arrival curves. As mentioned, the SSPS dataset naturally provides different modes of operations, i.e., *sense, process*, and *send*. Within a trace, a marker event indicates the start of each mode of operation which acts as the ground truth for evaluation.

	Hardware Variants: BeagleBone vs SabreLite	
	Trace Count: (Train, Normal, Anomalous)	(TPR,FPR)
Without Clustering	(132,67,69)	(100%,3%)
Sense Cluster	(39,23,23)	(100%,0%)
Process Cluster	(46,22,24)	(100%,7%)
Send Cluster	(47,22,22)	(100%,0%)

Table 5.1: Effect of Clustering on Classification: Hardware Variants

	Misconfiguration: *hd0* 2500 vs *random* 5000	
	Trace Count: (Train, Normal, Anomalous)	(TPR,FPR)
Without Clustering	(57,55,55)	(85%,45%)
Sense Cluster	(14,13,15)	(100%,0%)
Process Cluster	(21,20,20)	(100%,14%)
Send Cluster	(22,22,20)	(100%,0%)

Table 5.2: Effect of Clustering on Classification: Hardware Misconfiguration

Table 5.1 and Table 5.2 shows the results of the classifcation when splitting the traces based on the modes of operation obtained using the framework in Figure 5.1. The tables show the results of the classification obtained in Chapter 4 versus the results using the clustering engine which

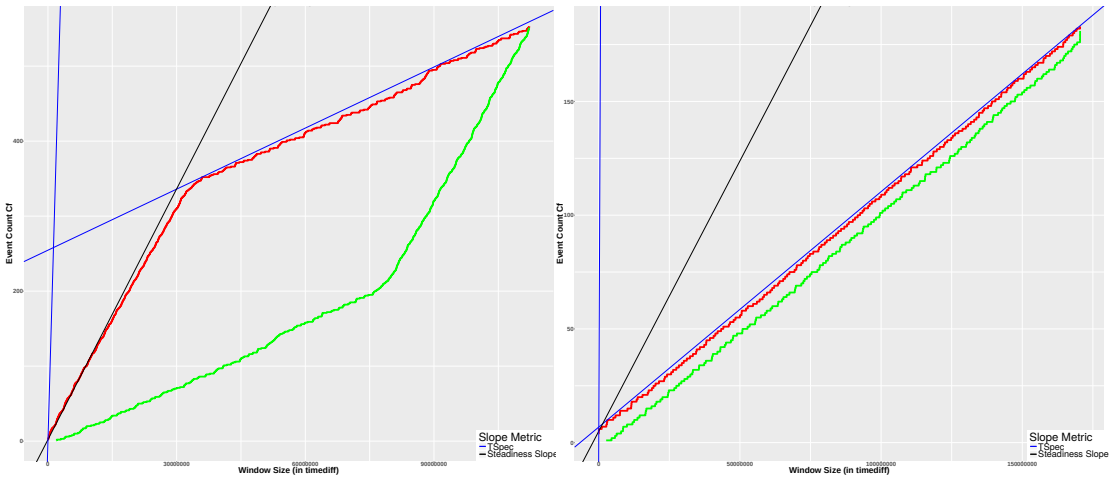
clearly demonstrates its benefit. We significantly improves the classification results, i.e., achieving perfect TPR $=(100\%)$ and lower FPR for all scenarios for both datasets, i.e., 14% and 0% instead of 45%. The reasoning behind such an improvement arises from the fact that a generic model that considers a wide range of behaviors originating from different operational modes is less capable to detect the slightest anomalous deviations to the normal model. In other words, a model with a wide confidence interval shown in Figure 4.2 would consider deviations from the mean curve to be always normal, however, a mode-specific model will have a confidence interval that is capable of picking up those slight deviations.

Semantics of Different Slope Metrics

The evaluation in this section applies the steadiness slope metric $\bar{S}$ for distance computation, however, similar classification results were achieved using the widely known $(burst, rate)$-model and *TSpec*-models. The nature of the anomalies introduced in the SSPS dataset had similar effects on the different slopes metrics visualized in Figure 5.3. In this thesis, the chosen affine curve models introduce an upper bound (over-approximation) on the arrival curve C_{max} under the assumption that the average rate of C_{max} can be linearly approximated by some rate r with the choice of an initial affine curve that passes through the origin.

However, the SSPS evaluation platform presented cases for empirical arrival curves C_{max} whose long-term behaviour can be represented by multiple slopes. As a result, the representation using an affine curve model whose initial segment passes through the origin and a single long-term rate introduces a significant error that is not suited for anomaly detection purposes. We visualize examples for these cases in Figure 5.5.

Figure 5.5a shows a limitation to the variant of *TSpec*-model used in the thesis. Although the long-term rate can be defined accurately, the initial burst introduced an over-approximation error. Choosing a different *TSpec*-model to better represent the bursty arrival behaviour is possible, yet, an algorithm to automate such decision-making procedure would be necessary. On the contrary, Figure 5.5b showcases another C_{max} that is accurately represented by the employed *TSpec*-model while an unacceptable approximation error is introduced by the steadiness slope $\bar{S}$.



(a) Traditional arrival curve metrics limitation (b) Steadiness slope metric $\bar{S}$ limitation

Figure 5.5: Comparing different slope metrics

5.1.3 Clustering Beyond Modes of Operations

We further demonstrate the benefit of the clustering engine, by presenting two examples for the introduced case-studies:

Clustering Anomalous Traces in the UAV Case Study

The first example targets the use of the proposed clustering approach for an easier task, which is to cluster anomalous traces versus normal traces. The original approach already got optimal classification results for this task. Figure 5.7 shows the clustering results from the hexacopter dataset. As expected, the clustering results separate the normal and anomalous traces where the height of the dendrogram shows the distance between them. The anomalous traces are the ones having *full-while, half-while, fifo-ls* terms in the file name. The traces having terms *baseline, clean, faultinj* show similar behavior with respect to each other.

Clustering Configuration Settings

The second example uses the traces that correspond to the same mode of operation from different configurations to see whether the approach can classify them as well. Figure 5.6 shows the configuration of using different storage media *hd0 vs random* where the clustering shows perfect separation as indicated by the file names.

5.1.4 Discussion

The clustering approach presented in this section completes the building blocks of the framework introduced in Chapter 4. The clustering engine serves as an optional block that would improve the classification accuracy when separate models are needed to separate normal scenarios. The extension to clustering using SAX demonstrate the applicability of arrival curves as characterization features to describe event traces that, in our case, originates from real-time systems.

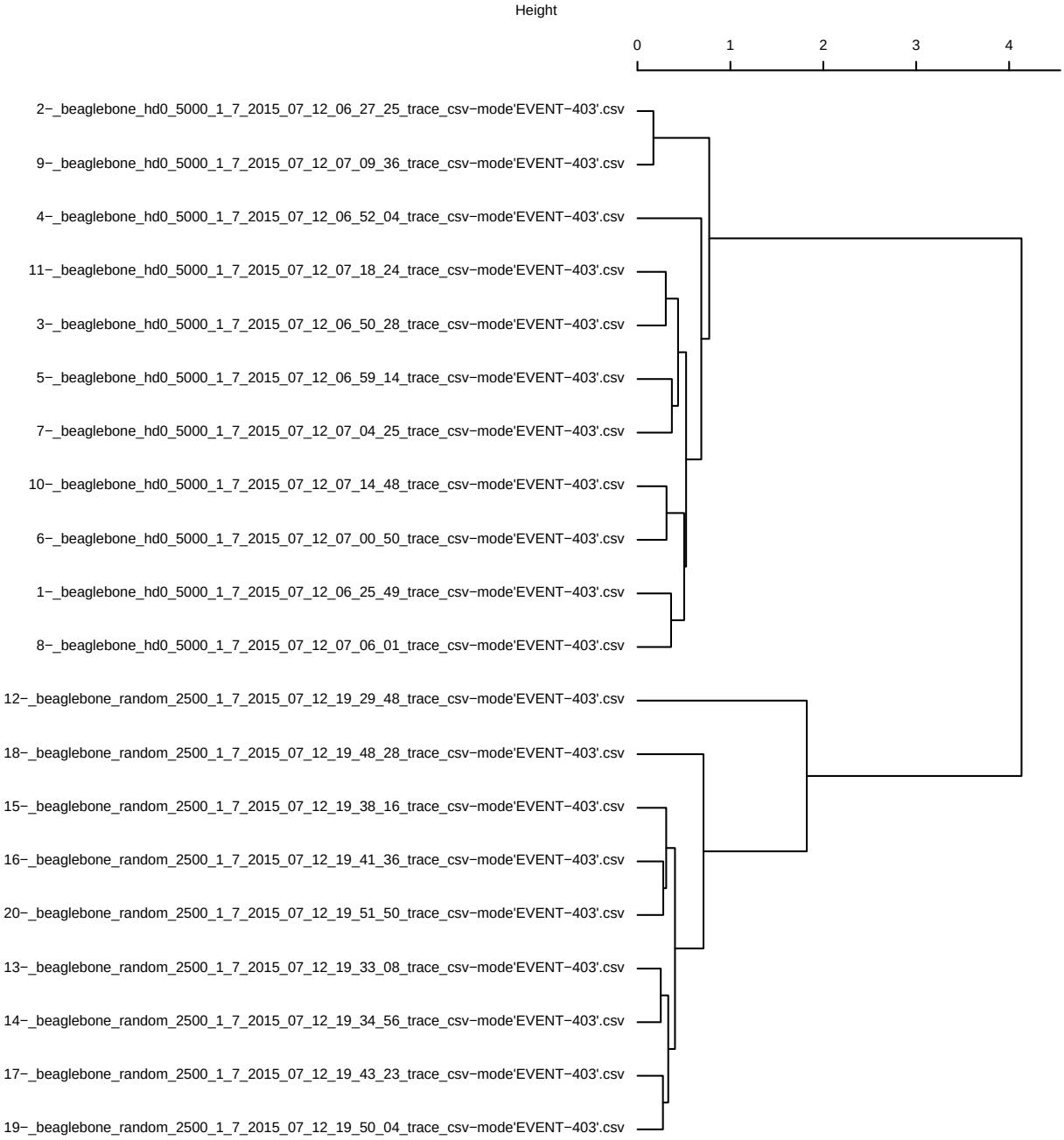


Figure 5.6: Dendrogram of SSPS *hd0* vs random scenario clustering

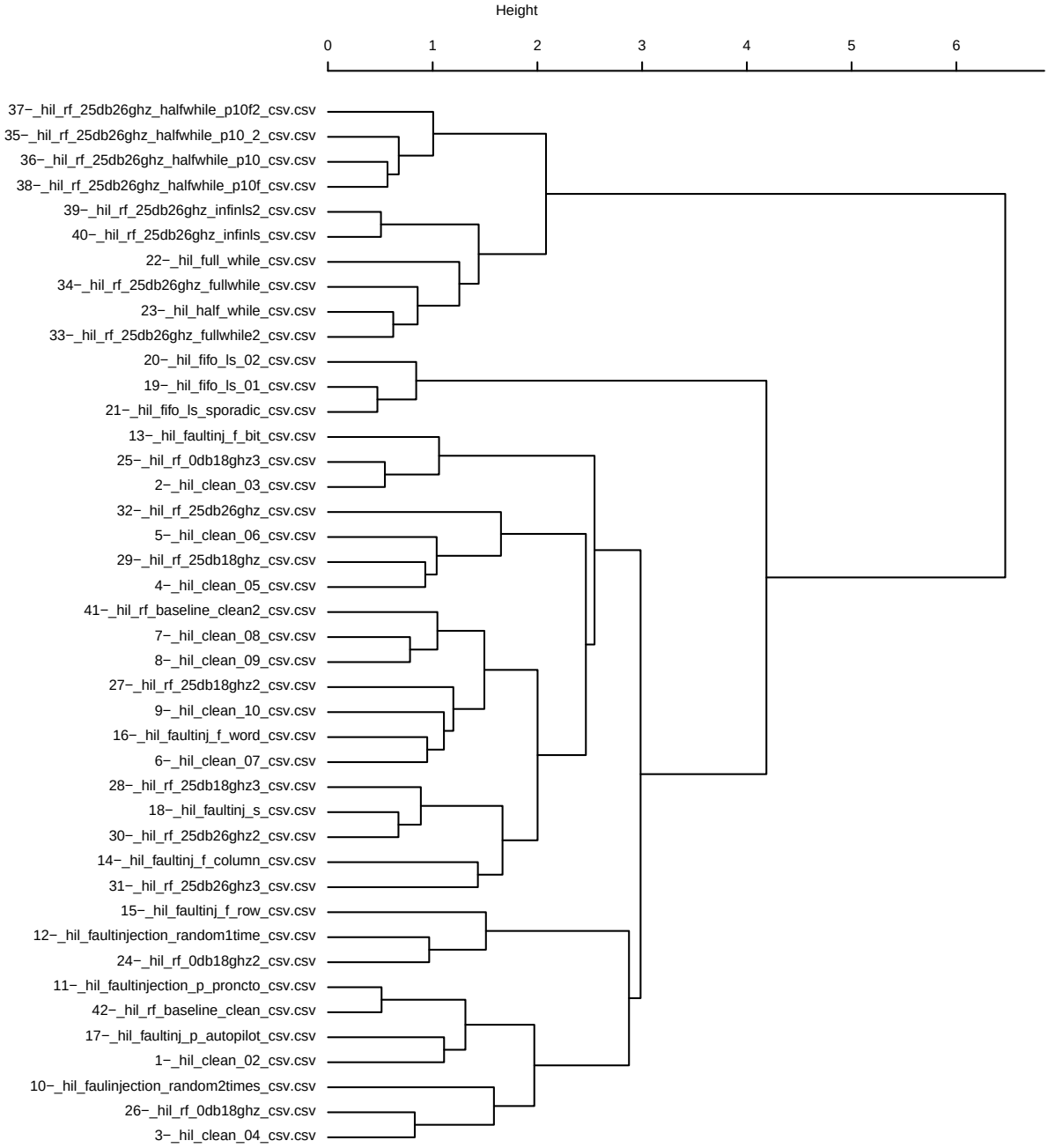


Figure 5.7: Dendrogram of UAV clustering results

5.2 TRACMIN in PALISADE Streaming Framework

Trace mining for online anomaly detection has become of recent research interest. One main reason is the availability of minimally intrusive hardware for tracing like [56]. The traces are generated as part of the normal operation of such devices making it suitable for inspecting unpredictable failures. These traces can be streamed into files or over the network when the target devices operate in remote locations. Interestingly, streaming the trace content over the network in online fashion eliminates the need for the storage required to keep the entire trace of an operating session that might reach terabytes of data making it an overwhelming process to retrieve that information at once for offline analysis which might delay the corrective action in case of an anomalous behavior. As a result, the problem of having an efficient online anomaly detection technique on the list of the current research problems [19].

We presented an offline anomaly detection framework that builds an offline model using entire traces for post-mortem analysis on complete traces. In this section, we deploy arrival curves in a streaming anomaly detection framework. The streaming approach faces two main challenges with respect to offline anomaly detection:

Single-Pass Requirement

Streaming anomaly detection frameworks require a single-pass approach for data processing. In other words, the data will be discarded after processing. As a result, the computational approach of the online anomaly detection technique affects the analysis efficiency. For example, consider an online anomaly detector using a non-overlapping sliding window, the processing of an input stream should be at least as fast as the generation of data having a length equal to the defined window length. In other words, data within the sliding window should be processed before another window of data replaces it. The requirement can be relaxed depending on the available memory.

Parameters Suited for Online Analysis

Based on our prior offline anomaly detector in Chapter 4, the primary parameter we need to choose for an online anomaly detection is Δ_{max}. The parameter controls the maximum number of events or time units considered in the sliding window for observing the maximum and minimum occurrence count. The objective is to minimize Δ_{max} in a way that achieves an acceptable ROC curve [25] as this would allow for lower computation time. However, for a streaming approach, it is desirable to have smaller Δ_{max} basically to minimize the stream size needed to accurately detect faults or anomalies on-the-fly.

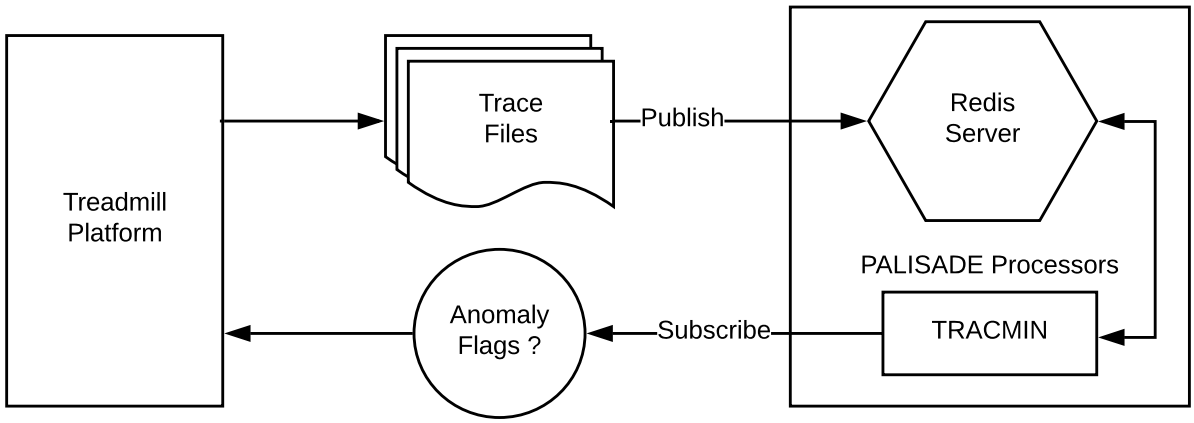


Figure 5.8: TRACMIN integrated in PALISADE streaming framework

5.2.1 Modifying TRACMIN for Streaming Anomaly Detection

We tackle the problem of online anomaly detection using traces generated by real-time systems in a streaming fashion. Although the problem looks similar to the traditional problem of mining traces for anomaly detection, there are two main differences: First, the entire trace is not available for inspection where the inspected data is not stored after analysis in a typical case. Second, online anomaly detection requires having fast monitoring tools that enable corrective actions during the execution of real-time systems. Typically, the analysis time by the online monitors should be negligible compared to the generation rate of data streams. In other words, the monitor should handle the streaming trace events in a bounded time with respect to the generation rate or amount of data processed.

The offline anomaly detection approach discussed in Chapter 4 provides a confidence interval on both the maximum and the minimum arrival curves, upon which the classification is performed by evaluating the proximity of a new curve to those regions. The intuition behind this approach is that traces of approximately same length would exhibit a behavior that can be described using similar minimum and maximum curves, however for streaming traces, this approach is not valid, as the only guarantee that can be obtained is that some stream of a trace would generate counts that are between the minimum and maximum arrival curves described by the model. However, further streams could push the counts towards the minimum and maximum curves as an entire trace streaming is performed. As a result, the evaluation method used in Chapter 4 for offline anomaly detection is not applicable to the streaming approach.

For streaming anomaly detection approach, we define the normal behavior by the region

bounded by the upper confidence interval of the maximum arrival curve and the lower confidence interval of the minimum arrival curve. The arrival curves constructed over the streaming data is considered normal as it falls in this region or within some defined proximity threshold from those two boundary curves. Figure 5.11 shows a snapshot of normal arrives curves that meet this criterion. We discuss the details of the snapshot later in this section.

5.2.2 Overview on PALISADE

PALISADE is a streaming anomaly detection framework for embedded systems developed by the Real-time Embedded Software Group at the University of Waterloo. The framework uses Redis in-memory database where the different processor nodes use a publish-subscribe interface on Redis channels to receive data from a database and output anomaly flags in addition to relevant information, i.e., anomaly scores.

Figure 5.8 shows a subset of the PALISADE framework to demonstrate how TRACMIN can be integrated. The objective for the framework is to facilitate the streaming of traces collected from real-time systems to evaluate the capabilities of different anomaly detection techniques simultaneously. The framework contains a data source which can be a database, a trace file, or a system running online. The Redis channels are the pathway for the data to the publishers and subscribers, where the processor node receives the data for processing and output the anomaly flags and relevant information on the channels. We evaluate TRACMIN for streaming anomaly detection on a case study that is integrated as a data source for PALISADE.

5.2.3 Integrating TRACMIN with PALISADE

The objective is to integrate TRACMIN as a processor node in PALISADE, and as a result, be able to process the streaming events in the framework. As discussed, PALISADE provides commands to load and save an offline model, and since TRACMIN already perform the anomaly detection using offline models. The models, in this case, will be an arrival curve model similar to the models discussed in Chapter 4. PALISADE then monitors the events streamed to specific channels, and as a result, registering TRACMIN as one those channels, the data can be streamed for processing from the data source to the processor node.

At this stage, the arrival curves are constructed using data buffered in the processor node and compared to the loaded model. The result of the anomaly detection is streamed on the corresponding channels. In this case, it will be an anomaly flag corresponding to the behavior of the data within the buffer, in addition to any scores that quantify the classification results. We

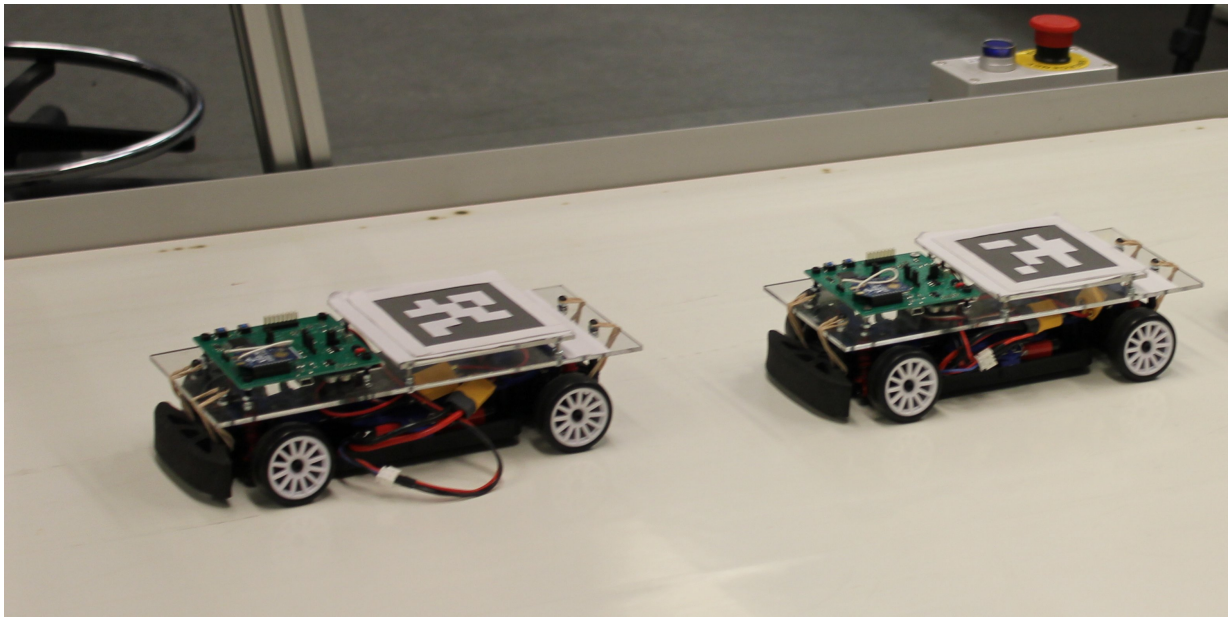


Figure 5.9: Demonstrator conveyor belt with two remotely-controlled cars

highlight that the objective of the anomaly detection approach for TRACMIN is to flag anomalous traces after processing an entire trace. However, for the streaming approach, the scope of the detection is limited to the buffered data from the stream.

5.2.4 Evaluation: Treadmill Case Study

The Treadmill platform is a conveyor belt that simulates an infinite path for remotely controlled cars. The cars are tracked by a top mounted camera where an electric motor controls the conveyor system using throttle set points sent over Ethernet from a PC. Custom circuit boards connected to the cars through XBee interface receives the throttle and steering commands from a base station in Robot Operating System (ROS) format. The traces of interest are the timestamped throttle values. Figure 5.9 shows the platform during operation using two cars. A normal behavior describes two cars moving steadily with a fixed distance separating them. An anomaly is injecting a change to the controller settings attempting to crash the front car from behind.

The objective of our evaluation is to detect the attack as the car in the back starts to attempt to crash the one in the front. A set of traces that describe the normal behavior of the two cars, using which we create a normal arrival curves model as discussed in Chapter 4. In deployment, the throttle commands are streamed to Redis as discussed in Figure 5.8 and the TRACMIN processor

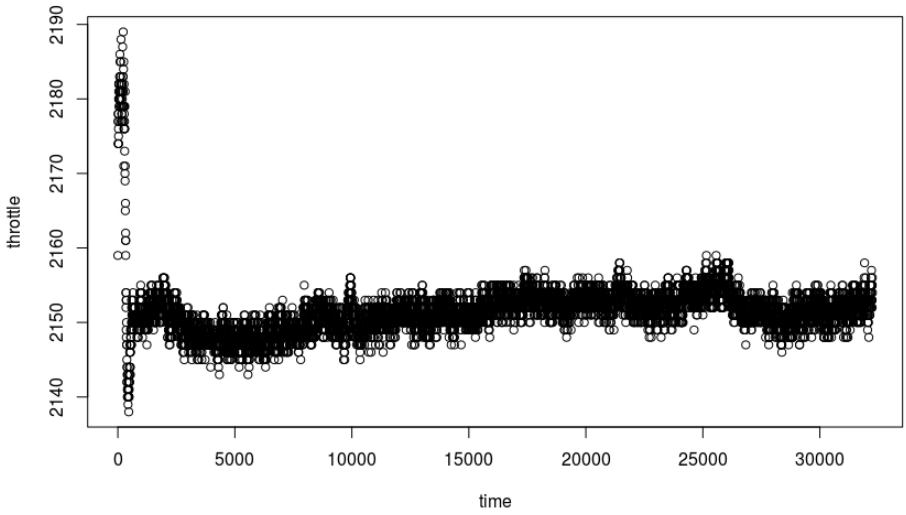


Figure 5.10: ROS commands for car throttle

nodes construct the arrival curves on-the-fly and assess the curves against the offline model. Figure 5.10 shows a sample of throttle commands plot versus time, while Figure 5.11 shows how an arrival curve can model the arrival of those throttle command versus a normal model obtained offline. The figure shows how the assessment in the online demo differs from the classification procedure in Chapter 4 where the shaded region bound between the C^{+}_{max} and C^{-}_{min} is considered the normal region.

5.2.5 Discussion

In this section, we show how arrival curves can be used as features to describe streams of events in addition to entire traces processed offline as discussed in the previous chapters. Online trace mining is a topic that is gaining research interest as Internet-of-Things and cloud computing start to prevail. With the efficient implementation and parameter consideration in building training model, the classification of streams allows for pinpointing anomalous sub-traces on-the-fly during system execution. Combined with post-mortem analysis, the online and offline approaches of using arrival curves in TRACMIN provide valuable insights to the domain expert when analyzing failures and monitoring real-time systems.

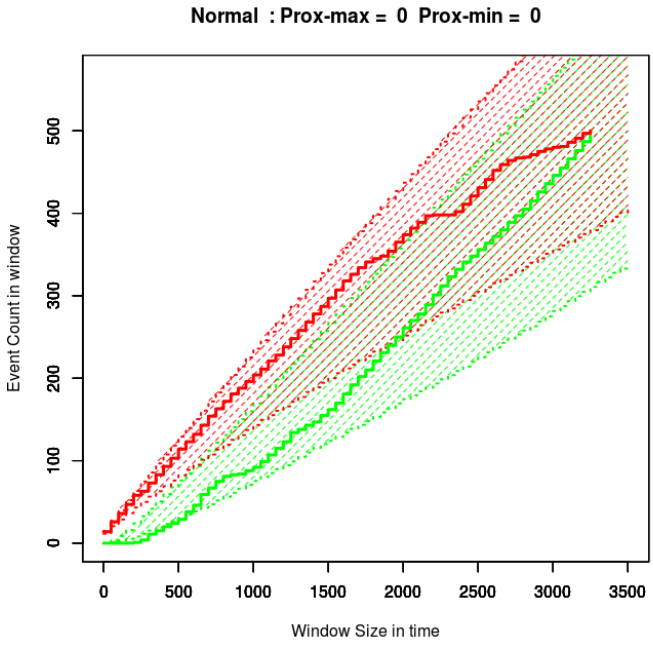


Figure 5.11: Arrival-curves model for throttle command

5.3 Recurrent Pattern Mining using Arrival Curves

Real-time systems can have periodic, aperiodic, and sporadic behavior. The term *recurrent* refers to a behavior that mixes between periodic and sporadic where we expect the traces collected from such systems to represent a repeating behavior within some flexible timing intervals [68]. Specifically for real-time systems, an analyst would require knowing the best-case and the worst-case execution behavior that can be considered normal. Using this information, one can detect the deviation to raise flags of anomalous behavior. Our work uses arrival curves as the main approach for targeting our research problems because of the ability to represent such best and worst cases defined by system execution, not the analytical bounds.

The detection of recurrence period is crucial for abstracting the behavior of a process using empirical arrival curves, as the arrival curves described by window sizes up to the recurrence period can be sufficient for the analysis as we discuss in Chapter 6 for robustness assessment. The recurrence period can indicate the frequency of the repeating pattern observed by event generation, for example, detecting core switching when considering traces originating from a specific CPU core.

5.3.1 Approach

To reason about the existence of recurrent behavior of a real-time system, our approach aims to extract intervals of repeating modes of operation and characterize the arrival behavior 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 approximately same C_{diff} or C_{max}–C_{min} 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. For example, 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 5.1 shows how to calculate that variance σ of distances in this case:

$$
\begin{aligned}
&\zeta_\Delta = \{\Delta' | C_{\text{diff}}(\Delta', T) \simeq 0 \wedge \Delta' \in \mathbb{N}_{>0}\} \\
&\text{diff}(\zeta_\Delta) = \\
&\quad \{\delta | \delta = \Delta'_j - \Delta'_i, \Delta'_j = \inf\{\zeta_\Delta > \Delta'_i\} \text{for all} \Delta'_i \in \zeta_\Delta\} \\
&\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \text{ with } x = \text{diff}(\zeta_\Delta), \mu \text{ as mean of } x
\end{aligned} \tag{5.1}
$$

Following this conjecture, we employ the *auto-correlation function (acf)* [11] to reason about the existence of recurrent patterns within some C_{diff} curve. Auto-correlation is widely used in signal processing to detect repeating patterns [44]. 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 [61].

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 similar values of C_{diff} at distances $\simeq \Delta_p$ of low variance σ is expected to yield an autocorrelation curve of statistically significant values separated by distances $\simeq \Delta_p$. We use the autocorrelation function *acf* implemented in R [72] which uses the definitions from [76]. The plots from the mentioned function provides a confidence interval based on uncorrelated series which can be used to determine the existence of patterns.

Example 5. *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 5.12a and Figure 5.12b 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 $RPT1$

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 $RPT1$. For $RPT2$, there exists 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 behavior, we randomly introduce events to disturb the patterns $RPT1$ and $RPT2$. As a result, the auto-correlation result shown in Figure 5.12a for the third trace did not have statistically significant values, i.e., $NORPT$. The significance is indicated by the horizontal confidence intervals around ± 0.2, where having autocorrelation values only within these bounds indicate that C_{diff} shows no correlation versus increasing lag [61]. 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.

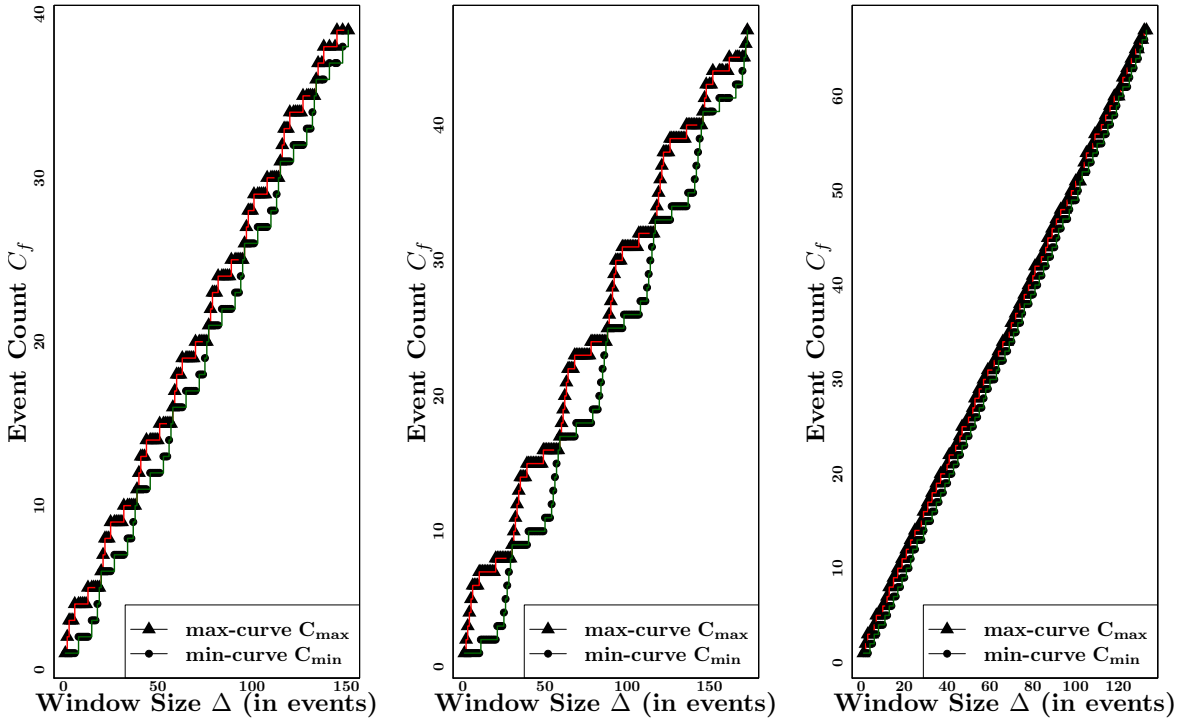
5.3.2 Evaluation: UAV Case Study

The results obtained using the synthetic trace of Example 5 shows how arrival curves are potentially good features for describing recurrent behavior within event traces. We show how this approach can be applied to event traces generated from deployed real-time systems.

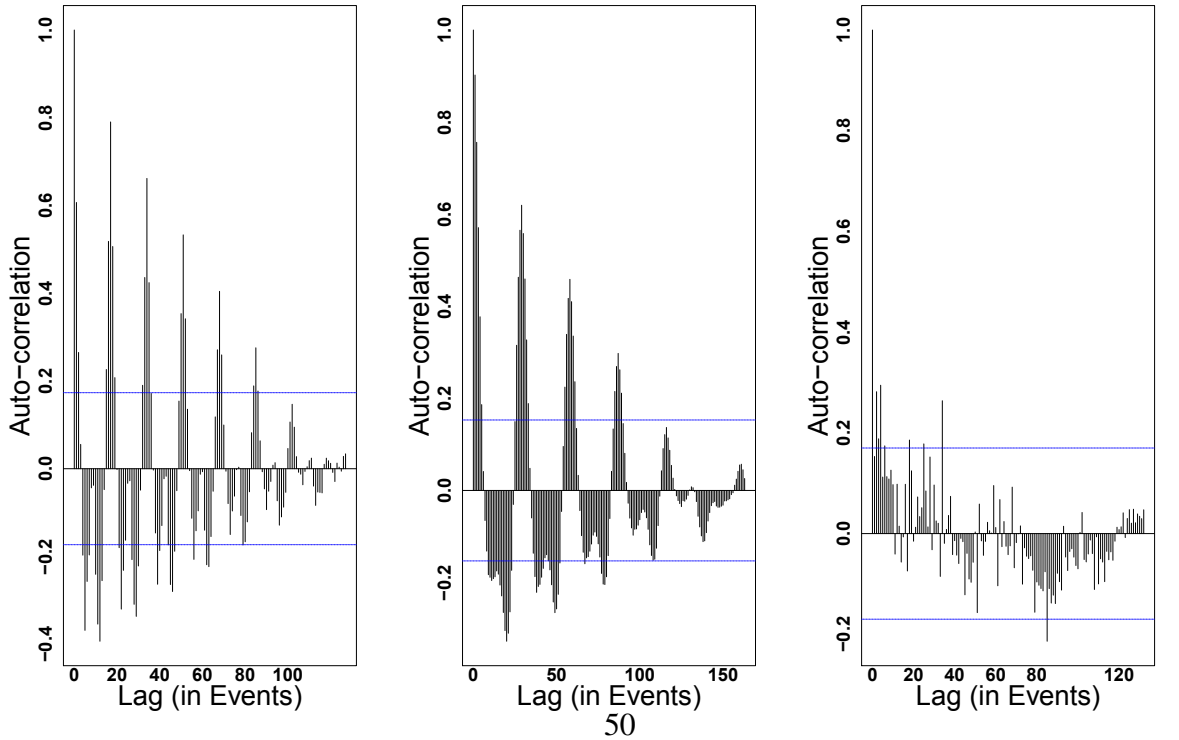
Indexed Traces

The aim of this case study is to validate our conjecture on traces known to exhibit recurrent behavior. As the UAV case study uses QNX event traces generated by a real-time system, we expect the traces to show signs of recurrent behavior. 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 behavior in the generated traces. The *fifo-ls* scenario shows similar numbers of event types having recurrent behavior; however, such behavior occurred at different intervals Δ_p. The *full-while* scenario shows on average of 7 event types to have recurrent behavior meaning that several events lost the recurrent behavior due to the highly utilized CPU behavior affecting the behavioral patterns of the system. The number of events that showed recurrent behavior in both case studies highlights the applicability of using arrival curves for the purpose of characterizing the recurrent behavior 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.

Figure 5.13 provides a sample of the result of autocorrelation function applied to the C_{diff} curves obtained from the case study. The curve corresponds to a QNX event `THREAD_THREPLY`



(a) $RPT1$, $RPT2$, $NORPT$ patterns



(b) Autocorrelation for C_{diff} arrival curves of $RPT1$, $RPT2$, $NORPT$ patterns

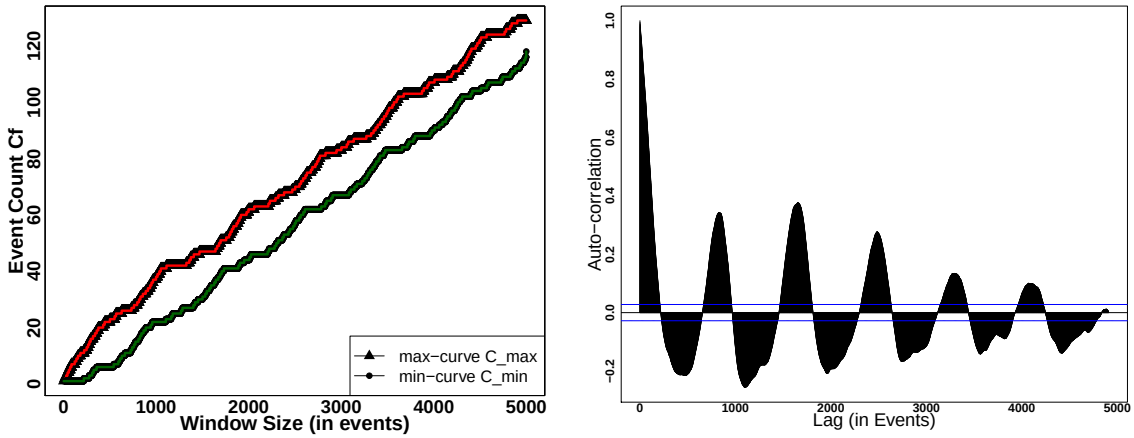


Figure 5.13: Mining *UAV* recurrent pattern [indexed traces]

and shows the recurrent behavior within the generated trace that can be characterized using the recurrence windows Δ_p events.

Timestamped Traces

Now we use timing information to demonstrate the generality of our approach to both traces with and without timing. In this case, we show event the describes the kernel event associated with communication message reply within the operating system. Figure 5.14 shows the behavior of the specified event within the UAV case study along with the autocorrelation obtained that indicates a recurrence interval.

As we mentioned earlier in this chapter, it is important to know the least window size needed for streaming purposes. The section discussed a way to quantify that size through ROC curves by inspecting the smallest window size that yields good enough classification result. Another method to achieve this is through recurrent pattern mining approach described in this section. Using the recurrence period detected by the autocorrelation method, we can strict the analysis to that recurrence pattern.

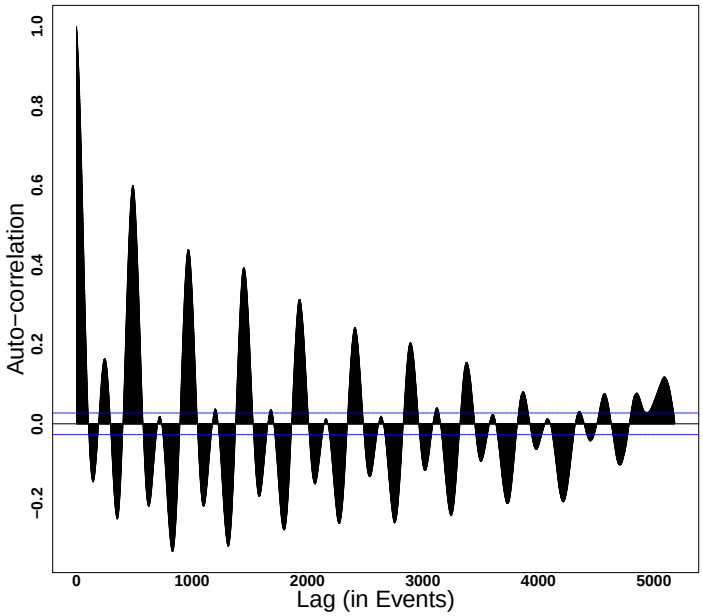


Figure 5.14: Mining *UAV* recurrent pattern [timestamped traces]

5.3.3 Discussion

The presented approach for recurrent pattern mining using arrival curves extends the applications to better characterize the recurrent behavior of real-time systems that they often exhibit for the purposes of better understanding to the systems. Also, the absence of recurrent behavior or the change in recurrence period can be also used for anomaly detection.

5.4 Conclusion

We conclude this chapter by emphasizing on the possibilities expanded by defining empirical arrival curves for real-time systems in Chapter 3. This chapter demonstrated the arrival curves can be used in clustering traces originating from different modes of operations or anomalous behaviors. We also showed how the TRACMIN framework can be extended to process the traces in a streaming fashion, demonstrated by the successful integration with the PALISADE streaming platform, exploiting the efficient implementation constructing arrival curves and the ability to detect recurrent patterns. The recurrent patterns in a real-time system can be detected using empirical arrival curves when combined with the concepts of autocorrelation applied to the defined metrics and definitions.

Chapter 6

Robustness Evaluation of Arrival Curves Model

We introduced empirical arrival curves and tackled the thesis statement by evaluating their feasibility for characterizing real-time systems having recurrent behavior. However, one problem with empirical models is the lack of guarantees to their performance after deployment. In this chapter, we evaluate the robustness of models based on arrival curves in order to provide bounds on the variation captured by the statistical bounds discussed in Chapter 4.

6.1 Motivation

One main challenge to empirical models constructed from tracing data is how they are evaluated. For example, surveys [14, 59, 86] categorize the evaluation of empirical models used for anomaly detection either by their ability to classify normal versus anomalous behavior or by measuring the time needed to construct a model for a given behavior. Concerning classification accuracy, the current research work shows a lack in the methods that derive robustness bounds on the acceptable behavior of a given system using margins provided by the empirical models. In this context, the authors in [37] acknowledge the problem that empirical models for anomaly detection are generally tuned in an ad-hoc manner without guidance by well-found theoretical framework or analysis. As a result, the authors claim that there are no guarantees on the effectiveness of the empirical models after deployment.

To overcome this problem, we aim to assess the robustness of arrival-curves models used to characterize the ranges of tolerable behavioral variations of a given real-time system, i.e.,

hardware degradation, external attack, etc. The general problem addressed can be stated as: *Given an arrival-curve model that represents the normal behavior of a real-time system, how much behavioral variation the underlying system would exhibit before the generated traces can no longer be represented using such model*

6.2 Problem Formulation and Approach

In the previous chapters, we show the feasibility of arrival curves as high-level features in a trace mining approach in modeling the recurrent behavior of a real-time system through computing arrival curves from event traces. We hypothesize that an empirical arrival-curves model which incorporates statistical bounds for acceptable variation in events arrival within a trace can be represented by the demand-bound function of some task load under a defined scheduler with a specified demand variation. Representing the statistical boundaries obtained from the empirical models as some variation of a given demand-bound function enables the use of well-established mathematical foundations to derive bounds on the variation of task parameters that can be yet accepted by such deviation in the demand-bound function.

Under this hypothesis, the general problem statement transforms to the following: *Given the demand-bound function (dbf) for sporadic task-set having implicit deadlines scheduled by an EDF scheduler with associated upper and lower bounds for allowed variations in the dbf, obtain a range of values for the task parameters (period, execution time) such that the system stays operational within the provided demand boundary*. We now discuss the mentioned task model.

6.3 Overview on Demand-Bound Functions for the Assumed Task Model

In this section, we review some basic definitions and present the assumed task model that provides the basis for our theoretical analysis.

Definition 7. *A task $T_K(p, e, d)$ is a dispatchable entity in the system where the period p is the number of time units between successive dispatches, e is the execution time (in time units) required to complete the work, and the deadline d is the maximum time available to complete the work after dispatching.*

The established mathematical foundations for the demand-bound functions motivates the theoretical analysis. A demand-bound function (dbf) models the maximum processor demand by

a task over any interval of length t [8]. The dbf of a given sporadic task under EDF assumption is defined as:

$$\mathrm{dbf}(t) = \left\lfloor \frac{t + p - d}{p} \right\rfloor e \tag{6.1}$$

We will also consider that the sporadic task has an implicit deadline ($d = p$) and there are no overloads, restricting the possible values of e to the range $]0, p]$, with $p \in \mathbb{R}$.

$$\mathrm{dbf}(t) = \left\lfloor \frac{t}{p} \right\rfloor e \tag{6.2}$$

Due to the empirical nature of the target arrival-curves model, the purpose of the chosen task model is to provide a reasonable approximation to the arrival curve that describes an increasing events count versus an increasing sliding window interval as we discuss in Chapter 7. Hence, we choose the specified sporadic task model with implicit deadline under EDF scheduler which yields an increasing step-wise function that steps e units every p time units. From the properties of step-wise functions [39], this increasing function can be approximated with a straight line with slope $\frac{e}{p}$. We evaluate the choice of this task model and the arrival curves approximation in Section 6.6.

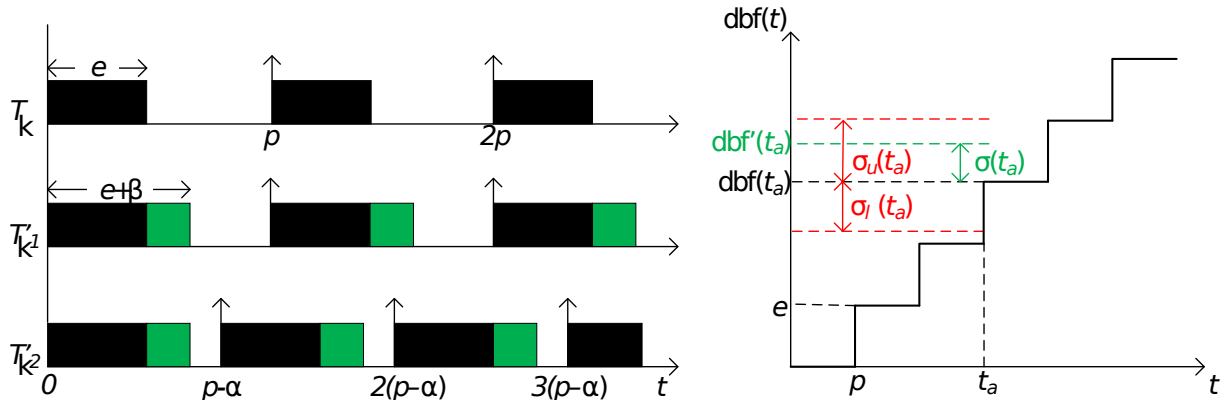


Figure 6.1: Graphical representation of variations in the nominal task parameters and dbf

Variations in the nominal task parameters can either increase or decrease the task demand. In practical settings, changes in the task parameters may arise from changing operational conditions. For example, activating a low-power system mode may increment the period of a given task,

while adapting to aging hardware might result in an increment of the execution time of a task. We formalize the range of possible values of the altered task parameters in terms of decrements in the nominal period and increments in the nominal execution time as follows:

Definition 8. *Decreasing the period of a task.* $\boldsymbol{\alpha}$ *is defined as the reduction of the task period* p *in time units, therefore* $\alpha \in (-\infty,\ p\ [$.

Definition 9. *Increasing the execution time of a task.* $\boldsymbol{\beta}$ *is defined as the increase of the task execution time* e *in time units, therefore* $\beta \in]\ -e,\ (p-\alpha)-e\ [$.

We define α as a decrement and β as an increment for mathematical convenience. But to generalize our analysis, we highlight that both Definition 8 and Definition 9 allow negative values for both α and β, i.e., increase of task period and decrease of task execution time, respectively.

Considering the definitions for α and β, we now introduce general model for an altered task $T'_K(p-\alpha, e+\beta)$, which incorporates the variations in period and execution time from Definitions 8 and 9 while maintaining the condition of implicit deadlines but for the altered period in this case, i.e., $(d-\alpha = p-\alpha)$. We can obtain a corresponding altered dbf as follows:

$$\mathrm{dbf}'(t) = \left\lfloor \frac{t}{p-\alpha} \right\rfloor (e+\beta) \tag{6.3}$$

Let us now consider that for each interval length t, we define arbitrary bounds on allowed variations in the nominal dbf from Equation 6.2 (with $\alpha = \beta = 0$), restricting the valid values of dbf' for a given application.

Definition 10. *Variation Bound on Task Demand. We denote the allowed variations of the dbf at time interval* t *as* $\sigma(t) = \mathrm{dbf}'(t) - \mathrm{dbf}(t)$*, where* $\sigma(t) \in [\sigma_l(t), \sigma_u(t)]$*, and* $\sigma_l(t), \sigma_u(t) \in \mathbb{R}$.

The restriction in the allowed values of $\sigma(t)$ can be either set by the system designer according to some specific operational requirement or can represent some uncertainty in the specifications. Note that Definition 10 permits describing deviations above and below the nominal demand. This is a key difference of our analysis with respect to related work on sensitivity analysis from the scheduling domain [69, 87, 94–96], which focuses on verifying that the demand stays below a certain limit such that the system remains schedulable. We contrast our work with sensitivity analysis in Section 6.7.

Figure 6.1 illustrates the previous definitions for the variations in the nominal task parameters and the corresponding dbf. The diagram on the left shows the timeline for the execution of a

task T_K with period p and execution time e, and also the execution of tasks T'_{K^1} and T'_{K^2} that include variations in the nominal parameters. In specific, T'_{K^1} increments the nominal execution time by β time units (represented in the shaded green box), and T'_{K^2} aggregates a reduction in the period. Both T'_{K^1} and T'_{K^2} generate a demand above the nominal value. The diagram to the right shows the step-wise nominal dbf together with the terms defined earlier for allowed variations at a certain point t_a. In this case, the demand of the altered task $\mathrm{dbf}'(t_a)$ is above the nominal value, but within the specified boundaries of allowed variations $\sigma_l(t_a)$ and $\sigma_u(t_a)$.

Considering the previous definitions, the problem of interest can be tackled by finding the region of allowed values of α and β, such that the value of $\sigma(t)$, representing the deviation in the demand of the altered task $\mathrm{dbf}'(t)$ with respect to the nominal demand $\mathrm{dbf}(t)$ stays within the predefined range $[\sigma_l(t), \sigma_u(t)]$. We acknowledge that under the assumed task model presented in this section, utilization-based approaches might be the typical solution to evaluate timing properties of a given real-time system. However, we use the demand-bound function as we hypothesize its possible abstraction to empirical arrival curves as we demonstrate in Section 6.6.

6.4 Computing Bounds on Task Alteration

In this section, we relate the demand deviation bound to a feasibility region for the parameters α and β of the altered task model T'_K. The mathematical foundations assume a specified demand variation bound for a given task. However, the analysis presented in this section can be directly extended to specified demand variation bounds for multiple independent tasks, i.e., a task T_{K_i} has a specified demand variation bound σ_i where $\sum_i \sigma_i = \sigma$. Such problem breaks down into multiple sub-problems that can be solved by finding the feasible region for each α_i and β_i for each task T_{K_i} separately.

Substituting Equation 6.2 and Equation 6.3 in Definition 10, we can derive a relationship between α and β values that alter a nominal task model T_K while meeting a deviation demand $\sigma(t_a)$ for a given time interval t_a as follows:

$$\beta = \frac{\sigma(t_a) - \left(\left\lfloor \frac{t_a}{p-\alpha} \right\rfloor - \left\lfloor \frac{t_a}{p} \right\rfloor \right) e}{\left\lfloor \frac{t_a}{p-\alpha} \right\rfloor} \tag{6.4}$$

The allowed deviation from the nominal dbf will be bounded by $[\sigma_l(t_a), \sigma_u(t_a)]$. By replacing $\sigma(t_a)$ by $\sigma_l(t_a)$ in Equation 6.4, we can establish a relationship between a lower bound for the

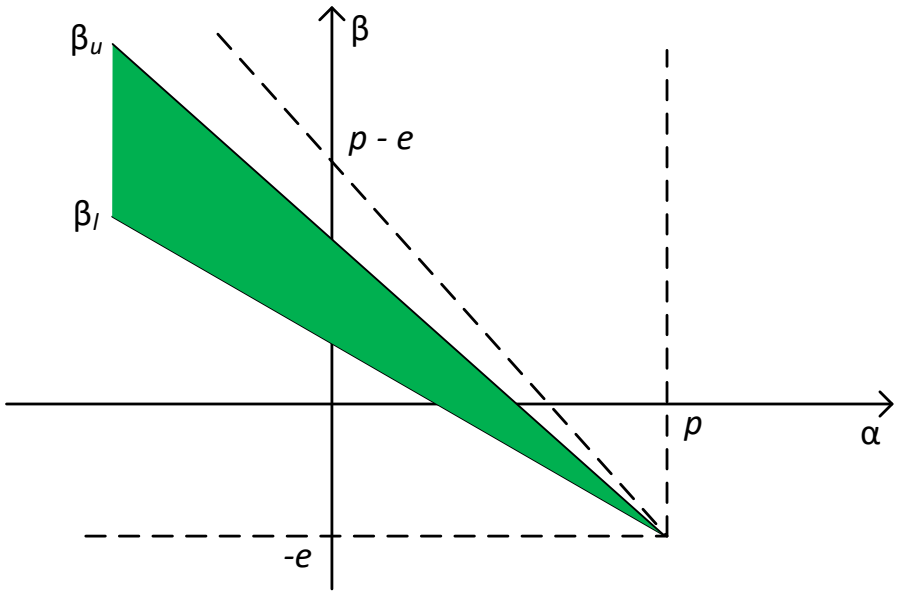


Figure 6.2: Illustration of the feasibility region for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

parameter β_l, and the possible values of α. In a similar manner, we can replace $\sigma(t_a)$ by $\sigma_u(t_a)$ to obtain the upper bound β_u.

Figure 6.2 illustrates how we can use the relationships described above to obtain a feasibility region for the values of α and β given a certain $\sigma(t_a)$. The figure shows a plot of β as a function of α, in addition to the resulting β_l and β_u. The figure also includes dashed lines to illustrate the valid intervals for α and β according to Definition 8 and Definition 9, respectively. The lines for β_l and β_u intersect at the point $(\alpha, \beta) = (p, -e)$. Considering the limits β_l and β_u and the restrictions over the parameters, we can obtain a feasibility region (shown in shaded green) for the valid combinations of α and β that will allow to keep the altered demand within predefined boundaries.

To further illustrate the theoretical foundations, we present the following example with concrete task parameters that we will use throughout the rest of the analysis.

Example 6. *Consider a sporadic task with parameters* $e = 0.375$ *and* $p = d = 0.5$. *Find the feasibility region for* α *and* β *such that the demand of the altered task at* $t_a = 30$ *remains within a range of* $\pm 10\%$ *of the nominal demand as shown in Figure 6.3.*

Figure 6.4 shows the computed upper bound β_u and lower bound β_l with respect to valid values for α by applying Equation 6.4 to the demand of the task described in Example 6. We

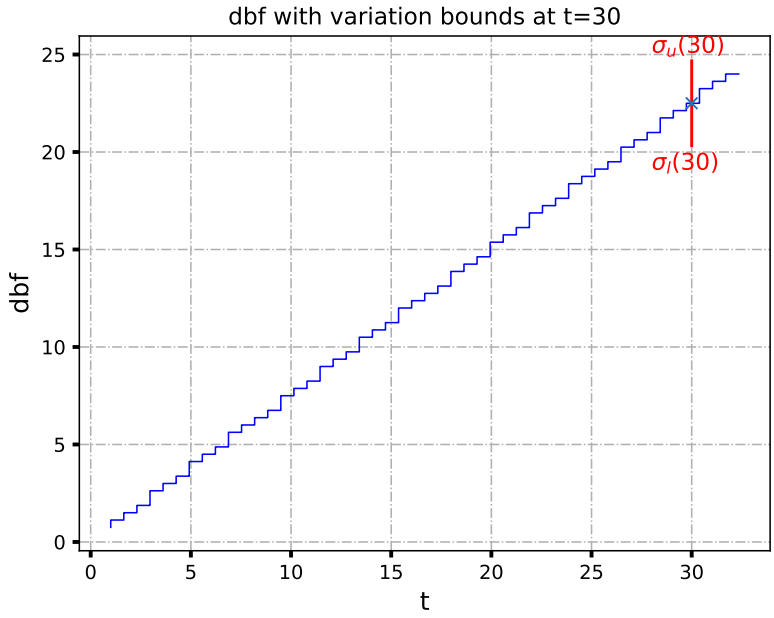


Figure 6.3: $\mathrm{dbf}(t)$, $\sigma_u(t_a)$, and $\sigma_l(t_a)$ of Example 6

restrict the lower bound of α to $-p$, so the range of allowed α values according to Definition 8 changes to $[-p,\ p]$. The resulting feasibility region for the variations in parameters is shown in shaded green. When drawing a vertical for a given value of α, any value of β within that region will ensure that the resulting demand from the altered system will remain within the specified variations.

Section 6.4 defined the feasibility region for α and β by defining a specific time interval of variation t_a. We analyze the effect of increasing time intervals t on the variation of task parameters α and β in Section 6.5.

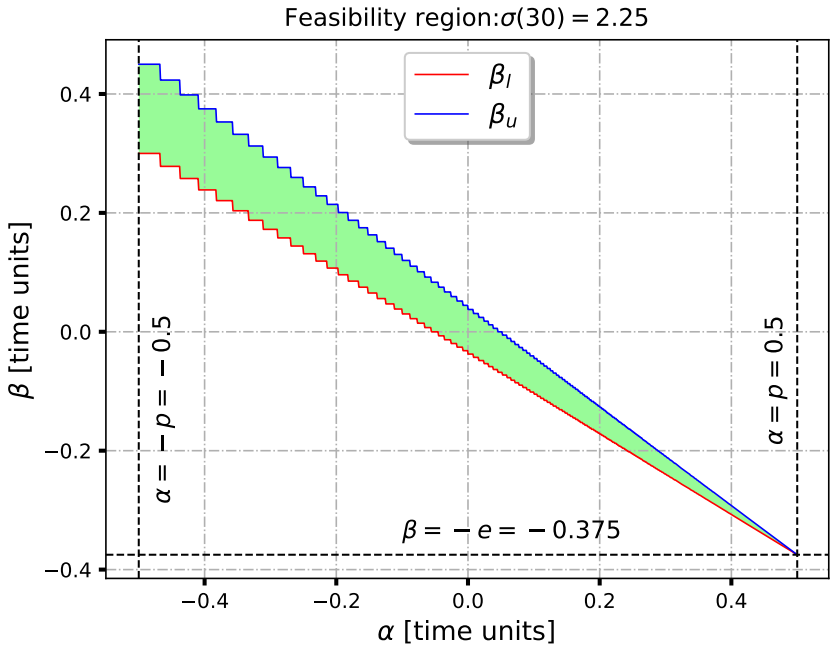


Figure 6.4: Task parameters feasibility region for Example 6

6.5 Asymptotic Analysis for Task Parameters Variation

The analysis in this section shows how variations in parameters α and β change over increasing time intervals t to meet the specified demand bounds. To achieve this, we use Equation 6.4 which defines the relation between β and time interval t for a given α. Similarly, for a given β, the equation defines the relation between α and time interval t. Analyzing the change of β and α as the time interval t increases gives us an insight about the change of permissible system parameters alteration over different time intervals.

We consider the cases where the bounds to demand variation σ are defined both as constant values and as relative values with respect to the nominal demand $\mathrm{dbf}(t)$. In practice, a real-time system designer will specify a requirement mandating that a given task should not exceed its demand by some amount of time units at any time interval. For this case, we study when the variation of the demand is constrained by a given range $[\sigma_l, \sigma_u]$, where σ_l and σ_u are constant values that remain unchanged for all time intervals t. Alternatively, a designer could choose a demand variation bound that changes with the time interval t. In this case, the variation of the demand is constrained by a given range $[\sigma_l, \sigma_u]$ that changes over t, i.e., deviations in period and execution time are relative to the nominal demand at time interval t.

6.5.1 Asymptotic Analysis for Variation in Execution Time β

To compute the asymptotic bounds of β_u and β_l, we apply a transformation to Equation 6.4 using some approximations that are valid for asymptotic values of time intervals t.

First, we relate the decrease in period α to the period p using a variable k, where $\alpha = k \times p$ such that $k \in (-\infty, 1[$. In other words, the variable k is a ratio of the decrease in period α with respect to the nominal period p. Second, we relate t to both p and α by defining c where $t \approx c\,(p - \alpha)$ assuming c is some factor much larger than $(p - \alpha)$. Hence, $t \approx c\,p\,(1 - k)$ as well.

We evaluate these approximation as t goes to ∞. To compute the limit of the floor operator, we apply the Squeeze Theorem of Limits [38] which can be used to find $\lim_{x\to\infty} f(x)$ where $f(x)$ is bounded by $g(x)$ and $h(x)$, $g(x) \leq f(x) \leq h(x)$ as follows:

$$\lim_{x\to\infty} g(x) \leq \lim_{x\to\infty} f(x) \leq \lim_{x\to\infty} h(x) \tag{6.5}$$

Applying Equation 6.5 to the definition of floor function, $c - 1 \leq \lfloor c \rfloor < c$, we deduce that $\lim_{c\to+\infty} c - 1 = c$ and $\lim_{c\to+\infty} c = c$, and as a result:

$$\lim_{c\to+\infty} \lfloor c \rfloor = c \tag{6.6}$$

Similarly, since $(1 - k)$ is a constant. We obtain the following result:

$$\lim_{c\to+\infty} \lfloor c\,(1 - k) \rfloor = c\,(1 - k) \tag{6.7}$$

Equation 6.6 and Equation 6.7 allows for transforming Equation 6.4 to obtain β.

Case 1: Constant demand variation bound σ

We now consider the case where σ is a constant value irrespective of the nominal demand dbf and time intervals t. Given that the dbf is an increasing function, a constant σ bound provides a decreasing margin for demand variation as time interval t increases. Consequently, as dbf increases, the σ represents a smaller fraction from dbf, until it can be negligible with respect to the factor c. We can approximate the asymptotic value of β using the following expression:

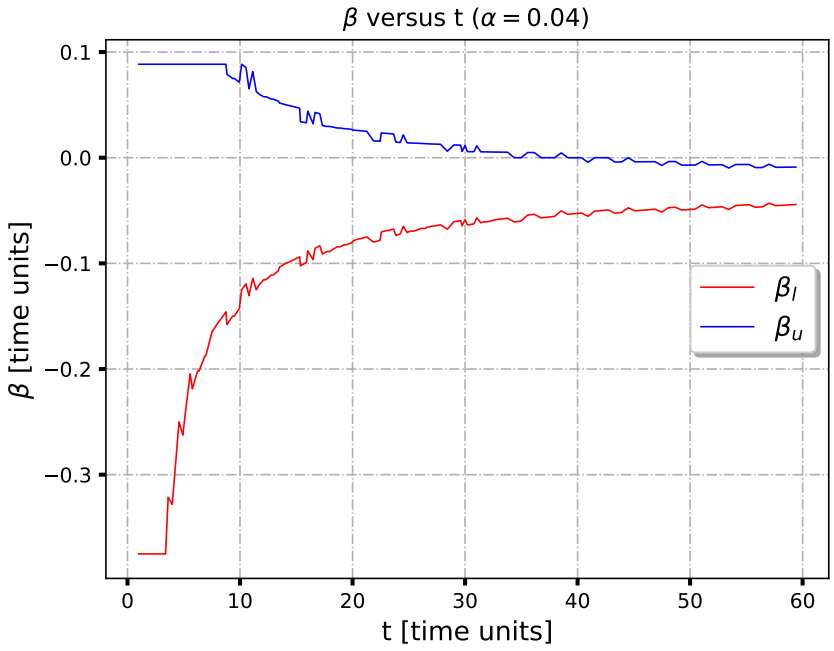


Figure 6.5: Asymptotic analysis of $\boldsymbol{\beta}$ using constant $\boldsymbol{\sigma}$

$$
\begin{aligned}
\lim_{t\to+\infty} \beta(t) &= \lim_{c\to+\infty} \frac{\sigma - \left(\left\lfloor \frac{c(p-\alpha)}{p-\alpha} \right\rfloor - \left\lfloor \frac{cp(1-k)}{p} \right\rfloor\right) e}{\lfloor c \rfloor} \\
&= \lim_{c\to+\infty} \frac{\sigma - \left(\lfloor c \rfloor - \lfloor c(1-k) \rfloor\right) e}{\lfloor c \rfloor} \\
&= \lim_{c\to+\infty} \frac{\sigma - (c - c(1-k))e}{c} \\
&= \left(\lim_{c\to+\infty} \frac{\sigma}{c}\right) - e(1-(1-k)) \\
&= 0 - e\,k \\
&= -e\,k \\
&= -e\,\frac{\alpha}{p}
\end{aligned}
\tag{6.8}
$$

Equation 6.8 shows that the asymptotic value is independent of σ_u and σ_l. In other words, the boundaries β_u and β_l should asymptotically converge to each other as t increases.

One way to confirm this finding is that the altered dbf' can be approximated to the nominal dbf as t increases. As a result, the slope e/p of dbf is equal to the slope $(e+\beta)/(p-\alpha)$, which is the same result as $\beta = -(\alpha/p)\ e$. Defining this asymptotic value for β bounds serves an example for an extreme case scenario where the boundaries do not provide flexibility for a range of permissible β values as t increases indefinitely.

Figure 6.5 shows the feasible β values bounded by β_l and β_u as a function of t, obtained from applying Equation 6.4 to Example 6 with an arbitrary value of $\alpha = 0.04$ and $\sigma_u = -\sigma_l = 2.25$. Using Equation 6.8, it can be shown that the bounds β_u and β_l converges to $-(\alpha/p)\ e = -0.03$.

Case 2: Demand variation bound σ as a function of nominal demand

We consider $\sigma(t)$ values that can be defined relatively to the nominal demand $\mathrm{dbf}(t)$. Let us define σ as a fraction f of the nominal demand $\mathrm{dbf}(t)$. For example, the demand variation bound can be set to be $\pm 10\%$ of the nominal demand at any given time interval t. In this case, to compute the asymptotic values we use Equation 6.9 as follows:

$$\sigma(t) = f\ \mathrm{dbf}(t) = f\ \left\lfloor \frac{t}{p} \right\rfloor e \tag{6.9}$$

Using $\sigma(t)$ from Equation 6.9 to compute the asymptotic values of β can be derived as follows:

$$\begin{aligned}
\lim_{t\to+\infty} \beta(t) &= \lim_{t\to+\infty} \frac{f\left\lfloor \frac{t}{p} \right\rfloor e - \left(\left\lfloor \frac{t}{p-\alpha} \right\rfloor - \left\lfloor \frac{t}{p} \right\rfloor\right) e}{\left\lfloor \frac{t}{p-\alpha} \right\rfloor} \\
&= \lim_{c\to+\infty} \frac{f\left\lfloor \frac{cp(1-k)}{p} \right\rfloor e - \left(\left\lfloor \frac{c(p-\alpha)}{p-\alpha} \right\rfloor - \left\lfloor \frac{cp(1-k)}{p} \right\rfloor\right) e}{\lfloor c \rfloor} \\
&= \lim_{c\to+\infty} \frac{f\lfloor c(1-k) \rfloor\, e - (\lfloor c \rfloor - \lfloor c(1-k) \rfloor)\, e}{\lfloor c \rfloor} \\
&= \lim_{c\to+\infty} \frac{fc(1-k)e - (c - c(1-k))e}{c} \\
&= -ke + f(1-k)e
\end{aligned} \tag{6.10}$$

We observe that the asymptotic value β is a function of f that relates the demand bound to the nominal demand dbf. As a result, the β bounds β_u and β_l will not converge to the same value as

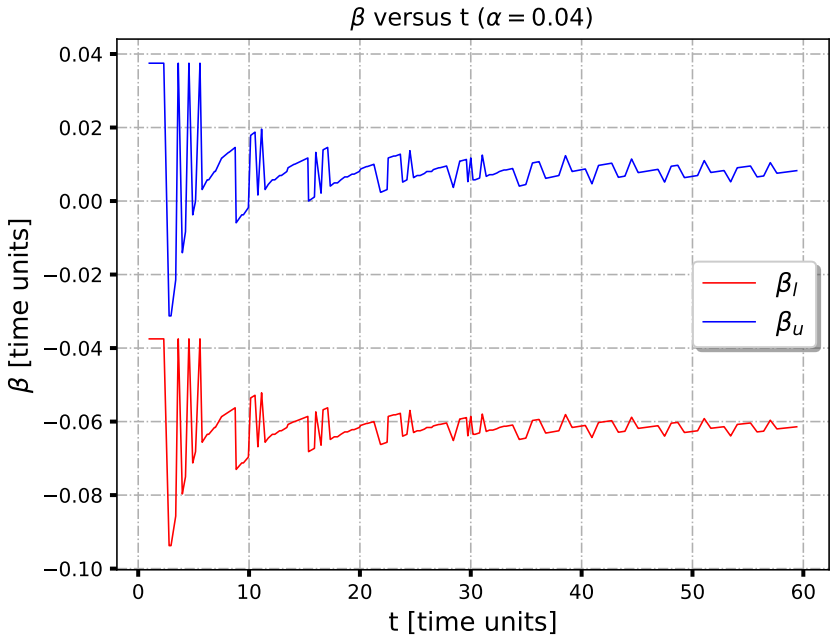


Figure 6.6: Asymptotic analysis of $\boldsymbol{\beta}$ using relative $\boldsymbol{\sigma}$

it was the case for constant σ value. Figure 6.6 shows the boundaries β_u and β_l when defining $\sigma(t) \in [0.9 * \mathrm{dbf}(t), 1.1 * \mathrm{dbf}(t)]$ in Example 6. Using Equation 6.10, we find the asymptotic values for the boundaries are $\beta_l \approx -0.064$ and $\beta_u \approx 0.0045$.

6.5.2 Asymptotic Analysis for Variation in Period α

For a given value of β, we study how α changes over time interval t by obtaining the relation between α and t from Equation 6.4 as follows:

$$Z = \left\lfloor \frac{t}{p - \alpha} \right\rfloor = \frac{\sigma + \left\lfloor \frac{t}{p} \right\rfloor e}{e + \beta} \tag{6.11}$$

Unlike the analysis for the values of β, defining a precise relation between α and t for a given β is not a straightforward operation. Since the inverse of the floor operator is undefined, we cannot obtain a closed formula for α. Instead, we restrict the analysis to obtain conservative bounds for

the range of α values that satisfy Equation 6.11. To do this, we can apply the range property of the floor operator [39], which states the following:

$$\lfloor x \rfloor = m \iff m \leq x < m + 1 \tag{6.12}$$

Using the property in Equation 6.12, we can describe a range for the values of α as:

$$p - \frac{t}{Z} \leq \alpha < p - \frac{t}{Z+1}, \text{ with } Z = \frac{\sigma + \left\lfloor \frac{t}{p} \right\rfloor e}{(e + \beta)} \tag{6.13}$$

Substituting σ for the specified demand variation bounds σ_l and σ_u in the obtained inequality, we can obtain the relation of the corresponding boundaries for α versus time interval t for a given β. Note that each boundary for σ leads to a feasible range of α, so we define Z_u and Z_l, which we obtain replacing σ_u and σ_l in the term Z defined in Equation 6.11, respectively. Substituting Z by Z_u and Z_l in Equation 6.13 yields two inequalities with four boundaries which can be bounded by the α in Equation 6.14. Now, we show the asymptotic values for all four boundaries when the demand variation bound σ is defined as a constant and as a function of nominal demand and we visualize these results in Figure 6.7 and Figure 6.8.

$$p - \frac{t}{Z_l} \leq \alpha < p - \frac{t}{Z_u + 1} \tag{6.14}$$

Case 3: Constant demand variation bound σ

Considering constant values for σ_l and σ_u, we analyze the change of the approximate region of α defined by Equation 6.14 over time interval t for a given β. We apply the same transformations used in Section 6.5.1 as follows:

$$Z = \frac{\sigma + \left\lfloor \frac{t}{p} \right\rfloor e}{e + \beta} \approx \frac{\sigma + c\,(1-k)\,e}{e + \beta} \tag{6.15}$$

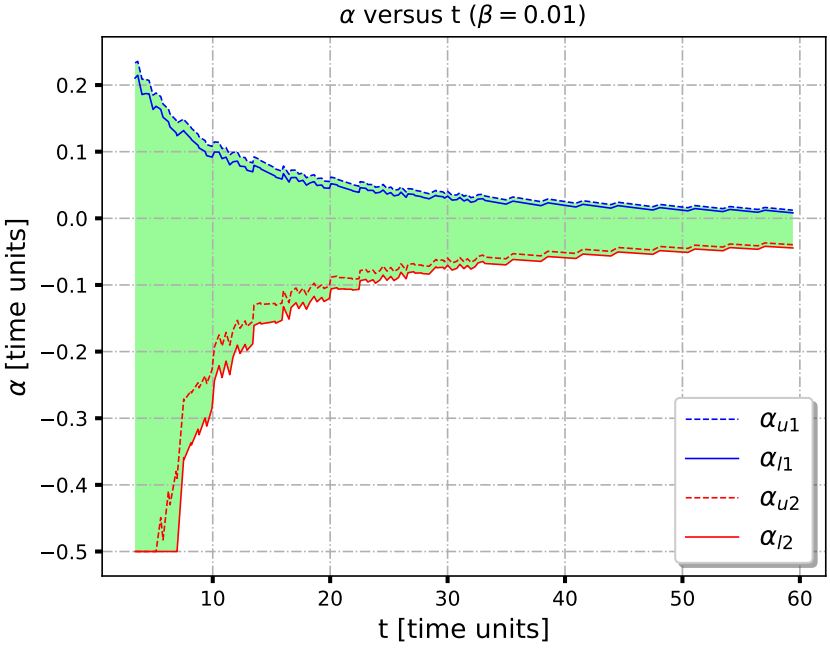


Figure 6.7: Asymptotic analysis of $\boldsymbol{\alpha}$ using constant $\boldsymbol{\sigma}$

As discussed, the constant value of σ becomes negligible to the factor c as t increases. We obtain the asymptotic value of α in the case of constant demand bound by applying Equation 6.15 to the boundaries in Equation 6.14. For the right-hand side of the inequality we can compute the asymptotic value as follows: – Note that we can deduce the asymptotic bound similarly for the left-hand side which will obtain the same result – :

$$p - \frac{t}{Z+1} = p - \frac{c\,p\,(1-k)}{\frac{\sigma + c\,(1-k)\,e}{e+\beta} + 1} = p - \frac{c\,p\,(1-k)(e+\beta)}{\sigma + c\,(1-k)\,e + (\,e+\beta)} \tag{6.16}$$

Taking the limit to the previous equations lead to the asymptotic value for α in our case:

$$\begin{aligned}\lim_{c\to+\infty} p - \frac{c\,p\,(1-k)(\,e+\beta)}{\sigma + c\,(1-k)\,e + (\,e+\beta)} &= \lim_{c\to+\infty} p - \frac{p\,(1-k)(\,e+\beta)}{\frac{\sigma}{c} + (1-k)\,e + \frac{(\,e+\beta)}{c}} \\ &= p - \frac{p\,(\,e+\beta)}{e} \\ &= p - p + \frac{p\,\beta}{e} \\ &= -\beta\left(\frac{p}{e}\right)\end{aligned} \tag{6.17}$$

We observe that the approximate boundaries of α range α_l and α_u converge to the same value irrespective of σ. The asymptotic value for bounds in α serves as an example for an extreme case scenario as time interval t increases indefinitely.

Figure 6.7 shows the feasible α values bounded by α_l and α_u as a function of time intervals t, obtained by applying Equation 6.17 to Example 6 with an arbitrary value of $\beta = 0.01$ and $\sigma_u = -\sigma_l = 2.25$. It can be shown that the bounds α_u and α_l converge to $-\beta\left(\frac{p}{e}\right) = -0.013$.

Case 4: Demand variation bound σ as a function of nominal demand

Similar to the asymptotic analysis in Case 2 for β, we consider $\sigma(t)$ values that are relative to the nominal demand $\mathrm{dbf}(t)$ where the variation of the demand is constrained by a given range $[\sigma_l, \sigma_u]$ that changes over time intervals t. Using the transformation from Equation 6.9, the asymptotic values for α can be computed using the boundaries in Equation 6.13 again as follows starting with the left-hand side in Equation 6.18 then the right-hand size in Equation 6.19:

$$p - \frac{t}{Z} = p - \frac{c\,p\,(1-k)}{\frac{f\;c\;(1-k)\;e + c\;(1-k)\;e}{e\;+\beta}} = p\;- \frac{c\,p\,(1-k)(e\;+\beta)}{c\,(1-k)\,e\,(1+f)} = p\;- \frac{p\,(e\;+\beta)}{e\,(1+\;f)} \tag{6.18}$$

$$p - \frac{t}{Z+1} = p - \frac{cp(1-k)}{\frac{c(1-k)e(1+f)+(e+\beta)}{e+\beta}} = p - \frac{cp(1-k)(e+\beta)}{c(1-k)e(1+f)+(e+\beta)} \tag{6.19}$$

Then taking the limit as t goes to ∞ for Equation 6.18 yields the same equation, however, for Equation 6.19, we obtain the asymptotic value as follows:

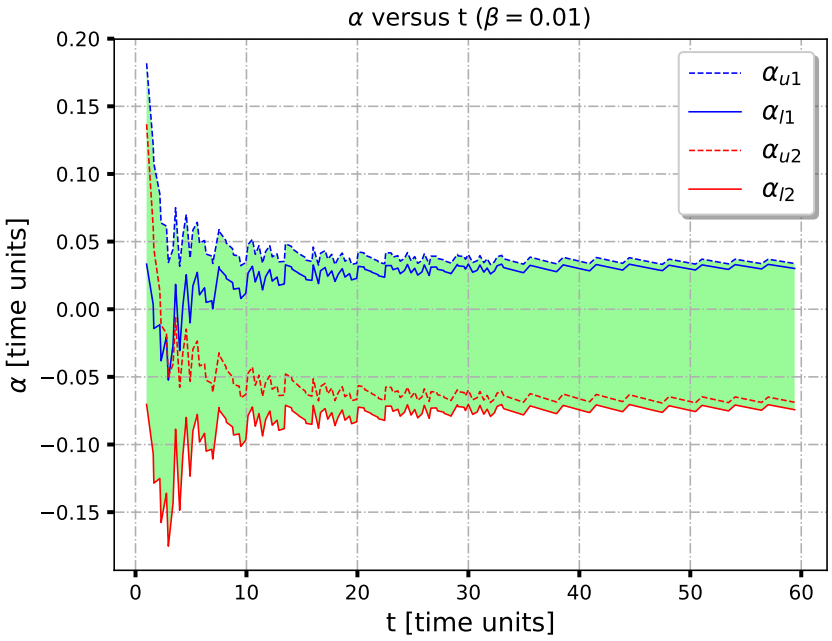


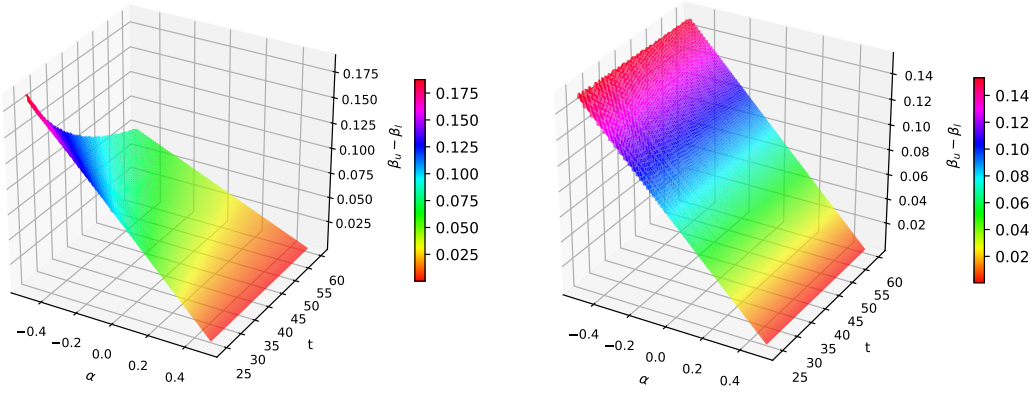
Figure 6.8: Asymptotic analysis of $\boldsymbol{\alpha}$ using relative $\boldsymbol{\sigma}$

$$\lim_{c\to+\infty} \frac{cp(1-k)}{\frac{c(1-k)e(1+f)+(e+\beta)}{e+\beta}} = \lim_{c\to+\infty} \frac{p(1-k)(e+\beta)}{e(1-k)(1+f)+\frac{(e+\beta)}{c}} = \frac{p(1+\beta)}{e(1+f)} \tag{6.20}$$

Thus, we conclude that asymptotically, both sides of the α inequality will converge to the same limit. According to the Squeeze Theorem, α can be defined asymptotically in this case as:

$$\alpha \approx\ p - \frac{p\,(e+\beta)}{e(1+f)} \tag{6.21}$$

From Equation 6.21, we observe that the α asymptotic value is function of f, which relates the demand bound to the nominal demand dbf. As a result, the approximate α bounds α_u and α_l will not converge to the same value as it was the case for constant σ value. Figure 6.8 shows the approximate boundaries α_u and α_l when defining $\sigma(t) = \pm 10\%\,\mathrm{dbf}(t)$ in Example 6. Using Equation 6.21, we find that the asymptotic values for the boundaries are $\alpha_l \approx -0.07$ and $\alpha_u \approx 0.03$ can be computed.



(a) $\beta_u - \beta_l$ vs α vs time t for constant $\boldsymbol{\sigma}$

(b) $\boldsymbol{\beta_u - \beta_l}$ vs $\boldsymbol{\alpha}$ vs time t for relative $\boldsymbol{\sigma}$

Figure 6.9: $\boldsymbol{\beta_u - \beta_l}$ vs $\boldsymbol{\alpha}$ vs time t

Figure 6.9 presents an asymptotic analysis for the feasibility region of α and β values for Example 6. Figure 6.9a shows the analysis for σ with a constant value, and Figure 6.9b shows the analysis for σ with a relative value to the nominal demand dbf, in this case $\pm 10\% of\ \mathrm{dbf}(t)$. The figures show a plane that defines the difference between β_u and β_l corresponding to valid α range as time interval t increases. Both Figures 6.9a and 6.9b visualize the inversely proportional relation between β and α discussed in Section 6.4. However, Figure 6.9a shows the observation from Equation 6.8 where the boundaries β_l and β_u converge to a fixed value, i.e., the difference between β_l and β_u converges to zero. On the contrary, for the case where σ bounds are defined relative to the nominal demand $\mathrm{dbf}(t)$, Figure 6.9b shows that the difference between the boundaries β_u and β_l converges to a constant value as defined by Equation 6.10.

6.6 Robustness Assessment for Arrival-Curves Models

In Chapter 4, we used arrival curves through statistical learning approaches that process execution traces to obtain upper and lower bounds for the arrival of trace events while the system executes normally under different operational conditions. The normal model is typically used in anomaly detection assessing the compliance of arrival curves obtained from new traces to specified bounds.

Now, we evaluate the hypothesis that an arrival curve can be represented as linear demand-bound functions of the assumed task model, and as a result, we perform robustness evaluation for the arrival-curves models using the presented theoretical foundations. Figure 6.10 summarizes the

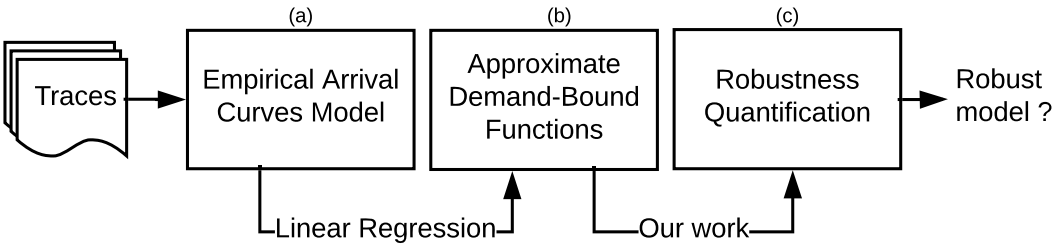


Figure 6.10: Summary of application framework

building blocks of the proposed framework.

The procedure for robustness assessment for the empirical arrival curves follows these steps: a) abstract an arrival-curves model to a task model as in Section 6.3, b) obtain the relation that describes the feasibility region of the allowed alteration for the task model which corresponds to a variation in the arrival behavior shown by the empirical model as in Section 6.4, c) evaluate the approach feasibility by quantifying the effect of approximating the curves to a linear demand-bound function of a sporadic task under an EDF scheduler which we demonstrate in this section.

6.6.1 Representing Arrival Curves as Demand-Bound Functions

In Section 6.3, the mathematical foundation assumes a task model of a sporadic task using the dbf under an EDF scheduler. The result of this assumption is obtaining a step-wise function for the demand, which we can approximate by a line passing through the origin. Similarly, an empirical arrival curve representing a maximum count is a function whose non-decreasing curve starts at the origin [78]. We introduce the methodology that relates both the arrival curve and the demand-bound function.

We apply Linear Regression [62] to obtain the line that best fits an empirical arrival curve. We offset the fitted line to pass by the origin, and as a result, it can be analogous to a demand-bound function. Later in this section, we quantify the negligible error introduced by this process. For example, Figure 6.11b shows the fitted regression lines for the mean arrival curve and the two confidence interval curves after being offset to pass by the origin. The linearity of the curves makes them a good pick for our demonstration.

The regression lines in Figure 6.11b are now analogous to a nominal $\mathrm{dbf}(t)$ with a upper and lower variation bound to the demand σ_l and σ_u respectively which are both functions of t. In other

words, we can define a sporadic task which demands e execution time units every p time interval whose demand-bound function can be approximated by an empirical arrival curve counting a maximum of e instances of an event in a trace every sliding window of p time units.

6.6.2 Robustness Assessment using Task Alteration Parameters

In order to enable the analysis presented in the work on demand-bound functions to the robustness assessment of empirical arrival curves, we use the task model assumed in Section 6.3 to map the task parameters and its variations to the slopes of the regression lines obtained in Figure 6.11b. We denote these slopes as, S for the slope of the regression line for the mean curve, S_u, and S_l for the slopes of the regression lines for both confidence interval curves. We compute these slopes as follows:

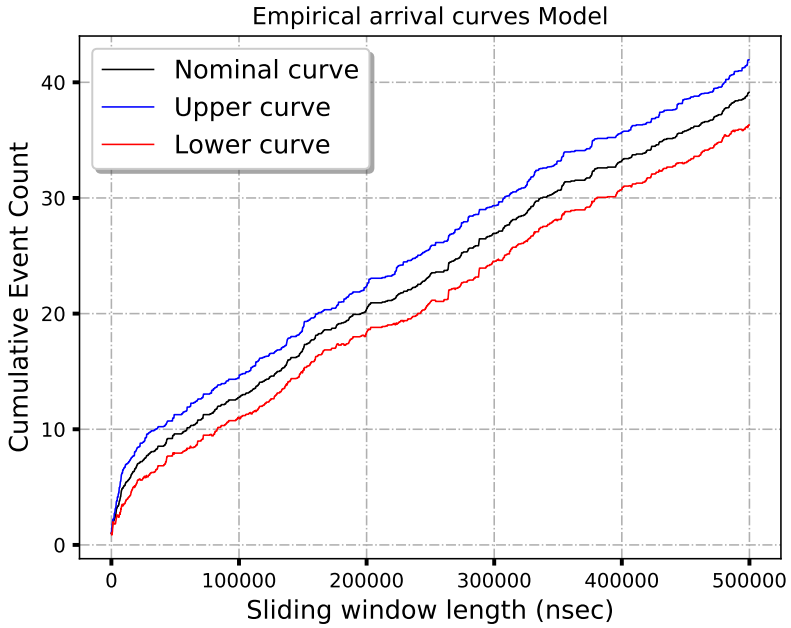
$$S = \frac{e}{p}, \quad S_u = \frac{e + \beta_{\mathrm{u}}}{p - \alpha_{\mathrm{u}}}, \quad S_l = \frac{e + \beta_{\mathrm{l}}}{p - \alpha_{\mathrm{l}}} \tag{6.22}$$

Equation 6.22 defines the relation between task parameters e and p and the regression slopes. We obtain the relations between the variation of parameters α and β from Equation 6.4 using the definitions of S_u and S_l as follows:

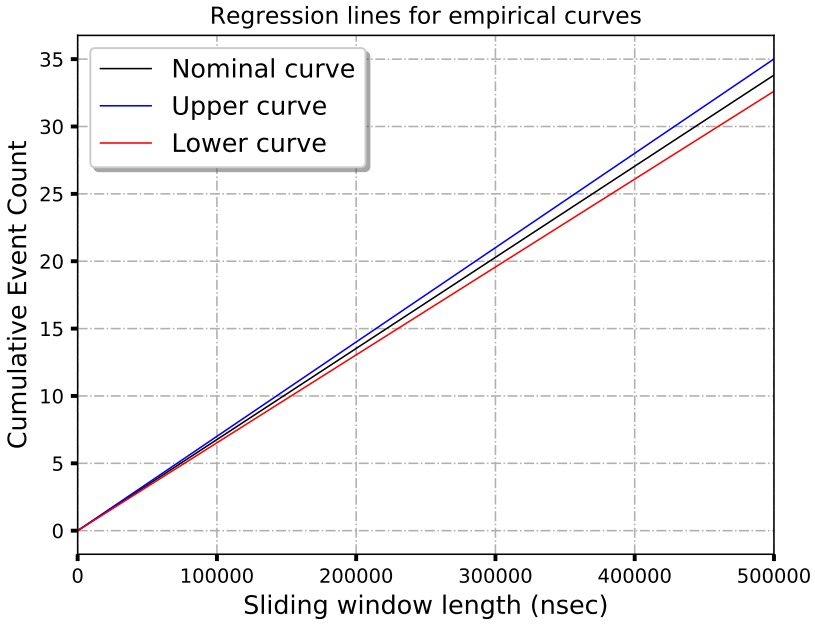
$$\beta_u = \frac{\sigma_u - \left(\left\lfloor \frac{t_a\, S_u}{e + \beta_u} \right\rfloor - \left\lfloor \frac{t_a\, S}{e} \right\rfloor \right) e}{\left\lfloor \frac{t_a\, S_u}{e + \beta_u} \right\rfloor} \tag{6.23}$$

$$\beta_l = \frac{\sigma_l - \left(\left\lfloor \frac{t_a\, S_l}{e + \beta_l} \right\rfloor - \left\lfloor \frac{t_a\, S}{e} \right\rfloor \right) e}{\left\lfloor \frac{t_a\, S_l}{e + \beta_l} \right\rfloor} \tag{6.24}$$

The above equations provide the relation between bounds on β versus execution time e. To define the relation between α and the period p, we substitute e by $S \times p$ from Equation 6.22:



(a) Empirical arrival-curves model for a QNX event



(b) Corresponding regression lines for the model

Figure 6.11: Fitting empirical arrival-curves model to demand-bound functions

$$\alpha_u = p - \frac{e + \beta_u}{S_u}, \quad \alpha_l = p - \frac{e + \beta_l}{S_l} \tag{6.25}$$

The above set of equations provide the relations between the parameters e and p, and the corresponding alterations β and α. The relations evaluates the alteration that would cause a deviation σ to the dbfs obtained by approximating the fitted regression lines of the empirical arrival-curves model. Analogously, β and α now describe the permissible variation to the arrival-curves model, i.e., the count of events of the corresponding sliding window interval of observance.

Now we provide an example to demonstrate how to use these relations to assess the robustness of a model for a real-time system. We exploit the proposed approach by obtaining the feasibility region for the permissible task parameter variations of the mapped task model through the approximation of the empirical arrival curves to demand-bound functions.

6.6.3 Evaluation on UAV Dataset

As discussed in Chapter 4, the UAV dataset traces are generated by a cyber-physical system running a real-time operating system QNX Neutrino 6.4. The traces are collected using the tracing facility *tracelogger*. A trace entry is a timestamped kernel event that shows the type of an event generated while running a specific process on a single CPU core. For the example in Figure 6.11a, we represent an arrival-curves model for a specific QNX event *THREAD THRUNNING* that marks every start of a thread execution for a specified process *proc/boot/procnto-instr*.

To evaluate the robustness of the example model in Figure 6.11a, we perform the following:

Compute Regression Slopes. We obtain the slopes of the fitted regression lines for the mean arrival curve and its confidence interval. It is advisable to assess the adjusted R squared of the regression model. The metric measures the goodness of the linear fit to evaluate whether the assumption that the model is linear was a valid [62].

In our example, the slopes of the lines in Figure 6.11b can be obtained as $S = 6.76 \times 10^{-5}$, $S_u = 7.01 \times 10^{-5}$, and $S_l = 6.52 \times 10^{-5}$. The adjusted R squared is 98% indicating a good linear fit.

Choose Task Parameters. Next step is specify the task parameters e and p in order to obtain the relation between α and β from Equation 6.23, Equation 6.24, and Equation 6.25. However, the provided empirical arrival-curves model cannot be used to obtain the e and p values. This comes from the fact that the slopes of fitted regression lines can represent any underlying task model satisfying the relation $S = \frac{e}{p}$ from Equation 6.22.

Therefore, to obtain reasonable values for e and p, we need to choose p that is a small fraction of t_a to obtain the asymptotic values for α and β, in other words, we aim to maximize the factor c defined in Section 6.5. Additionally, the choice of p or e can be arbitrarily guided by domain knowledge of the system under scrutiny. The other parameter can be estimated using the relation $S = \frac{e}{p}$ from Equation 6.22 upon deciding on the value of p or e.

In our example for $t_a = 4 \times 10^5$, we choose $p = 0.001$ x $t_a = 400$, and as a result, we can compute $e = S$ x $p = 0.027$.

Obtain Demand Variation Bound. The last parameters needed to obtain the relation between α and β are σ_u and σ_l. The σ parameter defines the difference between the fitted confidence interval curves after offsetting them to pass by the origin $(0, 0)$. Note that σ optimally can be expressed as $\sigma = (\mathrm{dbf}' + \text{intercept}') - (\mathrm{dbf} + \text{intercept})$, but we discard the difference between both intercepts as the error resulting from that approximation is negligible.

For the shown example in Figure 6.11b, we define $\sigma_u = -\sigma_l = \sigma'$ where $\sigma' = 0.957$ at $t_a = 4 \times 10^5$, which is the interval length of the window where we bound the fitted curve deviation.

Apply Feasibility Region Formulas. To obtain the estimated feasibility region for α and β, we plot the values of β_u and β_l corresponding to a valid range of α values using the slopes of fitted regression lines from Equation 6.23 and Equation 6.24. For the actual feasibility region, we plot the values of β_u and β_l versus the same α range using Equation 6.4.

Figure 6.12 shows the overlay of both feasibility regions using the arrival-curves model and the actual task model which, similarly to the illustration in Figure 6.4, describe the permissible values of α and the corresponding bounds on β. The negligible error between both the actual and estimated feasibility regions validates that the approximation of a linear empirical arrival-curves model to the assumed demand-bound function is reasonable.

The case study shows an example of an arrival-curves model that characterizes the behavior of QNX kernel event on a real-time system. Following the procedure presented in this section, the empirical model can now be represented as the demand-bound function of an equivalent task model whose parameters alteration can be bounded. The boundaries describe the robustness of the model since it quantifies the variation captured in the underlying normal behavior of the system, and as a result, giving valuable insights on how robust the model is and allowing for comparing different models.

One application for using the presented robustness assessment approach is the iterative model evaluation using arrival curves. In anomaly detection, it is important to evaluate whether the model is good enough during the training process. Our approach provides a way to quantify a feasibility region which can be integrated in the model training procedure, and as a result, the

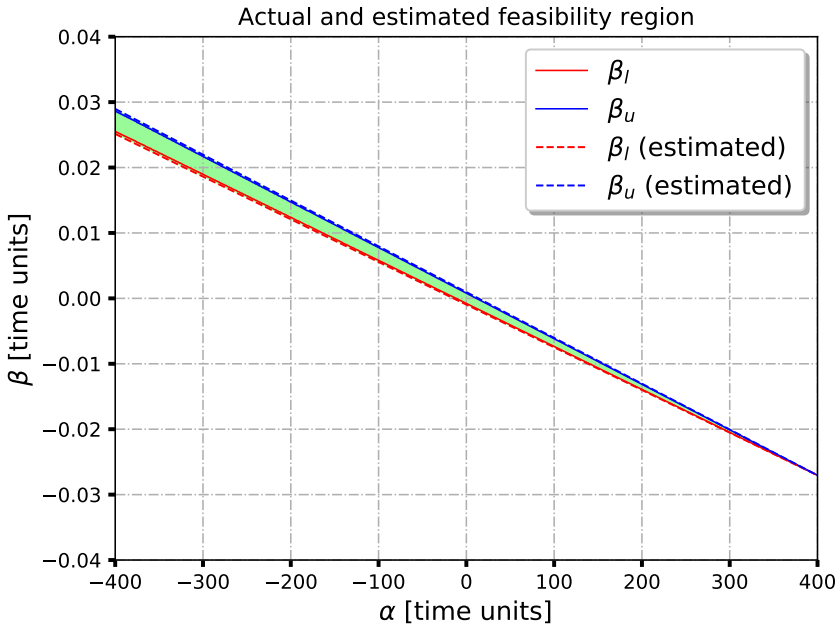


Figure 6.12: Feasibility region for representative parameters β, α

model is evaluated iteratively as new traces are added. This enables the tuning of models to meet industrial demands, i.e., certification purposes, where the domain expert specifies some robustness criteria based on these feasibility regions. In other words, to limit the tolerance of the model to given criteria, the specification can mention a relation between α and β as represented in the feasibility region, and the certification procedure can quantify the overlap between the specified region and the obtained feasibility region.

6.7 Related Work: Sensitivity Analysis in Scheduling

The work presented in this chapter fills the gap between the empirical evaluation of real-time systems and formal analysis that uses theoretical models to describe a given system. We provide a short overview on the closely related topic of sensitivity analysis from the scheduling domain.

In the presented analysis, we assume that the demand boundaries of a given task are defined and we aim to find the feasible task parameters that would not exceed such demand. Contrarily, research work in the domain of schedulability analysis aims to study whether a given set of tasks can be scheduled, i.e., meet the task demand without exceeding a given deadline, using different

scheduling methods [8, 52, 83] under various conditions such as, for example, permissible task overload [7]. Similarly, response time analysis (RTA) [6, 36] studies the worst case time for a task to meet a specified demand. For example, [27] studies the horizontal distance to the demand-bound function defining the time point at which a specific demand is achieved. In summary, existing works in schedulability analysis assume that task parameters are defined to study problems like worst-case execution time (WCET) for a given workload. However, in the domain of scheduling, our proposed analysis can be closely related to sensitivity analysis.

Sensitivity analysis [69, 73, 94, 96] studies how much change to task parameters, i.e., execution time or task period, will not violate scheduling constraints. Such analysis becomes relevant when constraints and specifications are not accurate. The early work on sensitivity analysis [46] computed the maximum variation of all execution time for a given set of tasks that keep a system schedulable for a rate-monotonic scheduler. Further work considered parameters other than execution time, for example, the authors in [9] presents a feasibility space for task deadlines to meet the constraint of schedulability. Authors in [94, 96] study the sensitivity analysis for EDF scheduling through the computation of optimal task parameters such that a given system remains schedulable. The authors exploit the Quick convergence Processor-demand Analysis (QPA) [93] algorithm which tests the schedulability of a task-set by a single iterative process. Particularly, [94] applied sensitivity analysis considering a varying task execution and [95] considered the case when the task period can be varied, while [96] assumed a fixed ratio between relative deadline and period. Our work considers a novel scope by obtaining feasibility regions for the permissible variation of task parameters, without restricting such variation to a single task variation parameter, to meet defined constraints on the increase and decrease to task demand rather than the schedulability condition.

6.8 Discussion

Linearity Assumption for Arrival Curves

The task model presented in Section 6.3 studies linear demand-bound functions by considering sporadic tasks with implicit deadlines under an EDF scheduler. We showed in Section 6.6 that having an empirical model that can be best approximated by a regression line minimizes the error between the actual and the estimated feasibility region as in Figure 6.12. However, the linearity assumption might not hold for other arrival-curves models [64], i.e., arrival curves describing a system with multiple modes.

Mode-switching yields an arrival curve that is increasing with positive slopes but with some horizontal gaps that correspond to the mode switches [64] because of the lack of events arrival

versus the increasing sliding window size. We conjecture that studying the increasing portion of the arrival curve is sufficient to study the behavior of the system under a specific mode of operation, i.e, discarding the horizontal gaps that do not belong to that specific mode.

Compositionality and Empirical Arrival Curves

Compositionality [17, 82] obtains a single timing requirement for a given real-time system from multiple timing requirements of the different system sub-components. Exploiting compositionality would translate multiple demand-variation bounds of different sub-components to a single demand variation bound for a given task model. Similarly, empirical evaluation using arrival curves allows for controlling the granularity of the analysis from system level to component level by controlling the source of trace events.

In Section 6.6, we presented an arrival-curves model that corresponds to a single QNX event, however, our work can be extended by using compositionality to combine the task models describing empirical arrival curves originating from multiple events into a system-level task model. This unified task specification can be compared against an empirical model constructed from traces that involves these events at once. In this case, the robustness evaluation can be performed on a system-level in contrast to the evaluation on event-level.

Handling Task Parameter Dependencies

Finding the feasibility region for the permissible task variation becomes a more complex problem if the parameters α_i and β_i of different tasks are dependent. In this case, translating such a complex relationship into a single demand-bound function, for example, using compositionality, might be a solution as we mentioned. The mathematical foundation presented in our work assumes that the variation of nominal parameters of multiple tasks is independent, but in practice, the tasks of a given real-time system might encounter the same alteration meaning that all tasks T_{K_i} are affected by the same variation α and β. To provide an example for this case, consider a system that has two independent tasks T_{K_1}, T_{K_2} with different execution time e_1, e_2 and period p_1, p_2. The system is prone to uncertainty affecting its demand-bound function, where the designer bounds the permissible variation to σ_1 and σ_2 respectively. However, the system tasks are affected by the same variation of execution time β and variation of period α respectively. This problem can be described as finding the feasibility region of α and β that satisfies Equation 6.4 for both Tasks T_{K_1}, T_{K_2} simultaneously. In other words, the feasibility region for α and β in the given system is defined by the intersection of both feasibility regions, where a valid α and β must satisfy both demand variation bounds for T_{K_1} and T_{K_2}.

6.9 Conclusion

This chapter presents an approach to evaluate the robustness of empirical arrival-curves models that characterize the behavior of real-time systems. We derive theoretical bounds on task parameter alteration permissible by the demand variation represented in the demand-bound function of a sporadic task with an implicit deadline under an EDF scheduler. We demonstrate the feasibility of the approach through an abstraction of an empirical arrival-curves model to a demand-bound function of the assumed task model. We evaluate the approach on the arrival-curves models constructed from QNX operating system events that describe the behavior of a real-time system.

Chapter 7

Methods for Constructing Empirical Arrival Curves

In this chapter, we discuss the implementations used to construct empirical arrival curves defined in Chapter 3. The algorithms evolved based on the applications discussed in this thesis. First, we discuss the basic approach of constructing arrival curves using a sliding window computation. Then, we discuss an efficient approach using Cartesian product mapping of a given trace which was shown not to be scalable enough in terms of memory usage. Finally, we present an algorithm implemented for parallel platforms that is scalable and efficient for longer traces.

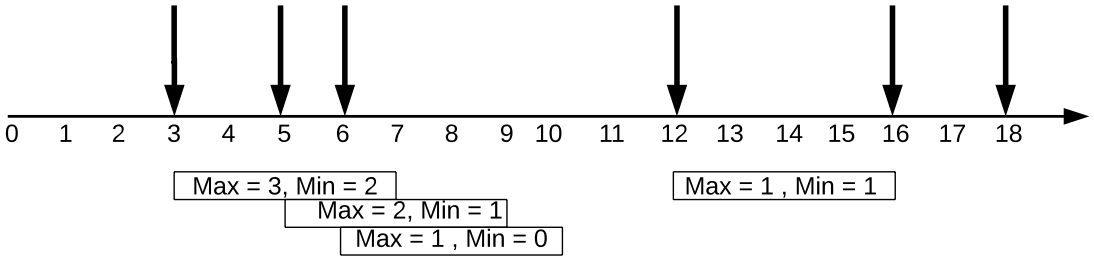


Figure 7.1: Example of Algorithm 1 execution

7.1 Variant # 1: Sliding Window Approach

The basic approach of constructing arrival curves from event traces gives a better understanding to the nature of the curves and their properties. For a given trace that can be described as a sequence of timestamps, we slide an overlapping window of a set of given time intervals throughout the time interval of the first and last trace element. The sliding windows are positioned to always start at the timestamp of an event.

During the process, the counts of the event occurrences within the sliding windows are recorded following the SymTA/S assumptions [75] where the maximum count considers the event at the beginning of the sliding window while discarding any event that would align with the end of the window, and vice versa for the minimum count. Repeating this procedure, the minimum and maximum counts corresponding to each window interval are obtained yielding the C_{max} and C_{min} as discussed in Chapter 3. Note that we discard incomplete windows, i.e., the windows that do not align with the end of the trace.

Algorithm 1 Sliding window algorithm to construct empirical arrival curves

Input: T, Δ_{max}, and res

// Loop on all window sizes
for $\Delta \leftarrow 0$ to Δ_{max} with step Δ_{min} **do**
// initialize counts to zero.
  $max_\Delta, min_\Delta = 0$
// wdx is the window index for current sliding window
  for wdx $\leftarrow 1$ to $\frac{\Delta}{\Delta_{min}}$ **do**
    t_{start} = wdx * Δ_{min}
    t_{end} = wdx * Δ_{min} + Δ
    wdx_max = number of events in wdx with $t \geq t_{start}$ and $t < t_{end}$
    wdx_min = number of events in wdx with $t > t_{start}$ and $t \leq t_{end}$
    if ($wdx\_max > max_\Delta$) **then** $max_\Delta = wdx\_max$
    end if
    if ($wdx\_min < min_\Delta$) **then** $min_\Delta = wdx\_min$
    end if
  end for
end for

We consider an example to show the construction using a trace T of timestamps $3, 5, 6, 12, 16,$ 18 and we consider a resolution, i.e., Δ_{min}, of 1 and the maximum window size, i.e, Δ_{max} of 16.

To give an example for the computation, consider the window size Δ of 4, the obtained counts from sliding a window of size 4 time units on the timeline in Figure 7.1 will result in max counts of $1, 2, 3$ events yielding a C_{max} of 3 and min counts of $0, 1, 2$ which gets a C_{min} of 0 for $\Delta = 4$. Similarly, executing the operation for different window sizes yields the maximum and minimum arrival curves.

Algorithm 1 shows a pseudocode for the illustrated computation in Figure 7.1. Obviously, the approach is exhaustive as the timestamps increase and resolution decrease. The algorithm is comprised of two loops that means for a long trace with all possible window sizes, the computation will be proportional to the square size of the trace. However, the purpose of introducing the algorithm is to show the basic idea of empirical arrival curves computation. Now, we show a more intuitive approach that has better efficiency.

7.2 Variant # 2: Cartesian Product Approach

To overcome the scalability issue mentioned in the previous section, empirical arrival curves can be constructed in a more intuitive way which does not require iterating on each time point away from a given event reducing the computation time needed by the sliding window algorithm.

We summarize Algorithm 2 as follows: The algorithm constructs a Cartesian product of the timestamps represented as two columns. The time difference between the two columns can be then reduced to the values less than the maximum window size defined in the input. An associated sequence number is combined with the difference in timestamps whose values are mapped to the corresponding window sizes. For a given window size, the counts are recorded and similar to the previous algorithm, the C_{min} and C_{max} values of that Δ can be computed through some aggregation method.

In the same example as before, we consider the trace $3, 5, 6, 12, 16, 18$ and the resolution 1 and a maximum window size of 16. The Cartesian product of the trace can be shown in Figure 7.2, we limit our analysis to the $\Delta = 4$, one column corresponds to the start time and the other to the end time. Subtracting both, we now convert the first column to the difference between start and end timestamps. We replace column two with an index that resets when the time difference is 0 in case of C_{min} and 1 for C_{max} case. We map the time difference to the resolution of interest, which is not a necessary step in our example as the resolution is 1. At this point, every window size in column one has a corresponding count in the second column. Performing a cumulative maximum and minimum analysis as shown in Figure 7.2, we can obtain C_{max} and C_{min}. Algorithm 2 shows pseudocode to perform this procedure.

Algorithm 2 Cartesian product algorithm to construct approximate empirical arrival curves

Input: T, Δ_{max}, and Δ_{min}

1: crtpdt = Cartesian product of T x T
   // limit the result to the maximum window size of interest
2: crtpdt = crtpdt[(crtpdt[, 1] >= crtpdt[, 2]),]
3: crtpdt = crtpdt[(crtpdt[, 1] < (crtpdt[, 2] + Δ_{max}),]
   // Obtain the difference between events
4: crtpdtmin = crtpdt − crtpdt[, 2]
5: crtpdtmax = crtpdt − crtpdt[, 2] + 1
   // Cut the time series into separate sub-series at difference of zero
6: *seriesseps* = which(crtpdt[, 1] == 0)
   // In case the resolution is higher than the timestamp differences
7: *serieswnds* = (crtpdt[, 1] % Δ_{min}) ∗ Δ_{min}
   // Assign counts for each subseries of *serieswnds*
8: *seriescntsmin* = a list of sub-series ranging from 0 to *seriesseps* length
9: *seriescntsmax* = a list of sub-series ranging from 1 to *seriesseps* lengths
   // Cumulative minimum in reverse order for each subseries of *serieswnds*
10: C_{min} = cummin(reverse(*seriescntsmin*))
   // Cumulative maximum for each subseries of *serieswnds*
11: C_{max} = cummax(*seriescntsmax*)

Although the algorithm is computationally more efficient than the sliding window algorithm, it becomes memory infeasible as traces grow due to the required storage of the Cartesian product of the trace. Fortunately, we partially overcome this problem through the breakdown of the Cartesian product into independent data blocks. And as a result, we modify the algorithm to perform computations limited to the available memory block size. The operation is performed block by block and then the maximum and minimum count is performed after all the grid has been processed.

However, the mentioned breakdown approach now faces another problem which is the trade-off between the memory size and the computation power needed to execute the algorithm. As the efficiency starts to degrade as the Cartesian product of the trace goes much beyond the available memory size. The iterations needed to process the entire Cartesian product increase which opposes the intuition of the original algorithm.

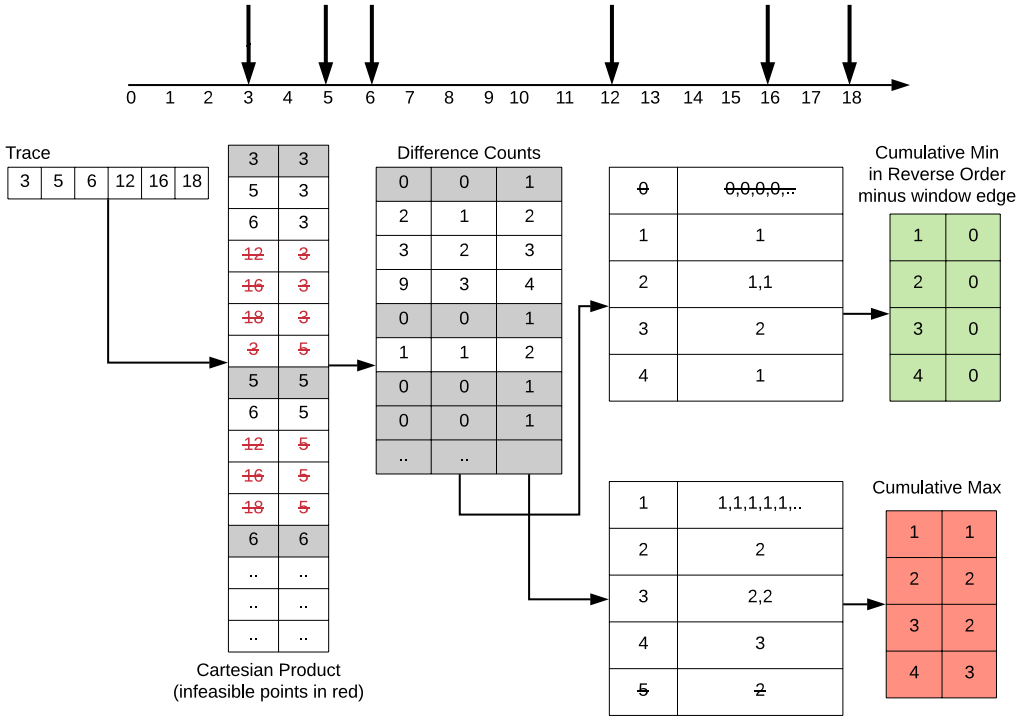


Figure 7.2: Example of Algorithm 2 execution

7.3 Scalable Implementation of Arrival Curves

In this section, we present an algorithm that was used throughout the thesis for constructing empirical arrival curves. The algorithm uses a quantization technique that enables having a highly parallelizable algorithm capable of running on CPU and GPU platforms and be scalable for longer traces. We discuss the definitions and experiments that are relevant to the work presented in this thesis, however, a detailed discussion for the algorithm and the acceleration approach can be found in [15, 74].

7.3.1 Definitions

We revisit some of the definitions introduced in Chapter 3, as we extend the definitions to a variant of arrival curves used by the algorithm.

As we discussed throughout the thesis, a trace registers chronological sequences of events occurring while the system operates. A trace will include information such as an index, timestamp, and additional parameters that depend on the application environment. In this section, we will consider a reduced representation where the index and timestamp of each event are the only parameters of interest in what we call a trace sequence.

Definition 11. *(**Event source**) An event source generates elements $ts_k, k \geq 0$, representing the timestamp of an event measured in an absolute time domain t. Timestamps are multiples of an atomic time unit ($ts_k \in \mathbb{N}$), and events are generated as time progresses ($ts_k < ts_{k+1}$).*

Definition 12. *(**Trace Sequence**) A trace sequence $TS = [ts_0, ts_1, \ldots, ts_{N-1}]$ is a finite sequence of N timestamps collected from an event source.*

Definition 13. *(**Actual Empirical arrival curve**) The pair of curves $(\phi^l(TS, \Delta t); \phi^u(TS, \Delta t))$ provide an actual lower and upper bound on the number of events seen in any time interval of length Δt in a trace sequence TS.*

We now formalize quantized arrival curves as approximations of actual empirical curves based on discrete *buckets* (or bins) of width r atomic time units. Discrete buckets provide a coarser representation of the time intervals used to construct the actual curve. The following section describes a rule to map events observed within a range of time intervals to a single bucket.

Definition 14. *(**Quantized arrival curves**) Given a set of discrete buckets Δt_i^B of width r atomic time units ($r > 1$), with bucket Δt_i^B enclosing all intervals Δt in the range $ir \leq \Delta t < (i+1)r$, with $i \in \mathbb{N}^0$. Quantized lower $\hat{\phi}^l(TS, \Delta t, r)$ and upper $\hat{\phi}^u(TS, \Delta t, r)$ arrival curves will provide a unique representative value for all intervals Δt enclosed in a bucket. Quantized curves must comply with the following property:*

$$\begin{aligned} \forall r > 1 : \hat{\phi}^l(TS, \Delta t, r) \leq \phi^l(TS, \Delta t) \wedge \\ \hat{\phi}^u(TS, \Delta t, r) \geq \phi^u(TS, \Delta t) \end{aligned} \tag{7.1}$$

□

Requirement (7.1) states that upper curves should be approximated from above and lower curves from below [42].

For notational simplicity, we will assume that all curves come from the same generic trace and omit the argument TS. We also consider the equivalence $\hat{\phi}^l(\Delta t, r = 1) \equiv \phi^l(\Delta t)$, and use the notation $\hat{\phi}^l(\Delta t_i^B, r)$ to represent the quantized value of the lower curve in all intervals Δt in the bucket i. The same applies to the corresponding upper curves.

7.3.2 Intuitive Overview

To model arrival patterns from a trace, we do not need to check for all possible intervals within the trace, but only for intervals aligned to the occurrence of events [63, 75]. Based on this observation, we propose an iterative method to construct empirical curves by aggregating the information observed in intervals of certain length measured from a pivot event.

For example, let us consider the sample trace sequence discussed throughout this chapter $TS = [3, 5, 6, 12, 16, 18]$, with each element representing the timestamp of an event. Figure 7.3 shows a representation of TS over the absolute time axis t, where vertical arrows indicate the time of occurrence of an event. For constructing arrival curves, we map each ts_k from the absolute time domain t to the interval time domain Δt using a reference pivot ts_p, with $p \in [0, N-1]$, according to a function $M : t \rightarrow \Delta t$:

$$M(ts_k, ts_p) = ts_k - ts_p, \text{with } p \leq k \leq N-1 \tag{7.2}$$

We perform the mapping iteratively, generating on each iteration p a new distribution of events in the Δt domain, for which we obtain a local lower $\hat{\phi}_p^l(\Delta t_i^B, r)$ and a local upper $\hat{\phi}_p^u(\Delta t_i^B, r)$ curve associated with the pivot. To obtain local curves, we find a lower and upper bound for the number of events observed in intervals measured from the origin of the interval time domain ($\Delta t = 0$), where the pivot is located, to the extremes of each interval bucket Δt_i^B. After finishing iteration p, the current pivot does not provide more information, and then we start a new iteration placing the next timestamp at the origin.

When $r = 1$ (i.e., no quantization), we obtain the local curves according to the following rules [75]:

- The local lower value $\phi_p^l(\Delta t)$ corresponds to the number of events in the interval $(0, \Delta t]$ in the current iteration.
- The local upper value $\phi_p^u(\Delta t)$ corresponds to the number of events in the interval $[0, \Delta t)$ in the current iteration.

According to Definition 14, when $r > 1$, we must assign a unique value to all interval lengths enclosed in a bucket, ensuring that the quantized lower (upper) curve approximates the curves obtained with $r = 1$ from below (above). The following rules accomplish this requirement:

- The local value $\hat{\phi}_p^l(\Delta t_i^B, r > 1)$ corresponds to the number of events in the interval $(0, ir)$ in the current iteration.

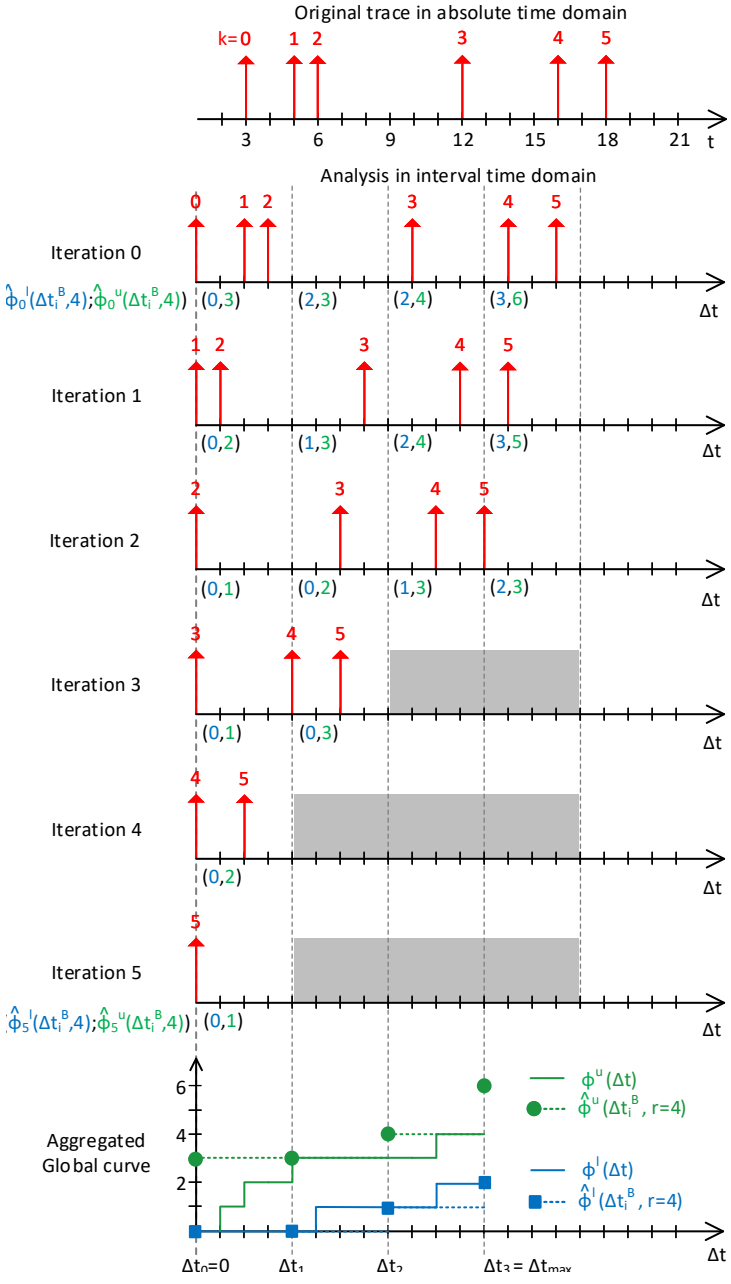


Figure 7.3: Overview of an intuitive processing flow for computing arrival curves. [15]

- The local value $\hat{\phi}_p^u(\Delta t_i^B, r > 1)$ corresponds to the number of events in the interval $[0, (i+1)r)$ in the current iteration.

Figure 7.3 illustrates the application of the method to the sample trace, with a bucket width of $r = 4$ time units. The argument Δt_{max} specifies the maximum timespan of interest and the number of buckets in the resulting curves. In the example, the curves consider three buckets ($\Delta t_{\text{max}} = 3r$). In general, $\Delta t_{\text{max}} \leq (ts_{N-1} - ts_0)$. The vertical dashed lines delimit the range of intervals that map to a single bucket. The pair of numbers below the Δt axes represent the local lower and the upper number of events obtained for each bucket following the previous rules for $r > 1$. After iterating over all pivots, we get the global lower and upper curves by finding the minimum and the maximum number of events seen in all local curves for each bucket Δt_i^B, respectively. The frayed regions in the latest local curves indicate that no events occur in those buckets, and we omit them when computing the global value. The last plot shows the global curve for $r = 4$, which properly bounds the curve that would be obtained from the same trace using a bucket width $r = 1$. Intuitively, we construct the curve by sliding the trace events over the axis representing the interval domain. The curves are only defined for integer values of Δt, and the staircase plots are just included to facilitate visualization. Also, the value of the curve exactly at the origin ($\Delta t = 0$) is zero [75].

7.3.3 Algorithmic Formulation

Algorithm 3 shows a pseudo-code for constructing empirical curves given the arguments Δt_{max} and $r > 1$. The algorithm first computes the number of buckets and initializes variables and vectors (Lines 1-4). The i-th element of vectors LowLoc and UppLoc will store the local lower and upper value for bucket Δt_i^B on each iteration. Similarly, LowGlob and UppGlob will store the aggregated global curves.

The main loop starting at Line 5 iterates over the timestamps, each time constructing a local curve with respect to the pivot ts_p. After initializing the local vectors to a negative value, a first inner loop maps each timestamp $ts_k \geq ts_p$ to a bucket based on its distance from the pivot. The resulting vector IntervalToBucket contains a sequence of elements with value i for all timestamps within the range $ir \leq \Delta t < (i+1)r$ in the interval domain. Note that W sets the length of IntervalToBucket, which must store all events that lie in the timespan Δt_{max}. For example, if the minimum delay d_{min} between consecutive events is known, then an adequate estimation would be $W \geq (\Delta t_{\text{max}}/d_{\text{min}})$.

The algorithm gets the local upper value for bucket Δt_i^B by searching elements with value i in IntervalToBucket. If there is any, then UppLoc$[i]$ will store the number of elements until bucket i

Algorithm 3 Get quantized arrival curve from trace TS

Input: TS, $\Delta t_{\max}$, r, W

1: $nBuckets = \lfloor \Delta t_{\max}/r \rfloor$
2: LowLoc [$nBuckets$], UppLoc [$nBuckets$]
3: LowGlob [$nBuckets$] = N+1 , UppGlob [$nBuckets$] = 0
4: IntervalToBucket [W]

5: **for** $p \leftarrow 0$ to $N-1$ **do**
6: $h = 0$, LowLoc[all] = -1, UppLoc[all]= -1
// Loop 1: Compute quantized values of intervals
7: **for** $k \leftarrow p$ to $N-1$ **do**
8: IntervalToBucket[h] = $\lfloor (TS[k] - TS[p])/r \rfloor$
9: $h = h + 1$
10: **if** ($h \geq W$ or IntervalToBucket[h] $> nBuckets$) **then**
11: break;
12: **end if**
13: **end for**

// Loop 2: Find local upper value for each bucket Δt_i^B
14: **for** $i \leftarrow 1$ to $nBuckets$ **do**
15: **if** $i \in$ IntervalToBucket **then**
16: UppLoc[i] = (index of last element equal to i);
17: **end if**
18: **end for**

// Loop 3: Fill gaps in local upper curve
19: **for** $i \leftarrow 1$ to $nBuckets$ **do**
20: **if** UppLoc[i] < 0 **then**
21: UppLoc[i] = UppLoc[$i-1$];
22: **end if**
23: **end for**

// Loop 4: Update global upper curve
24: **for** $i \leftarrow 1$ to $nBuckets$ **do**
25: UppGlob[i] = max(UppLoc[i], UppGlob[i])
26: **end for**
27: **end for**

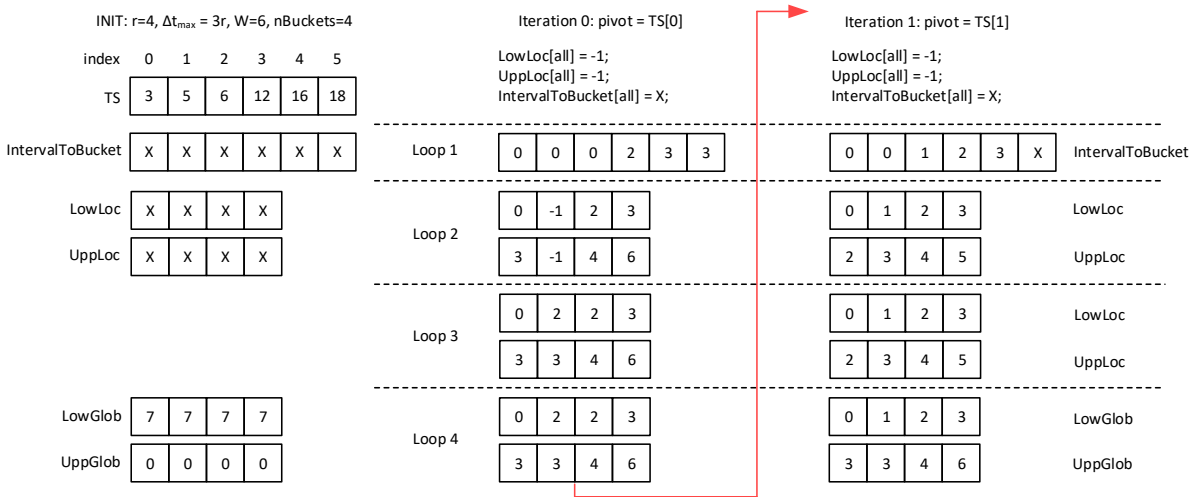


Figure 7.4: Illustration of steps to construct curves using the first two iterations over an example trace.

ends, i.e., the index of the last element with value i in IntervalToBucket (Loop 2). If the trace has consecutive timestamps separated by more than r time units, then some indexes may not appear during some iterations, and the corresponding element UppLoc$[i]$ will hold a negative value. Loop 3 fills the gaps in UppLoc, replacing all negative values with the value of the previous bucket that had at least one event mapped in the current iteration. Finally, Loop 4 performs a point to point comparison between UppLoc and UppGlob, keeping in UppGlob the maximum between the upper local curve and the upper global curve constructed from previous iterations.

Algorithm 3 omits the steps for the lower curves, which are similar to the ones in Loops 2 to 4 with some few straightforward modifications. The difference with respect to Loop 2 is that LowLoc$[i]$ will store the number of events until bucket i starts, i.e., the index of the first element with value i decremented by one. An equivalent Loop 3 fills the potential gaps (negative values) in the local lower curve, in this case replacing negative values with the value of the next bucket with a positive value. Finally, a similar processing to Loop 4 updates the global lower curve keeping the minimum between the current local curve and the aggregated global curve. Figure 7.4 illustrates the processing steps performed during the first two iterations.

Loops 1 and 4 operate on independent elements of the corresponding vectors, exposing a high-degree of data-level parallelism that is straightforward to exploit using simple kernels for parallel devices. Loops 2 and 3 exhibit a more complex data dependency in their operations, but we can still accelerate them using advanced libraries and custom kernels. For example, Loop 2 performs a form of stream compaction (given a sparse input vector, pack a subset of this vector into a dense output vector), for which there exist multiple libraries targeting parallel processing [23, 66].

7.3.4 Performance Evaluation

We use the traces from the SSPS dataset introduced in Chapter 4. Using the *tracelogger* utility, we register transitions in kernel states using a timestamp (in nanoseconds) and identifiers for the type of event. We collect these transitions in *master traces*, each one containing 19 type of events. We then split each master trace into 19 sub-traces, one per each event type. Under normal operation, each sub-trace should map to arrival curves that conform to the model of recurrent behavior for the corresponding event type as discussed in Chapter 3. We also include traces for anomalous operation scenarios, obtained by inserting sporadic tasks that interfere with the normal flow of the SSPS application. We implemented Algorithm 3 using two configurations:

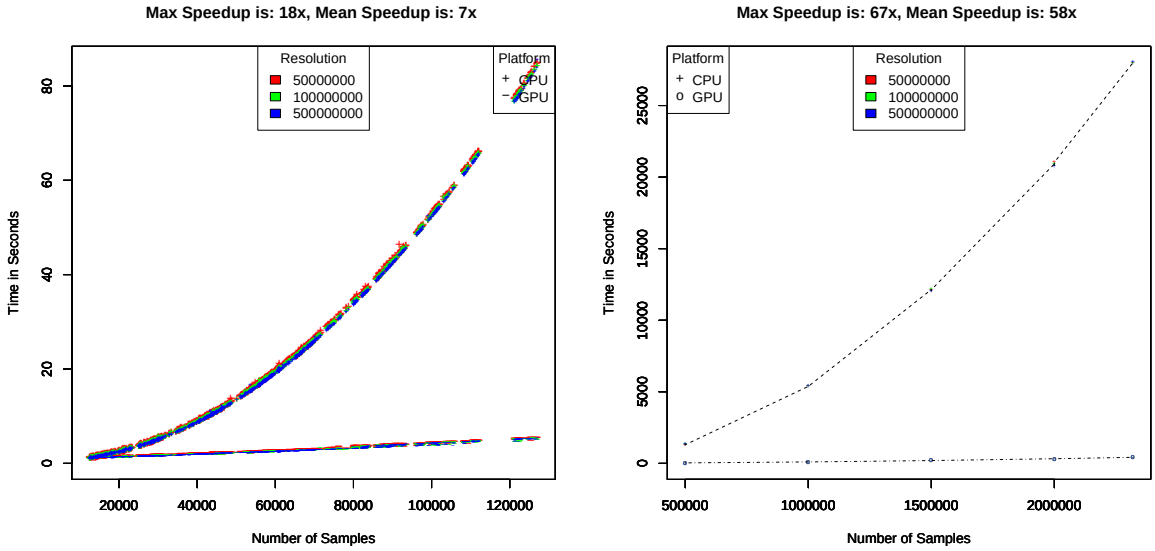
- A sequential version in C running on a machine with four 8-core Intel Xeon E2360 v3 and 128GB of RAM.
- An OpenCL version targeting GPUs, using custom kernels that parallelize the inner loops. This version runs on a machine with a 4-core Intel i7-3820, 32GB of RAM and AMD Firepro W9100 GPU.

We evaluate the effects of the trace length in the classification performance of TRACMIN, we generated different sets of master traces that consider only the first N events of the original traces, with $N = [0.5, 1, 1.5, 2.5] \times 10^6$. For each N, we obtained sub-traces for each event type whose lengths range between 20×10^3 and 160×10^3 events. We constructed empirical arrival curves for each sub-trace and used these curves to test the classification performance. We also evaluated the effect of the bucket width argument required to construct the curves. Although using a small bucket width(measured in time units of the timestamps) would allow us to capture fine-grained details from the dataset, this is not always desirable from a classification perspective as it may lead to over-fitting of the data. Besides, the separation between consecutive events may vary along the traces and have a coarser granularity than the timestamps. For example, the minimum and maximum distance between events in the experimental set move around 15×10^3 and 120×10^3 time units, respectively. In this case, bucket widths smaller than the minimum separation between events will lead to unnecessary buckets in the global curve. Therefore, the bucket width offers a tunable parameter to find the best trade-off between resource utilization and classification performance.

Figure 7.6 shows the computation time spent on constructing curves for each sub-trace in the experimental set. The plot includes a set of shorter sub-traces (not used for classification tests) to emphasize the effects of the different implementations. The sequential CPU version runs faster for low values of N, but the benefits of the parallel GPU version become evident as $N > 10 \times 10^3$. For $N \approx 160 \times 10^3$ the GPU version runs about 18X faster than the CPU one.

An additional experiment reports that the GPU version runs around 67X faster than the CPU one for $N \approx 2.5 \times 10^6$. Figure 7.5a shows the computation time for both platforms versus the increasing number of samples using event sub-traces mentioned earlier. The max performance gain achieved is 15x and the mean speedup is 7x. The computation time of the CPU platform despite being much slower than the GPU version was still feasible to achieve the accuracy results of Table 7.1, as we mentioned the computation was not feasible using the existing algorithms.

Using entire traces from the SSPS dataset without separating similar event types, the trace lengths reach 2M samples as mentioned earlier. We make use of the lengthy traces to compute the speedup in Figure 7.5b which shows max performance gain of 67x and mean performance gain of 56x. Both plots in Figure 7.5 show that the CPU timing versus the trace lengths curve show an exponential trend, however the GPU timing versus the trace lengths show a low slope linear trend. Hence the scalability of the GPU implementation versus the trace length.



(a) Event-type trace computation performance

(b) Entire trace computation performance

Figure 7.5: Effect of trace length on performance

Table 7.1 summarizes the main results for the classification performance, including the best TPR and FPR obtained for each length of the master traces and the computation time for constructing the curves for all sub-traces in the set. The table shows that a perfect classification score is possible only when using large traces, which justify the need of having efficient computational tools. The table also indicates that the bucket width argument can affect the classification.

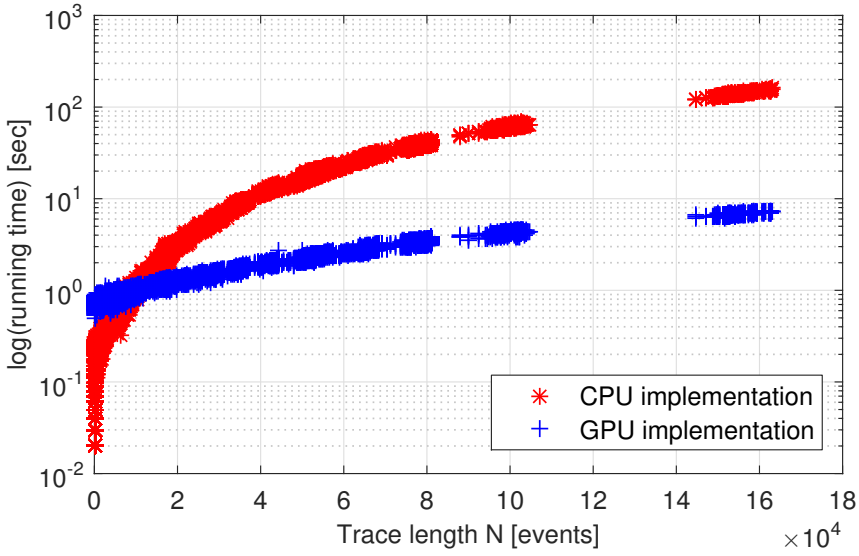


Figure 7.6: Elapsed wall-clock time spent on constructing the arrival curves

Table 7.1: Summary of Classification Performance for Different Trace Sets.

Trace length	Bucket width [ns]	TPR	FPR	Comp. time (GPU) [s]
$5x10^5$	$100x10^6$	70%	45%	1551.37
$1x10^6$	$100x10^6$	75%	25%	2447.26
$1.5x10^6$	$10x10^6$	100%	0%	3545.79
$1.5x10^6$	$100x10^6$	100%	0%	3439.17
$2.5x10^6$	$10x10^6$	100%	4%	3722.37
$2.5x10^6$	$100x10^6$	100%	0%	3620.57

Figure 7.7 illustrates the effect of using different resolutions when constructing a particular curve. We see that the curve with larger bucket width correctly bounds the one with lower bucket width. Again, calculating these curves was infeasible with the basic algorithms of sliding window and Cartesian product.

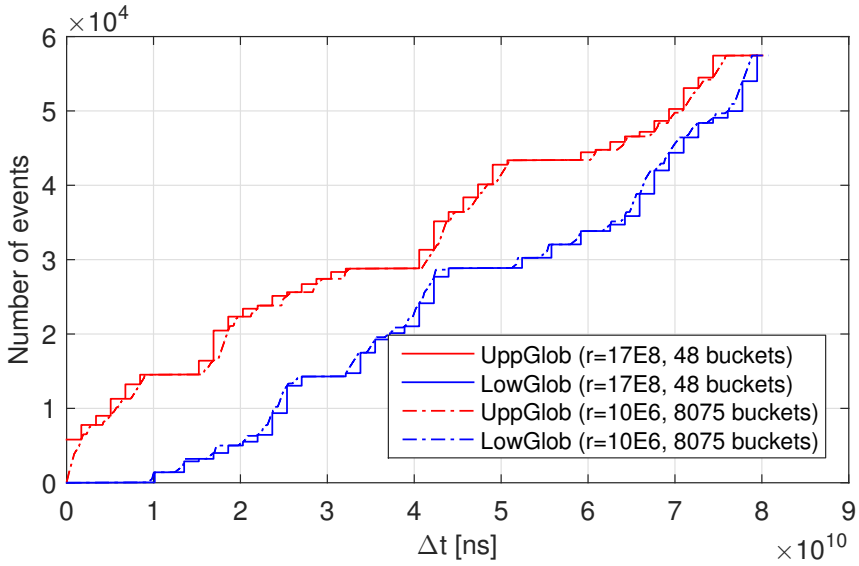


Figure 7.7: Arrival curves for the same trace with different bucket widths

7.4 Conclusion

In this chapter, we presented three algorithms that can be used to construct arrival curves from event traces. The sliding window approach provides a clear understanding of the approach but can be used for demonstration on short traces. The Cartesian product algorithm provided an intermediate step in achieving efficient performance, however, it can serve the purpose on general-use machines. Finally, we presented a more complex algorithm to construct the curves. The algorithm exposes a high-degree of data-level parallelism, we showed an implementation using a commodity GPU that outperforms equivalent non-parallel reference implementations and demonstrated its utility using a case-study that uses lengthy traces.

Chapter 8

Conclusion

With the rise of Industry 4.0 and digital twin concepts [35, 80], data collected from non-invasive tracing tools are used to improve diagnostics and prognostics of real-time systems. Event traces collected from a given system provide valuable information for performing runtime analysis when formal methods become complicated and infeasible [3]. Although trace mining to model the behavior of a given system is not novel, most of the applications targeted non-real time domains [4]. The combination of trace mining with the advanced modeling and statistical learning techniques over runtime data provides a powerful tool for performing postmortem analysis or audits of modern CPS [32, 33, 55, 91].

Identifying anomalous behavior in event traces is a step towards improving the resilience of complex real-time systems through the development of graceful degradation techniques and countermeasures for preventing faults. As it becomes clearer that verification of modern real-time systems cannot purely rely on broad generic abstractions and theoretical analysis, it is more important than ever to promote evidence-based discussions and drive new research towards novel data-driven techniques for analyzing more representative system-specific data.

The thesis answers the question of whether arrival curves, widely used in the formal analysis of real-time systems, can be constructed from event traces and provide good high-level features to describe the behavior of such systems for anomaly detection purposes. Throughout the thesis, we evaluated the feasibility of arrival curves in this context through a trace mining framework that used a set of metrics and properties applied to real-world case studies. The evaluation methods demonstrate the viability of empirical arrival curves in abstracting the behavior of real-time systems using indexed and timestamped event data. More specifically, we summarize the thesis contributions as follows:

- Defining empirical arrival curves and exploiting the widely-used concepts of arrival curves to a variant computed over both indexed and timestamped event data generated from real-time systems. In Chapter 3, we introduced a set of properties and metrics that allow reasoning about the characteristics of empirical curves for feature extraction purposes. We relate our work to the domains of Network Calculus and Real-time Calculus as we reuse a couple of shape-based metrics that are typically used to define the curve slope and burstiness.

- Introducing a trace mining framework, TRACMIN, in Chapter 4. The framework builds offline models based on empirical arrival curves, then employs statistical learning techniques for anomaly detection purposes. The classification accuracy achieved by TRACMIN in labeling normal and anomalous traces while experimenting with multiple datasets from a real-time system demonstrated the feasibility of the approach. For the presented case study that uses indexed event traces, TRACMIN achieved a true positive rate of at least 94% while having a false positive rate of 0%. For another case study on a single-mode real-time system having timestamped traces, TRACMIN achieved a classification result of 100% true positive rate and a 3% false positive rate.

- Extending the TRACMIN framework in Chapter 5 as follows: First, Section 5.1 introduced a clustering approach that uses SAX method for representing arrival curves as strings of characters on which we perform an edit-distance computation. Building offline arrival-curves models specific to modes of operations improved the classification results in a case study that uses a real-time system having multiple modes of operation from a true positive rate and false positive rate of 85% and 45% to 100% and 14% respectively. Second, we deployed TRACMIN in a trace streaming framework PALISADE in Section 5.2, and evaluated its on-the-fly anomaly detection on ROS operating system traces streaming from remote vehicles in a lab demonstrator. Third, we used TRACMIN framework in Section 5.3 to successfully mine and visualize the existing recurrent patterns using auto-correlation computations on a variant of empirical arrival curves.

- Evaluating the robustness of arrival-curves based models in Chapter 6 fills a gap in the research methods of evaluating empirical anomaly detection techniques. We derived theoretical bounds on the statistical variations obtained in TRACMIN offline models. Through a novel mapping between empirical arrival curves and demand-bound functions from the scheduling domain, arrival-curves model can be abstracted to some task model whose parameters variation boundaries can be obtained. We demonstrated how the arrival curves model for a given trace event from a real-time system can be represented as a sporadic task with implicit deadlines under an EDF scheduler, the feasibility region obtained for the

variation of the task parameters allow for comparing different arrival-curves models and enabling for the certification and standardization purposes in the industry.

- Introducing a rapid and scalable algorithm suited for parallel platforms to construct what is called quantized arrival curves in Chapter 7. The quantized curves bound the maximum and minimum empirical arrival curves used throughout the thesis in the TRACMIN framework. As a result, we overcome the scalability issues for constructing empirical arrival curves. We relate our work to the existing arrival curves computation methods by constructing the curves using the SymTA/S assumptions [75]. The presented algorithm in the chapter achieves mean speedup of 7x on typical traces in our case studies when running on GPU platform compared to CPU platforms. However, mean speedup of 58x can be achieved on lengthy traces used for performance measurement purposes. The implementation ensures the ability to construct empirical arrival curves from timestamped traces in order to obtain offline models and classify traces in both offline and online fashions with reasonable performance. We highlight that such scalability is not achievable using the existing algorithms in the literature.

The contributions of this thesis pave the way for more research to be done. Either on exploring means that better characterizes the empirical curves for even more complex tasks or extending the applications of the curves beyond real-time systems. We facilitate these possibilities by highlighting some possible research problems revealed throughout the thesis:

- **Anomaly Localization within Traces.** The purpose of anomaly detection in the thesis was to classify an entire trace either as normal or anomalous. Using the entire trace ensures that all the information is considered for analysis. Since arrival curves proved to be useful for this kind of post-mortem analysis, a challenging line of research would be to study the suitability of arrival curves for localizing anomalies within a given trace. For example, the streaming framework presented in this thesis can be adapted to locate anomalous windows (or sub-traces) within a trace. However, one main issue with handling sub-traces will be defining the splitting points within a trace without affecting the recurrence characteristics captured in the trace.

- **Machine Learning using Arrival Curves.** The statistical approaches used in this thesis rely on reasoning about the shape of the curve defined by slope-based metrics of a single curve, the area under that curve, the proximity of multiple curves, etc. One main advantage of shape-based metrics is the engineerability of the obtained results. For example, the change in slope-based metrics can be interpreted as a change in the rate of arrival of events

within a trace. Similarly, the proximity of maximum and minimum arrival curves describes an envelope of the normal behavior change during execution.

Since arrival curves abstract the behavior captured in a given trace as a multi-dimensional set of features, it would be interesting to apply sophisticated machine learning approaches to arrival curves. The methods would allow for less control on the interpretability of the results, however, the results from these techniques might allow for better feature abstractions than the shape-based metrics used throughout the thesis. Examples on the machine learning-based approaches include Autoencoders [31] for predicting the arrival behavior using the curves, and Random Forests [12] for classifying a bigger set of traces while pinpointing anomalous window intervals contributing to a specific fault.

- **Modeling Behavioral Trends and Correlations.** Real-time systems would exhibit changing trends throughout their execution, for example, the software would need to accommodate for a decaying battery performance. It is crucial to account for these trends in the empirically computed models through incremental updates. Building an arrival-curves model allows for such ongoing modeling that incorporates benign variations in the arrival behavior of events.

  In addition, since the events generated by a real-time system are naturally related, studying the behavior captured by a given arrival curve of an event as it changes with the behavior of another event would reveal more insights on the execution dynamics of the target system. One way to achieve this is by studying the correlation of arrival curves corresponding to the events of different processes.

- **Statistical Learning and Worst-Case Analysis.** The TRACMIN framework presented in Chapter 4 builds a normal behavioral profile for the target system. However, the employed statistical learning approach is not directly applicable to worst-case analysis purposes.

  For trace mining approaches, the accuracy of the training model improves with the quality of the available data [34]. To account for worst-case scenarios, the training phase must consider traces collected during the execution of such scenarios in contrast to the experiments presented in the thesis where the traces represent the typical behavior of the system. In addition, as discussed in the thesis, TRACMIN aggregates the arrival curves using the mean with confidence intervals to suppress the effect of outliers during training. As a result, another aggregation methods should be employed for the purpose of worst-case analysis to consider the points of arrival curves that characterize the extreme scenarios and incorporate their effect in the trace mining procedure.

References

[1] ISO-26262, Road Vehicles, Functional Safety. 2011.

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

[3] Leonie Ahrendts, Rolf Ernst, and Sophie Quinton. Exploiting execution dynamics in timing analysis using job sequences. *IEEE Design & Test*, 35(4):16–22, 2018.

[4] Terrasa Andres and Bernat Guillem. Extracting temporal properties from real-time systems by automatic tracing analysis, technical university valencia. *Real-time systems research group University of York*, 2003.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[6] Neil C Audsley, Alan Burns, Robert I Davis, Ken W Tindell, and Andy J Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2-3):173–198, 1995.

[7] Akramul Azim, Shreyas Sundaram, and Sebastian Fischmeister. An efficient periodic resource supply model for workloads with transient overloads. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 249–258. IEEE, 2013.

[8] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190. IEEE, 1990.

[9] Enrico Bini and Giorgio Buttazzo. The space of edf deadlines: the exact region and a convex approximation. *Real-Time Systems*, 41(1):27–51, 2009.

[10] George Box. Box and jenkins: time series analysis, forecasting and control. *A Very British Affair: Six Britons and the Development of Time Series Analysis During the 20th Century*, 161, 2012.

[11] George EP Box and Gwilym M Jenkins. *Time series analysis: forecasting and control, revised ed*. Holden-Day, 1976.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Peter Buchholz and Sebastian Vastag. Toward an analytical method for sla validation. *Software & Systems Modeling*, pages 1–19, 2018.

[14] Alvaro A Cardenas and Natalia Stakhanova. Analysis of metrics for classification accuracy in intrusion detection. In *Empirical Research for Software Security*, pages 173–199. CRC Press, 2017.

[15] Gonzalo Carvajal, Mahmoud Salem, Nirmal Benann, and Sebastian Fischmeister. Enabling Rapid Construction of Arrival Curves from Execution Traces. *IEEE Design & Test*, 2017.

[16] Samarjit Chakraborty. *System-level timing analysis and scheduling for embedded packet processors*. PhD thesis, ETH Zurich, 2003.

[17] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, volume 3, page 10190, 2003.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5):823–839, 2012.

[20] Varun Chandola, Varun Mithal, and Vipin Kumar. Comparative evaluation of anomaly detection techniques for sequence data. In *2008 Eighth IEEE International Conference on Data Mining*, pages 743–748. IEEE, 2008.

[21] Joint Commission et al. Medical device alarm safety in hospitals. *Sentinel Event Alert*, (50):1, 2013.

[22] Rene L Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on information theory*, 37(1):114–131, 1991.

[23] Denis Demidov, Karsten Ahnert, Karl Rupp, and Peter Gottschling. Programming CUDA and OpenCL: A case study using modern C++ libraries. *SIAM Journal on Scientific Computing*, 35(5):C453–C472, 2013.

[24] Mellitus Ezeme, Akramul Azim, and Qusay H Mahmoud. An imputation-based augmented anomaly detection from large traces of operating system events. In *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pages 43–52. ACM, 2017.

[25] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[26] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[27] Nan Guan and Wang Yi. General and efficient response time analysis for edf scheduling. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 255. European Design and Automation Association, 2014.

[28] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[29] M. Hassan. Heterogeneous mpsocs for mixed-criticality systems: Challenges and opportunities. *IEEE Design Test*, 35(4):47–55, Aug 2018.

[30] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis-the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[31] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[32] Kai Huang, Gang Chen, Christian Buckl, and Alois Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*, pages 430–436. ACM, June 2012.

[33] Oleg Iegorov and Sebastian Fischmeister. Mining task precedence graphs from real-time embedded system traces. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Porto, Portugal, 2018.

[34] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[35] Nasser Jazdi. Cyber physical systems in the context of industry 4.0. In *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, pages 1–4. IEEE, 2014.

[36] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[37] Brendan Juba, Christopher Musco, Fan Long, Stelios Sidiroglou-Douskos, and Martin C Rinard. Principled sampling for anomaly detection. In *NDSS*, 2015.

[38] Anthony W Knapp. *Basic real analysis*. Springer Science & Business Media, 2005.

[39] Donald Erwin Knuth, Ronald L Graham, Oren Patashnik, et al. Concrete mathematics. *Adison Wesley*, 1989.

[40] Roger Koenker. *quantreg: Quantile Regression*, 2018. R package version 5.38.

[41] Simon Künzli and Lothar Thiele. Generating event traces based on arrival curves. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2006 13th GI/ITG Conference*, pages 1–18. VDE, 2006.

[42] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and Yi W. Generalized finitary real-time calculus. In *Proceedings of IEEE INFOCOM*, 2017.

[43] Kai Lampka, Björn Forsberg, and Vasileios Spiliopoulos. Keep it cool and in time: With runtime monitoring to thermal-aware execution speeds for deadline constrained systems. *Journal of Parallel and Distributed Computing*, 95:79–91, 2016.

[44] Srivatsan Laxman and P Shanti Sastry. A survey of temporal data mining. *Sadhana*, 31(2):173–198, 2006.

[45] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

[46] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE, 1989.

[47] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

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

[49] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In *SDM*, volume 7, pages 273–284. SIAM, 2007.

[50] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

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

[52] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[53] David Lo, Siau-Cheng Khoo, and Chao Liu. Efficient mining of recurrent rules from a sequence database. In *Database Systems for Advanced Applications*, pages 67–83. Springer, 2008.

[54] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 104(5):1013–1024, May 2016.

[55] H. Lu, Y. Li, S. Mu, D. Wang, H. Kim, and S. Serikawa. Motor anomaly detection for unmanned aerial vehicles using reinforcement learning. *IEEE Internet of Things Journal*, pages 1–1, 2017.

[56] Embedded Trace Macrocell. Architecture specification. *Arm Limited*, 2007.

[57] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[58] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[59] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys (CSUR)*, 48(1):12, 2015.

[60] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295, 2014.

[61] Mary Natrella. Nist/sematech e-handbook of statistical methods. 2010.

[62] John Neter. Applied linear regression models.

[63] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *IEEE 34th Real-Time Systems Symposium (RTSS)*, pages 88–96, Dec 2013.

[64] Moritz Neukirchner, Kai Lampka, Sophie Quinton, and Rolf Ernst. Multi-mode monitoring for mixed-criticality real-time systems. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*, pages 1–10. IEEE, 2013.

[65] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and information systems*, 45(3):535–569, 2015.

[66] Hubert Nguyen. *GPU Gems 3*. Addison-Wesley Professional, first edition, 2007.

[67] Simon Perathoner, Tobias Rein, Lothar Thiele, Kai Lampka, and Jonas Rox. Modeling structured event streams in system level performance analysis. In *ACM Sigplan Notices*, volume 45, pages 37–46. ACM, 2010.

[68] A.D. Pimentel, L.O. Hertzbetger, P. Lieverse, P. van der Wolf, and E.F. Deprettere. Exploring embedded-systems architectures with artemis. *Computer*, 34(11):57–63, Nov 2001.

[69] Sasikumar Punnekkat, Rob Davis, and Alan Burns. Sensitivity analysis of real-time task sets. *Advances in Computing Science—ASIAN'97*, pages 72–82, 1997.

[70] Yan Qiao, XW Xin, Yang Bin, and S Ge. Anomaly intrusion detection method based on hmm. *Electronics Letters*, 38(13):1, 2002.

[71] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[72] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018.

[73] Razvan Racu, Marek Jersak, and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 160–169. IEEE, 2005.

[74] Benann Rajendra and Nirmal Joshi. A study on the acceleration of arrival curve construction and regular specification mining using gpus. Master's thesis, University of Waterloo, 2018.

[75] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models - The SymTA/S Approach*. PhD thesis, Technical University Carolo-Wilhelmina of Braunschweig, 2005.

[76] BD Ripley. *Modern applied statistics with S*. 2002.

[77] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. End-to-end communication delay analysis in industrial wireless networks. *IEEE Transactions on Computers*, 64(5):1361–1374, May 2015.

[78] Mahmoud Salem, Mark Crowley, and Sebastian Fischmeister. Anomaly Detection using Inter-arrival Curves for Real-time Systems. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[79] Stan Salvador, Philip Chan, and John Brodie. Learning states and rules for time series anomaly detection. In *FLAIRS Conference*, pages 306–311, 2004.

[80] Benjamin Schleich, Nabil Anwer, Luc Mathieu, and Sandro Wartzack. Shaping the digital twin for design and production engineering. *CIRP Annals*, 66(1):141–144, 2017.

[81] Scott Shenker and John Wroclawski. General characterization parameters for integrated service network elements. Technical report, 1997.

[82] Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 57–67. IEEE, 2004.

[83] Marco Spuri. Analysis of deadline scheduled real-time systems. 1996.

[84] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[85] Ming-Yang Su. Discovery and prevention of attack episodes by frequent episodes mining and finite state machines. *Journal of Network and Computer Applications*, 33(2):156–167, 2010.

[86] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(5):516–524, 2010.

[87] Steve Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4):308–317, 1994.

[88] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer*, 8(6):649–667, 2006.

[89] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Online system problem detection by mining patterns of console logs. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 588–597. IEEE, 2009.

[90] 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.

[91] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Mihai Christodorescu, and Lui Sha. Learning execution contexts from system call distribution for anomaly detection in smart embedded system. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 191–196, 2017.

[92] Mohammad Mehdi Zeinali Zadeh, Mahmoud Salem, Neeraj Kumar, Greta Cutulenco, and Sebastian Fischmeister. SiPTA: Signal Processing for Trace-based Anomaly Detection. In *Proceedings of the 14th International Conference on Embedded Software (EMSOFT)*, 2014.

[93] Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with edf scheduling. *IEEE Transactions on Computers*, 58(9):1250–1258, 2009.

[94] Fengxiang Zhang, Alan Burns, and Sanjoy Baruah. Sensitivity analysis for edf scheduled arbitrary deadline real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, pages 61–70. IEEE, 2010.

[95] Fengxiang Zhang, Alan Burns, and Sanjoy Baruah. Sensitivity analysis of task period for edf scheduled arbitrary deadline real-time systems. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 3, pages 23–28. IEEE, 2010.

[96] Fengxiang Zhang, Alan Burns, and Sanjoy Baruah. Task parameter computations for constraint deadline real-time systems with edf scheduling. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 3, pages V3–553. IEEE, 2010.

[97] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 12(2):159–170, 2010.

Appendix A: Maple Proof for Robustness Formulas

 # The following is a machine-proof using Maple software to the presented derivations in Chapter 6.

 # For the task-model used in Chapter 6, we consider a sporadic task with no overloads, so execution time cant exceed the period of the task which is by definition a positive value.

$p > 0$

$$0 < p \tag{1}$$

$0 < e \le p$

$$0 < e \le p \tag{2}$$

 # In addition, we assume the task model has implicit deadlines which means the deadline of the task aligns with its period.

$d = p$

$$d = p \tag{3}$$

 # First, we define a nominal demand-bound function as follows:

$dbf = \text{floor}\left(\frac{(t + p - d)}{p} \right) \cdot e$

$$dbf = \left\lfloor \frac{t + p - d}{p} \right\rfloor e \tag{4}$$

 # We define alpha as the decrease of the period of a task as in Definition 8
$-\text{infinity} < \text{alpha} < p$

$$-\infty < \alpha < p \tag{5}$$

 # we define beta as the increase in execution time as in Definition 9
$-e < \text{beta} < ((p - \text{alpha}) - e)$

$$-e < \beta < p - \alpha - e \tag{6}$$

 # We define the altered task model which still considers implicit deadlines and no overload. The demand-bound function
can be defined as follows :

$dbfalt = \text{floor}\left(\frac{(t + p - \text{alpha} - (d - \text{alpha}))}{p - \text{alpha}} \right) \cdot (e + \text{beta})$

$$dbfalt = \left\lfloor \frac{t + p - d}{p - \alpha} \right\rfloor (e + \beta) \tag{7}$$

 # Now we define sigma as the variation in the demand due to the change in the task parameters as in Definition 10

$\text{sigma} = dbfalt - dbf$

$$\sigma = dbfalt - dbf \tag{8}$$

#Using the above definitions, we obtain the realtion of beta with the other task variation parameters and time.

subs({**(4)**, **(7)**}, **(8)**)

$$\sigma = \left\lfloor \frac{t+p-d}{p-\alpha} \right\rfloor (e+\beta) - \left\lfloor \frac{t+p-d}{p} \right\rfloor e \qquad \textbf{(9)}$$

subs(**(3)**, **(9)**)

$$\sigma = \left\lfloor \frac{t}{p-\alpha} \right\rfloor (e+\beta) - \left\lfloor \frac{t}{p} \right\rfloor e \qquad \textbf{(10)}$$

#To obtain the equation for beta as in Equation 6.4:
solve(**(10)**, [beta])[1][1]

$$\beta = -\frac{\left\lfloor -\frac{t}{-p+\alpha} \right\rfloor e - \left\lfloor \frac{t}{p} \right\rfloor e - \sigma}{\left\lfloor -\frac{t}{-p+\alpha} \right\rfloor} \qquad \textbf{(11)}$$

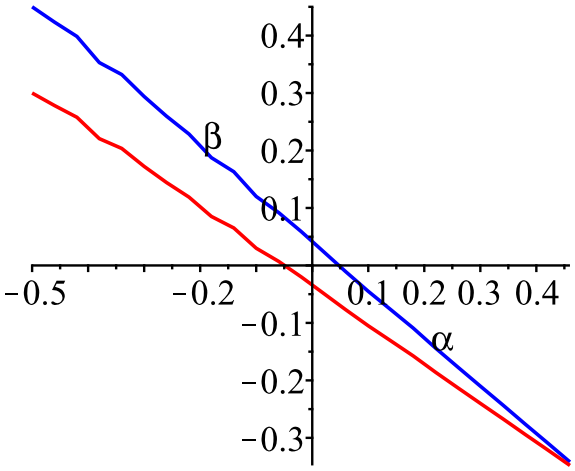
 # We validate the results of Example 5:

eval(**(11)**, {$t=30$, $p=0.5$, $e=0.375$})

$$\beta = -\frac{0.375 \left\lfloor -\frac{30}{-0.5+\alpha} \right\rfloor - 22.500 - \sigma}{\left\lfloor -\frac{30}{-0.5+\alpha} \right\rfloor} \qquad \textbf{(12)}$$

with(*plots*, *implicitplot*)
implicitplot([*eval*(**(12)**, sigma = 2.25), *eval*(**(12)**, sigma = -2.25)], alpha = -0.5 ..0.5, beta = -0.5 ..0.5,
color = [*blue*, *red*])

[*implicitplot*]



 # We now verify the proofs in Section 6.5 as follows:

#Asymptotic analysis for beta: in other words for a given alpha , we study the relation between beta and time

#First, we define the relation at a given value of alpha which is the input of Equation 6.8
eval(**(11)**, alpha = *alphaconst*)

$$\beta = -\frac{\left\lfloor -\frac{t}{-p + alphaconst} \right\rfloor e - \left\lfloor \frac{t}{p} \right\rfloor e - \sigma}{\left\lfloor -\frac{t}{-p + alphaconst} \right\rfloor} \tag{13}$$

#Using maple , you can obtain the result of Equation 6.8 directly using limits as follows: (the asymptotic analysis as the limit of beta as t approaches infinity.)

Basympt = *limit*(*rhs*(**(13)**), *t* = infinity)

$$Basympt = -\frac{e\, alphaconst}{p} \tag{14}$$

Here we follow the mathematical approach presented in Equation 6.8 to verify the result as well.

#Add first replacement term
eval(**(13)**, *t* = *c*·(*p* - *alphaconst*))

$$\beta = -\frac{\left\lfloor -\frac{c\,(p - alphaconst)}{-p + alphaconst} \right\rfloor e - \left\lfloor \frac{c\,(p - alphaconst)}{p} \right\rfloor e - \sigma}{\left\lfloor -\frac{c\,(p - alphaconst)}{-p + alphaconst} \right\rfloor} \tag{15}$$

#Add second replacement term
eval(**(15)**, *alphaconst* = *k*·*p*)

$$\beta = -\frac{\left\lfloor -\frac{c\,(-k\,p + p)}{k\,p - p} \right\rfloor e - \left\lfloor \frac{c\,(-k\,p + p)}{p} \right\rfloor e - \sigma}{\left\lfloor -\frac{c\,(-k\,p + p)}{k\,p - p} \right\rfloor} \tag{16}$$

simplify(**(16)**)

$$\beta = \frac{-\lfloor c \rfloor e + \lfloor -c\,(k-1) \rfloor e + \sigma}{\lfloor c \rfloor} \tag{17}$$

#add the floor operation approximations
eval(**(17)**, {beta = *Basympt* , floor(-*c*·(*k* − 1)) = - *c*·(*k* − 1) + O(1), floor(*c*) = *c* + O(1) })

$$Basympt = \frac{-(c + O(1))\, e + (-c\,(k-1) + O(1))\, e + \sigma}{c + O(1)} \tag{18}$$

simplify(**(18)**, *eliminate*)

$$Basympt = \frac{-c\,e\,k + \sigma}{c + O(1)} \tag{19}$$

#The result of Case 1 can be obtained as follows:
Basympt = $\lim_{c \to \infty}$ *rhs*(**(17)**)

$$Basympt = -e\,k \tag{20}$$

#Consider Case 2 where sigma can be defined as a function of the nominal dbf as in Equation 6.9

sigmafract = *f*· *dbf*

$$sigmafract = f\, dbf \tag{21}$$

subs(**(3)**, *subs*(**(4)**, **(21)**))

$$sigmafract = f\left\lfloor \frac{t}{p} \right\rfloor e \tag{22}$$

eval(**(22)**, {*t* = *c*·(*p* - *alphaconst*)})

$$sigmafract = f\left\lfloor \frac{c\,(p - alphaconst)}{p} \right\rfloor e \tag{23}$$

eval(**(23)**, *alphaconst* = *k*·*p*)

$$sigmafract = f\left\lfloor \frac{c\,(-k\,p + p)}{p} \right\rfloor e \tag{24}$$

simplify(**(24)**)

$$sigmafract = f\lfloor -c\,(k-1)\rfloor e \tag{25}$$

#now we return to the case where we got asymptotic value of beta
eval(**(19)**, sigma = *sigmafract*)

$$Basympt = \frac{-c\,e\,k + sigmafract}{c + O(1)} \tag{26}$$

subs(**(25)**, **(26)**)

$$Basympt = \frac{-c\,e\,k + f\lfloor -c\,(k-1)\rfloor e}{c + O(1)} \tag{27}$$

subs({beta = *Basympt*, floor(-*c*·(*k* - 1)) = (-*c*·(*k* - 1) + O(1))}, **(27)**)

$$Basympt = \frac{-c\,e\,k + f\,(-c\,(k-1) + O(1))\,e}{c + O(1)} \tag{28}$$

simplify(**(28)**)

$$Basympt = -\frac{e\,(-O(1)\,f + c\,((k-1)\,f + k))}{c + O(1)} \tag{29}$$

#We obtain the result in Equation 6.10 as follows:
Basympt = *limit*(*rhs*(**(29)**), *c* = infinity)

$$Basympt = -e\,f\,k + e\,f - e\,k \tag{30}$$

#We apply the results to the example shown earlier
eval(**(13)**, {*alphaconst* = 0.04, *p* = 0.5, *e* = 0.375})

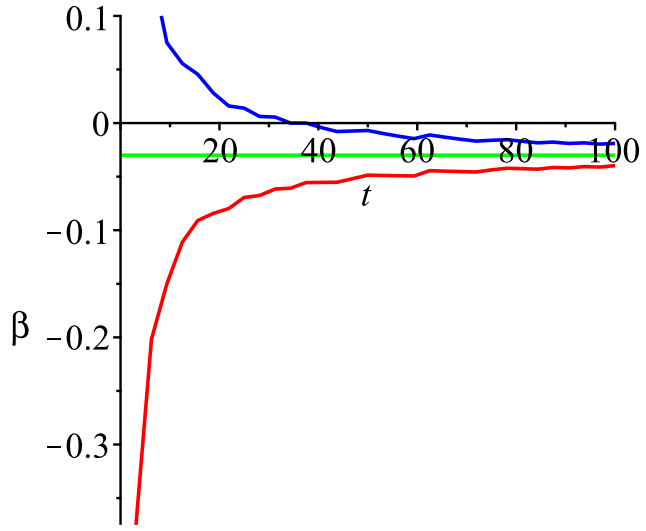
$$\beta = -\frac{0.375\,\lfloor 2.173913043\,t\rfloor - 0.375\,\lfloor 2.000000000\,t\rfloor - \sigma}{\lfloor 2.173913043\,t\rfloor} \tag{31}$$

eval(**(14)**, {*alphaconst* = 0.04, *p* = 0.5, *e* = 0.375})

$$Basympt = -0.03000000000 \tag{32}$$

with(*plots*, *implicitplot*)
implicitplot([*eval*(**(31)**, sigma = 2.25), *eval*(**(31)**, sigma = -2.25), *subs*(**(32)**, *Basympt* = beta)], *t* = 0 .. 100, beta = -0.4 ..0.1, *color* = [*blue*, *red*, *green*], *numpoints* = 1000)

[*implicitplot*]



#We verify the example on the second case where we say that the fraction of sigma up is 1.1 and the fraction of sigma down is 0.9 of the nominal demand

eval(**(13)**, sigma = *sigmafract*)

$$\beta = -\frac{\left\lfloor -\frac{t}{-p+alphaconst}\right\rfloor e - \left\lfloor \frac{t}{p}\right\rfloor e - sigmafract}{\left\lfloor -\frac{t}{-p+alphaconst}\right\rfloor} \tag{33}$$

subs(**(22)**, **(33)**)

$$\beta = -\frac{\left\lfloor -\frac{t}{-p+alphaconst}\right\rfloor e - \left\lfloor \frac{t}{p}\right\rfloor e - f\left\lfloor \frac{t}{p}\right\rfloor e}{\left\lfloor -\frac{t}{-p+alphaconst}\right\rfloor} \tag{34}$$

eval(**(34)**, {*alphaconst* = 0.04, *p* = 0.5, *e* = 0.375})

$$\beta = -\frac{0.375\,\lfloor 2.173913043\,t\rfloor - 0.375\,\lfloor 2.000000000\,t\rfloor - 0.375\,f\lfloor 2.000000000\,t\rfloor}{\lfloor 2.173913043\,t\rfloor} \tag{35}$$

$$eval\left(\textbf{(29)}, \left\{k = \frac{alphaconst}{p}\right\}\right)$$

$$Basympt = -\frac{e\left(-O(1)\,f + c\left(\left(\frac{alphaconst}{p} - 1\right)f + \frac{alphaconst}{p}\right)\right)}{c + O(1)} \tag{36}$$

limit(**(36)**, *c* = infinity)

$$Basympt = \frac{e\,f\,p - e\,f\,alphaconst - e\,alphaconst}{p} \tag{37}$$

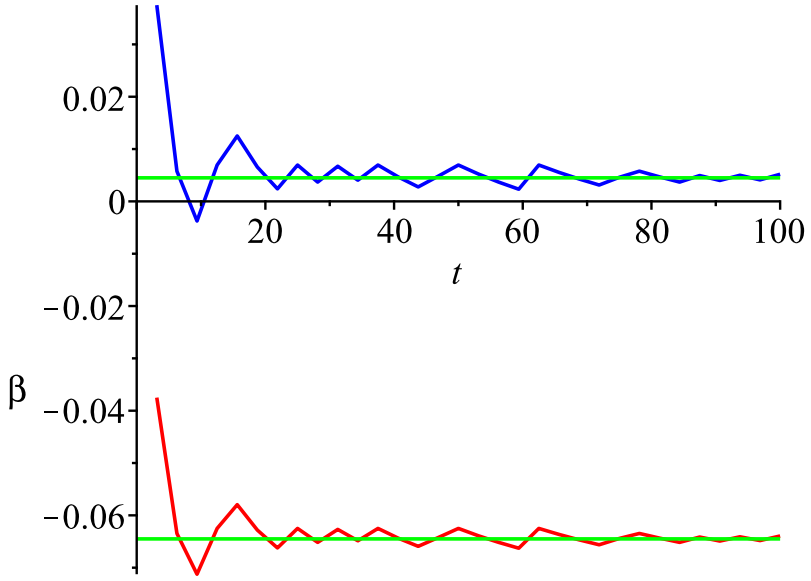
eval(**(37)**, {*alphaconst* = 0.04, *p* = 0.5, *e* = 0.375})

$$Basympt = 0.3450000000\, f - 0.03000000000 \quad \textbf{(38)}$$

with(*plots*, *implicitplot*)

implicitplot([*eval*(**(35)**, *f* = 0.1), *eval*(**(35)**, *f* = −0.1), *eval*(**(38)**, {*Basympt* = beta, *f* = 0.1}), *eval*(**(38)**, {*Basympt* = beta, *f* = −0.1})], *t* = 0 .. 100, beta = −0.4 ..0.4, *color* = [*blue*, *red*, *green*, *green*], *numpoints* = 1000)

[*implicitplot*]



#Section 6.5.2 Asymptotic analysis for alpha: for a given beta , how alpha changes over time

The issue here is the floor operator as the inverse does not exist.

$$\text{floor}\left(\frac{t}{p - \text{alpha}}\right) = Z$$

$$\left\lfloor \frac{t}{p - \alpha} \right\rfloor = Z \quad \textbf{(39)}$$

eval(**(10)**, **(39)**)

$$\sigma = Z\,(e + \beta) - \left\lfloor \frac{t}{p} \right\rfloor e \quad \textbf{(40)}$$

solve(**(40)**, [*Z*])[1][1]

$$Z = \frac{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma}{e + \beta} \quad \textbf{(41)}$$

#adding the floor property
#**if** floor$(x) = m$ **then** $m \le x < m + 1$ **end if**

#the greater than part

$$Z \le \frac{t}{p - \text{alpha}}$$

$$Z \le \frac{t}{p - \alpha} \tag{42}$$

map(`·`, **(42)**, $p -$ alpha) assuming $p \ne$ alpha

$$Z\,(p - \alpha) \le t \tag{43}$$

solve(**(43)**, alpha) assuming $t ::$ *positive*, $p \ne$ alpha

$$\begin{cases} \left[\left\{\alpha \le \frac{Zp - t}{Z}\right\}\right] & Z < 0 \\ [\{\alpha = \alpha\}] & Z = 0 \\ \left[\left\{\frac{Zp - t}{Z} \le \alpha\right\}\right] & 0 < Z \end{cases} \tag{44}$$

#the less than part

$$\frac{t}{p - \text{alpha}} < Z + 1$$

$$\frac{t}{p - \alpha} < Z + 1 \tag{45}$$

map(`·`, **(45)**, $p -$ alpha) assuming $p \ne$ alpha

$$t < (Z + 1)\,(p - \alpha) \tag{46}$$

solve(**(46)**, alpha) assuming $t ::$ *positive*, $p \ne$ alpha

$$\begin{cases} \left[\left\{\frac{Zp + p - t}{Z + 1} < \alpha\right\}\right] & Z < -1 \\ [\,] & Z = -1 \\ \left[\left\{\alpha < \frac{Zp + p - t}{Z + 1}\right\}\right] & -1 < Z \end{cases} \tag{47}$$

#Find aysmptotic value for alpha at constant beta
eval(**(11)**, beta = *betaconst*)

$$betaconst = -\frac{\left\lfloor -\frac{t}{-p + \alpha}\right\rfloor e - \left\lfloor \frac{t}{p}\right\rfloor e - \sigma}{\left\lfloor -\frac{t}{-p + \alpha}\right\rfloor} \tag{48}$$

#To workaround the floor operator, we first define both floor terms into new variables then we find the values asymptotically.

$$\mathrm{floor}\left(-\frac{t}{\alpha - p}\right) = fl1$$

$$\left\lfloor -\frac{t}{-p+\alpha} \right\rfloor = fl1 \tag{49}$$

subs(**(49)**, **(48)**)

$$betaconst = -\frac{fl1\, e - \left\lfloor \frac{t}{p} \right\rfloor e - \sigma}{fl1} \tag{50}$$

solve(**(50)**, *fl1*)

$$\frac{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma}{e + betaconst} \tag{51}$$

asympt(**(51)**, *t*)

$$\frac{e\, t}{p\,(e + betaconst)} + \mathrm{O}(1) \tag{52}$$

asympt(*lhs*(**(49)**), *t*)

$$-\frac{t}{-p+\alpha} + \mathrm{O}(1) \tag{53}$$

 # Using the asymptotes of both sides, we can obtain the asymptote value as follows:
simplify(**(52)** = **(53)**) assuming *t* :: *positive*

$$\frac{e\, t}{p\,(e + betaconst)} + \mathrm{O}(1) = \frac{t}{p - \alpha} + \mathrm{O}(1) \tag{54}$$

#We obtain the result in Equation 6.17 of Case 3:
Aasympt = *solve*(**(54)**, alpha)

$$Aasympt = -\frac{p\, betaconst}{e} \tag{55}$$

#Case 4 where the demand variation sigma is function of nominal demand.
#Again, we work on both sides of the inequality describing the boundary

$$alpha1 = subs\left(Z = \frac{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma}{e + betaconst},\ \frac{Zp - t}{Z}\right)$$

$$\alpha 1 = \frac{\left(\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + \sigma\right) p}{e + betaconst} - t\right)(e + betaconst)}{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma} \tag{56}$$

$$alpha2 = subs\left(Z = \frac{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma}{e + betaconst},\ \frac{Zp + p - t}{Z + 1}\right)$$

$$\alpha 2 = \frac{\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + \sigma\right) p}{e + betaconst} + p - t}{\frac{\left\lfloor \frac{t}{p} \right\rfloor e + \sigma}{e + betaconst} + 1} \tag{57}$$

eval(**(56)**, sigma = *sigmafract*)

$$\alpha 1 = \frac{\left(\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + sigmafract\right) p}{e + betaconst} - t \right) (e + betaconst)}{\left\lfloor \frac{t}{p} \right\rfloor e + sigmafract} \tag{58}$$

eval(**(57)**, sigma = *sigmafract*)

$$\alpha 2 = \frac{\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + sigmafract\right) p}{e + betaconst} + p - t}{\frac{\left\lfloor \frac{t}{p} \right\rfloor e + sigmafract}{e + betaconst} + 1} \tag{59}$$

subs(**(22)**, **(58)**)

$$\alpha 1 = \frac{\left(\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + f\left\lfloor \frac{t}{p} \right\rfloor e\right) p}{e + betaconst} - t \right) (e + betaconst)}{\left\lfloor \frac{t}{p} \right\rfloor e + f\left\lfloor \frac{t}{p} \right\rfloor e} \tag{60}$$

subs(**(22)**, **(59)**)

$$\alpha 2 = \frac{\frac{\left(\left\lfloor \frac{t}{p} \right\rfloor e + f\left\lfloor \frac{t}{p} \right\rfloor e\right) p}{e + betaconst} + p - t}{\frac{\left\lfloor \frac{t}{p} \right\rfloor e + f\left\lfloor \frac{t}{p} \right\rfloor e}{e + betaconst} + 1} \tag{61}$$

#Now, we use the asymptotes of both sides to obtain the results as in Equations 6.18 and 6.19:
alpha1asympt = *asympt*(*rhs*(**(60)**), *t*)

$$alpha1asympt = \frac{(f e - betaconst)\, p}{e\,(f+1)} + O\left(\frac{1}{t}\right) \tag{62}$$

alpha2asympt = *asympt*(*rhs*(**(61)**), *t*)

$$alpha2asympt = \frac{(f e - betaconst)\, p}{e\,(f+1)} + O\left(\frac{1}{t}\right) \tag{63}$$

#both boundaries upper and lower and approximation will reach same asymptotic value as shown in Equation 6.2
limit(**(62)**, *t* = infinity)

$$alpha1asympt = \frac{e f p - p\, betaconst}{f e + e} \tag{64}$$

limit(**(63)**, $t = \text{infinity}$)

$$alpha2asympt = \frac{e\,f\,p - p\,betaconst}{f\,e + e} \tag{65}$$

#Applying the results of Section 6.5 to the example as previous:
eval(**(48)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

$$0.01 = -\frac{0.375\left|-\frac{t}{-0.5+\alpha}\right| - 0.375\,|2.000000000\,t| - \sigma}{\left|-\frac{t}{-0.5+\alpha}\right|} \tag{66}$$

eval(**(55)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

$$Aasympt = -0.01333333333 \tag{67}$$

eval(**(56)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

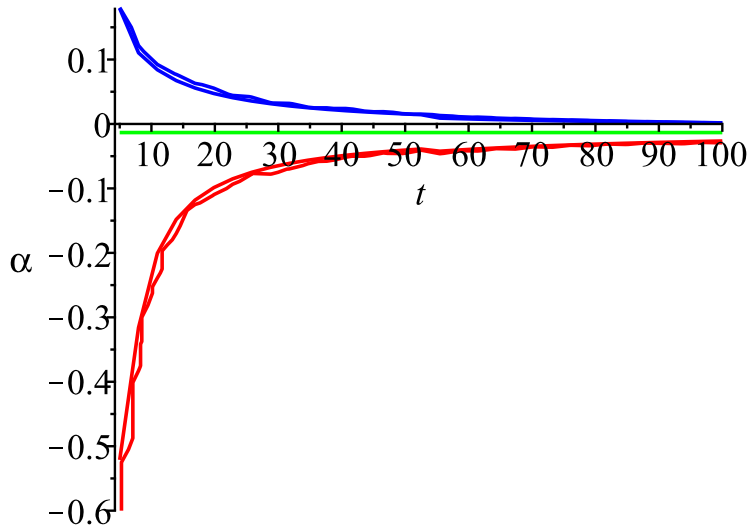
$$\alpha 1 = \frac{0.385\,(0.4870129868\,|2.000000000\,t| + 1.298701298\,\sigma - t)}{0.375\,|2.000000000\,t| + \sigma} \tag{68}$$

eval(**(57)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

$$\alpha 2 = \frac{0.4870129868\,|2.000000000\,t| + 1.298701298\,\sigma + 0.5 - t}{0.9740259739\,|2.000000000\,t| + 2.597402597\,\sigma + 1} \tag{69}$$

with(*plots*, *implicitplot*)
implicitplot([*eval*(**(66)**, sigma = 2.25), *eval*(**(68)**, {*alpha1* = alpha, sigma = 2.25}), *eval*(**(66)**, sigma = −2.25), *eval*(**(69)**, {*alpha2* = alpha, sigma = −2.25}), *subs*(**(67)**, *Aasympt* = alpha)], $t = 5 .. 100$, alpha = −0.6 ..0.6, *color* = [*blue*, *blue*, *red*, *red*, *green*], *numpoints* = 1000)

[*implicitplot*]



eval(**(60)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

$$\alpha 1 = \frac{0.385\,(0.4870129868\,|2.000000000\,t| + 0.4870129868\,f\,|2.000000000\,t| - t)}{0.375\,|2.000000000\,t| + 0.375\,f\,|2.000000000\,t|} \tag{70}$$

eval(**(61)**, {*betaconst* = 0.01, $p = 0.5$, $e = 0.375$})

$$\alpha 2 = \frac{0.4870129868\,\lfloor 2.000000000\,t \rfloor + 0.4870129868\,f\lfloor 2.000000000\,t \rfloor + 0.5 - t}{0.9740259739\,\lfloor 2.000000000\,t \rfloor + 0.9740259739\,f\lfloor 2.000000000\,t \rfloor + 1} \quad \textbf{(71)}$$

eval(**(64)**, {*betaconst* = 0.01, *p* = 0.5, *e* = 0.375})

$$alpha1asympt = \frac{0.1875\,f - 0.005}{0.375\,f + 0.375} \quad \textbf{(72)}$$

eval(**(65)**, {*betaconst* = 0.01, *p* = 0.5, *e* = 0.375})

$$alpha2asympt = \frac{0.1875\,f - 0.005}{0.375\,f + 0.375} \quad \textbf{(73)}$$

with(*plots*, *implicitplot*)

implicitplot([*eval*(**(70)**, {*alpha1* = alpha, *f* = 0.1}), *eval*(**(71)**, {*alpha2* = alpha, *f* = 0.1}), *eval*(**(70)**, {*alpha1* = alpha, *f* = −0.1}), *eval*(**(71)**, {*alpha2* = alpha, *f* = −0.1}), *eval*(**(72)**, {*alpha1asympt* = alpha, *f* = 0.1}), *eval*(**(73)**, {*alpha2asympt* = alpha, *f* = −0.1})], *t* = 10 .. 100, alpha = −0.6 ..0.6, *color* = [*blue*, *blue*, *red*, *red*, *green*, *green*], *numpoints* = 1000)

[*implicitplot*]