Non-Intrusive Program Tracing of Non-Preemptive Multitasking Systems Using Power Consumption

by

Kamal Lamichhane

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2017

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 Contributions

Chapter 3 and chapter 4 of this thesis have been adopted from the work [68] which is accepted for publication at the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE) in Dresden, Germany. Author of this thesis is the primary author of the accepted paper along with co-author Dr. Carlos Moreno and Associate Professor Sebastian Fischmeister. Carlos Moreno is a Senior Reseach Associate under the supervision of Associate Professor Sebastian Fischmeister in the Real-time Embedded Software Group at the University of Waterloo.

Hardware setup used in chapter 3 for data capture were contributed by Carlos Moreno, who also contributed his ideas, reviews, and suggestions. All the work presented in this thesis were completed under the supervision of Professor Sebastian Fischmeister in the ECE department at the University of Waterloo, who contributed with his ideas, reviews, and suggestions.

Abstract

System tracing, runtime monitoring, execution reconstruction are useful techniques for protecting the safety and integrity of systems. Furthermore, with time-aware or overhead-aware techniques being available, these techniques can also be used to monitor and secure production systems. As operating systems gain in popularity, even in deeply embedded systems, these techniques face the challenge to support multi-tasking.

In this thesis, we propose a novel non-intrusive technique, which efficiently reconstructs the execution trace of non-preemptive multitasking system by observing power consumption characteristics. Our technique uses the Control flow Graph (CFG) of the application program to identify the most likely block of code that the system is executing at any given point in time. For the purpose of the experimental evaluation, we first instrument the source code to obtain power consumption information of each Basic Block (BB), which is used as the training data for our Dynamic Time Warping (DTW) and k-Nearest Neighbors (k-NN) classifier. Once the system is trained, this technique is used to identify live code-block execution (LCBE). We show that the technique can reconstruct the execution flow of programs in a multi-tasking environment with high accuracy. To aid the classification process, we analyze eight widely used machine learning algorithms with time-series power-traces data and show the comparison of time and computational resources for all the algorithms.

Acknowledgements

Past twenty-eight months has been a period of intense learning for me, not only in the area of my research but also on a personal level. I would like to extend my gratitude to the people who have supported me throughout this period.

First and foremost I would like to express my gratitude to my supervisor Dr. Sebastian Fischmeister for his continuous guidance, inspiration, suggestions, and all the opportunitites I was given. Thank you for believing in me.

Secondly, I would like to extend my words of sincere appreciation to Dr. Carlos Moreno who has provided excellent guidance and support throughout this period. Big thanks!! to you for your patience; it was a great honor and a tremendous learning opportunity working with you.

I would like to thank all my course instructors for the invaluable knowledge that you shared. I would also like to acknowledge Sean Kauffman, Jack Morgan, and Jeet for their assistance and suggestions during this period. My words of acknowledgment to all the administrative staff at the University of Waterloo. Thank You!

Finally, there are my friends who not only discussed papers, problems and findings but also supported me throughout. I would like to thank all my friends for the support and suggestions in my work – Bismaya Sahoo, Hemant surale, Pratik Bhandary. thank you!!

Once again thank you!!! Sebastian Fischmeister for your guidance. I'm grateful for your support.

Dedication

This thesis is dedicated to my parents and sisters for their endless love, support, and encouragement.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ADPCM Adaptive Differential Pulse-code Modulation

BB Basic Block

CFG Control flow Graph

CMOS Complementary Metal-Oxide-Semiconductor

CRC Cyclic Redundancy Check

DT Decision Tree

DTW Dynamic Time Warping

IDS Intrusion Detection System

LCBE Live Code Block Execution

MCU Microcontroller Unit

MLP Multi-layer Perceptron

NB Naive Bayes

QDA Quadratic Discriminant Analysis

LDA Linear Discriminant Analysis

RF Random Forest

k-NN k-Nearest Neighbors

NN Nearest Neighbor

Chapter 1

Introduction

Microelectronic embedded systems are an important part of our daily lives, as they have been an important factor in many technological advances in recent decades. From handling tasks like controlling temperature in our refrigerators to complex tasks such as automating our cars, embedded systems have shaped the progress of our generation. Such an extensive outreach has demanded an increase in research devoted to the development and security of these systems. Embedded systems are deployed to run independently in a self-sufficient manner. Once deployed, these systems behave as a black box with no capacity for runtime debugging. This opacity of internal code execution offers the benefit of increased security. However, it also makes system tracing, runtime monitoring, and execution reconstruction challenging.

If the faulty behavior is observed in the production and/or deployment phase of product development cycle where developers are no longer able to make use of debugging tools, non-intrusive tracing is possibly the only available technique. Due to the dynamic nature of multitasking system, some runtime anomalies are non-recurrent in consecutive system executions and may be difficult to reproduce after recompiling or restarting the device. One popular workaround is to hard-code code snippets to aid in runtime debugging. However, it is often challenging to think ahead and exhaust all possible error scenarios prior to deployment. Further, due to memory constraints in deployed products, developer prefer omitting debugging information. Added instrumentation to obtain the runtime tracing could break extra-functional requirements of the system. These memory limitations and non-reproducible nature of run-time errors advocate the necessity of a non-intrusive run-time monitoring system to safeguard the integrity of modern electronic devices. Traditionally, run-time monitoring has been used to enforce safety properties in embedded systems. However, corrupted firmware in embedded system makes it difficult to assure the

system's safety. These problems advocate the necessity of an approach which can make the tracing and debugging easier, not only for application software but also for the system with corrupted firmware.

In this thesis, we propose a novel non-intrusive technique, which efficiently reconstructs the execution trace of non-preemptive multitasking system by observing power consumption characteristics. Our proposed technique can be used efficiently in the application of embedded system safety, to overcome the problems stated above. We capture the power consumed by the target platform as a time series data and classify those data to reconstruct the execution trace.

Time-series classification is of significant interest in the area of machine learning these days. Due to the usefulness of time-series representation of data in various domains such as agriculture, medicine, politics, and industry; the classification and prediction have gained in popularity to improve the performance of the applications. Time-series classification has been extensively used in domains such as anomaly detection, customer flow/demand, and political prediction since past decades. For instance, an accurate classifier for anomaly detection in car lock system might replace the requirement of security personnel looking for the cars. Such an extensive application of time-series classification has demanded more research devoted to this area. Along with this growth, however, there is increasing concern over scalable and highly accurate classification technique. Despite its application and usefulness, there are no standard classification techniques that can classify time-series data from any domain. The controversy about scientific evidence for classification accuracy on high-frequency time-series data has raged unabated for over a century. Proposed techniques are either scalable with low accuracy or accurate with low scalability. Specifically, time-series classification of very high-frequency data is rarely seen in the literature.

Additionally, in this thesis, we will explore the possibility of using different time-series classification algorithms in high-frequency power-trace data. The primary motive behind this analysis is to use the power-trace data in runtime monitoring, debugging and real-time anomaly detection of cyber-physical systems.

The proposed technique can be extended to a broad range of MCU platforms. For this work, we chose the AVR ATMega2560, which is extensively used in cyber-physical systems in recent decades. Although the MCU we used is simple which doesn't contain any cache, multicore, and pipelines, the technique can be used in the platform with cache, pipelines, and multicores. One limitation to keep in mind is that the exact same system should be trained(we will talk more about training power trace in chapter 3) in order to use that for execution reconstruction.

1.1 Related Work

In [2, 3], Moreno et al. showed a technique for non-intrusive program tracing and debugging through power side channel analysis where they use the power consumption characteristics of the MCU to identify blocks of code being executed. In [4], they improve performance through a compiler-assisted stage that maximizes distinguishability of traces for different blocks of code. All of these works lack multitasking support, which limits the practical applicability of the techniques. Liu et al. [5] have shown reasonable accuracy in execution tracking by using control flow graph and source code information. Though the accuracy is high, they capture the trace with a high-end oscilloscope with sampling frequency 1.25 GS/sec. It is shown that this method can be used in anomaly detection; however, having the oscilloscope to capture the data and classifying the traces offline limits the usefulness of this work in real-life systems. In [6], Eisenbarth et al. have presented side-channel disassembler technique to obtain the sequence of CPU instructions without using the source code information. Accuracy is the main limitation of this work which is too low for any practical environment. Msgna et al. [7] presented the idea of side channel control flow security in the embedded system where they collect several traces and calculate the mean of traces to minimize the inherent and ambient noise introduced by the measurement setup. Calculating the mean of all traces may work in the dedicated environment, but it is not suitable for a practical application where the processing must be done on-the-fly to ensure the security and reconstruct the execution trace of the program. Clark et al. [8] uses the behavior monitoring system of medical devices to model permissible behavior and detect deviation. That work, however, is limited to the simple and highly repetitive operation of the device. The usefulness of non-intrusive systems has been highlighted in [9] where authors have presented the hardware-assisted paradigm to extract properties of an embedded program through static program analysis and use them to secure the system.

In regards to time-series power data classification, Deng et al. [10], use ARMA models and use Euclidean distance as a distance like score of a model to perform a nearest neighbor classification. An early approach to time-series classification using qualitative and quantitative methods are presented by Bakshi et al. in [11]. An overview of time-series knowledge extraction, data classification, data clustering and relationship finding is presented in [12]. In this paper, the author provides the analysis in different types of similarity measures in time-series data. Authors also focus on the critical issue of measuring the similarity between two sequences where the ability to deal with noise in the data, amplitude differences and gap in time axis are the primary problems. Hidden Markov Model representations are used in [13]. In [14], Nanopoulos et al. use neural networks on statistical features to perform time series classification. In [15], Povinelli et al. use a

combination of phase space representation and Bayesian-based on Gaussian mixture models. Kadous et al. [16, 17] proposed the TClass method for time series classification with comprehensible descriptions for multivariate time series. They extract the global features of the time series and cluster them to train the C4.5 classifier. DTW distance like measure has gained popularity in recent years in time series classification. DTW algorithm is first introduced by [18], and being used in time series data mining and sequence classification [19, 20, 21, 22, 23]. [24] has shown the efficient online sequence learning using unsupervised convolutional the neural network, but lately, DTW has gained in popularity in time series classification with the introduction of 1-Nearest Neighbor and faster computation with tighter lower bound [25, 26, 27]. [28, 29, 30, 31] have shown that DTW combined with 1 Nearest Neighbor (NN) is exceptionally difficult to beat in time series classification.

1.2 Problem Statement and Assumptions

Our proposed technique addresses the following problem: given the power trace P and the CFG G of a system running a program in embedded system, determine the correct node $b \in G$ executing at a given time point k.

In the context of this work, we make the following assumptions:

- Known MCU: the system runs on a known processor model. This assumption is necessary since the relationship between power consumption and program execution depends on the processor design and implementation.
- Input Identification: all possible combination of execution paths are generated using random input initialization and multiple runs. We make sure that all valid paths are generated using input randomization. The power trace associated with each basic block may exhibit variations due to the context in which it is executed (the input data and state). Thus, we should train the system with power traces of a set of execution instances that is statistically representative of the power traces generation process.
- Components Involved: targeted platform does not have a cache or any other micro-architectural component. Additionally, we do not consider multicore execution in this work. Presence of such component would produce high variation in power consumption depending on the execution context.

- Control Flow Integrity: we do not consider cases such as random execution due to memory or stack corruption, undefined behaviour deriving from uninitialized pointers or in general invalid pointer operations, "system crashes", etc. We also assume that no active adversary tampers with the control flow using techniques such as code injection [32] or code reuse [33] through buffer overflow attacks.

1.3 Contributions

The main contribution of this thesis is related to the reconstruction of the execution trace of embedded software. Work presented in this thesis is useful in intrusion detection system and in debugging of embedded software, more specifically in the context of multitasking system.

- We address some important limitations of the current state-of-the-art by adding support for co-operative multitasking scheduling [1]. Our technique uses the control flow graph (CFG) of the application program along with the power trace to identify the most likely block of code that the system is executing at any given point in time. To this end, we use a statistical classification approach, with DTW as the distance metric used by the NN technique.
- In order to make the classification process of traces more accurate and faster with fewer resources, we analyze eight widely used machine learning algorithms in power-trace data. We discuss the time and computational requirement of different algorithm and trade-off for using them in context of our data.

1.4 Organization of the Thesis

The remaining of this thesis advances as follows: We discuss the background required for all the chapters in chapter 2. Chapter 3 presents the details on system model and chapter 4 presents the classification techniques in power-trace with regards to cooperative operating system. Chapter 5 includes discussion and future work followed by some concluding remarks.

[1] DATE-2018 Conference Paper –Tracing of non-preemptive multitasking system

Chapter 2

Background

In this section, we discuss the key concepts used in our approach. In section 2.1, concepts of the system model is discussed followed by background on machine learning algorithms in section 2.2.

2.1 System Model Concepts

In this section, we discuss the concepts that we will use in our system. We explain about the power consumption in microcontrollers in section 2.1.1, control flow graph in section 2.1.3, followed by multitasking systems basics and time series classification in section 2.1.4 and 2.1.5 respectively.

2.1.1 Power Trace Analysis

The power trace $P = \langle k, p_k \rangle$ is the time series representing the power consumption of a microcontroller as a function of time, with power being sampled periodically. where p_k is the measured (instantaneous) power consumption at time index $k \in \mathbb{Z}$.

The total power consumed by a microcontroller depends on the static power consumption and dynamic power consumption [34], which is a function of load capacitance, the frequency of operation, and supply voltage. Static power consumption is due to leakage current when all the inputs are at constant logic level and are not switching the level. Dynamic power consumption contributes significantly to the overall power consumption which

is due to switching of logic levels and charging and discharging of capacitive load [35]. Power consumed by the microcontroller is calculated as:

$$P_C = [(C_{pd} \times f_I \times V_{cc}^2) + (C_L \times f_o \times V_{cc}^2)] \tag{2.1}$$

where:
C_{pd} = power-consumption capacitance
f_I = input frequency i.e., frequency at which the device(CMOS) is switching from one logic state to another
f_o = output frequency i.e., switching frequency of capacitive-load
VCC = supply voltage (V)
C_L = load capacitance at output.

Power consumed $\propto$ load capacitance and switching frequency.
Since power consumed by microcontroller varies accordingly with the load, i.e., Complementary Metal-Oxide-Semiconductor (CMOS) transistor and component inside the controller, power consumption can be analyzed to extract the information about the data processing and operation being performed inside the microchip [36].

2.1.2 Basic Block (BB)

A basic block is a sequence of executable instruction with a single entry point at the beginning and exit point at the end and no branches in between. Whenever the first instruction in a basic block is executed, all the following instruction of that basic block runs exactly once in order. Basic block forms the vertices in a control flow graph.

```
void quicksort( int m, int n ) {
  int i, j, v, x;
  if ( n <= m ) return;
  i = m-1;
  j = n;
  v = a[n];
  while(1) {
    do i=i+1; while( a[i] < v );
    do j=j-1; while( a[j] > v );
    if ( i >= j ) break;
    x = a[i]; a[i] = a[j]; a[j] = x;
  }
  x = a[i]; a[i] =a [n]; a[n] = x;
  quicksort( m, j );
```

```
15   quicksort( i+1, n );
16 }
```

Listing 2.1: Quick Sort

IR code of the quick sort from Listing 2.1 is:

(1) i := m-1
(2) j := n
(3) t1 := 4*n
(4) v := a[t1]
L0: L1:
(5) i := i+1
(6) t2 := 4*i
(7) t3 := a[t2]
(8) if t3<v goto L1
L2:
(9) j := j-1
(10) t4 := 4*j
(11) t5 := a[t4]
(12) if t5>v goto L2
(13) if i>=j goto L3
(14) t6 := 4*i
(15) x := a[t6]
(16) t7 := 4*i
(17) t8 := 4*j
(18) t9 := a[t8]
(19) a[t7] := t9
(20) t10 := 4*j
(21) a[t10] := x
(22) goto L0
L3:
(23) t11 := 4*i
(24) x := a[t11]
(25) t12 := 4*i
(26) t13 := 4*j
(27) t14 := a[t13]
(28) a[t12] := t14
(29) t15 := 4*j
(30) a[t15] := x
(31) 2 calls ...

Basic Blocks:
BB1: (1)- -(4)
BB2: (5)- -(8)
BB3: (9)- -(12)
BB4: (13)
BB5: (14)- -(22)
BB6: (23)- -(30)

2.1.3 Control Flow Graph (CFG)

The Control Flow Graph $G = \langle V, E \rangle$ is a directed graph which represents the execution flow of the program. Where V is the set of vertices representing BBs of code, and E is the set of edges — an edge from block $BB1$ to block $BB2$ ($BB1, BB2 \in V$) indicates that execution of block $BB2$ can immediately follow the execution of block $BB1$. A BB

is the sequence of executable instruction with a single entry point at the beginning and exit point at the end [37]. Instructions within the BB always execute in sequential order. Execution flow transfers to the different BB depending on the last branching instruction executed. Figure 2.1 shows an example of CFG. Power consumed by the microcontroller is dependent on the execution flow of the program being executed. Let us consider the control flow graph in Figure 2.1. There are six BBs in the program, BB1, BB2, BB3, BB4, BB5 and BB6. The last statement executed in the first BB decides the next BB to execute. If the condition in the last statement of BB4 is true the execution transit to BB5 else execution steps to BB6. CFG in Figure 2.1 can be directly related to the embedded devices which run the code repetitively with different execution path.
The power consumed by each BB is different [7]. We can instrument the source code and capture the power consumed of each BB. Although we cannot directly observe the control flow transition of the programs running on the device, we can obtain the control flow transition with power side channel analysis [38]. Given the CFG of the code running in the MCU, normal execution always follows the CFG - any deviation from normal execution can be traced by analyzing the CFG [39]. Since we know the control flow graph and the power trace of all the nodes/BB on the program, any external interfering program will have significantly different control flow path and different power traces, hence assisting in anomaly detection. Structured control flow graph can be used to classify the malware execution in the system accurately and to reveal the hidden code [40].

2.1.4 Multitasking Systems – Interrupts and Context Switching

Our work addresses multitasking systems where a scheduler switches contexts, giving each task the illusion that it is the only task running on the processor. There are two main categories of scheduling: cooperative and preemptive multitasking. Cooperative scheduling system uses the CPU until the task is either terminated or it voluntarily gives up control of the processor. In preemptive systems, the scheduler can asynchronously remove control of the processor from the task that is running. Context switching is the mechanism of storing and restoring the state of a task such that it can be resumed later when the scheduler decides to assign the processor to said task. Interrupts in multitasking systems introduce non-linear control flow behavior, making tracing the interrupts in the real-time embedded system difficult [41].

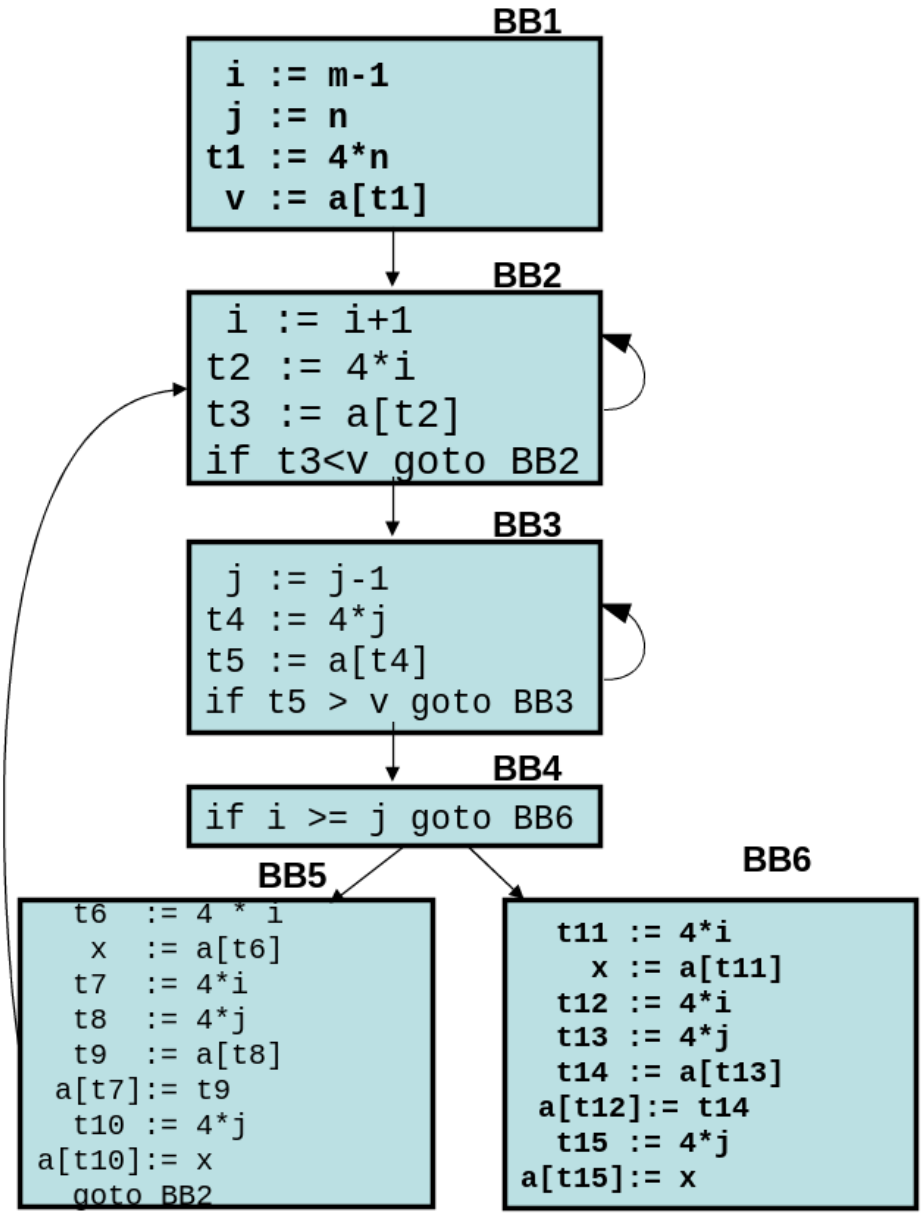


Figure 2.1: An example control flow graph

2.1.5 Time Series Classification

Time series classification can be interpreted in two ways. First one is, time-series classification and the second one is time-point classification. They are different considering the fact that earlier one has one label for one complete time series while the later one has labels for each time point. Given a database of time series and a single label for each time series, time-series classification classifies the future time-series that is statistically similar to training time series. Time-point classification takes a time series and label of each time point and classifies the future time point. e.g., customer demand, daily weather(snow, rain, sun) are time-point classification problems while daily power consumption, ECG, human movement (walking, running, sitting) classification, etc. are time-series classification. Similarity measures of time series data are divided into four categories [42]; Figure 2.2 shows the general view of distance measures in time series data. Shape-based methods compare the shape/appearance of the time series while feature-based methods extract statistical features that are representative of the time series. Model-based methods fit a model to generalize the time series and compare test time series to classify. The compression-based method looks at the compression of concatenated time series.

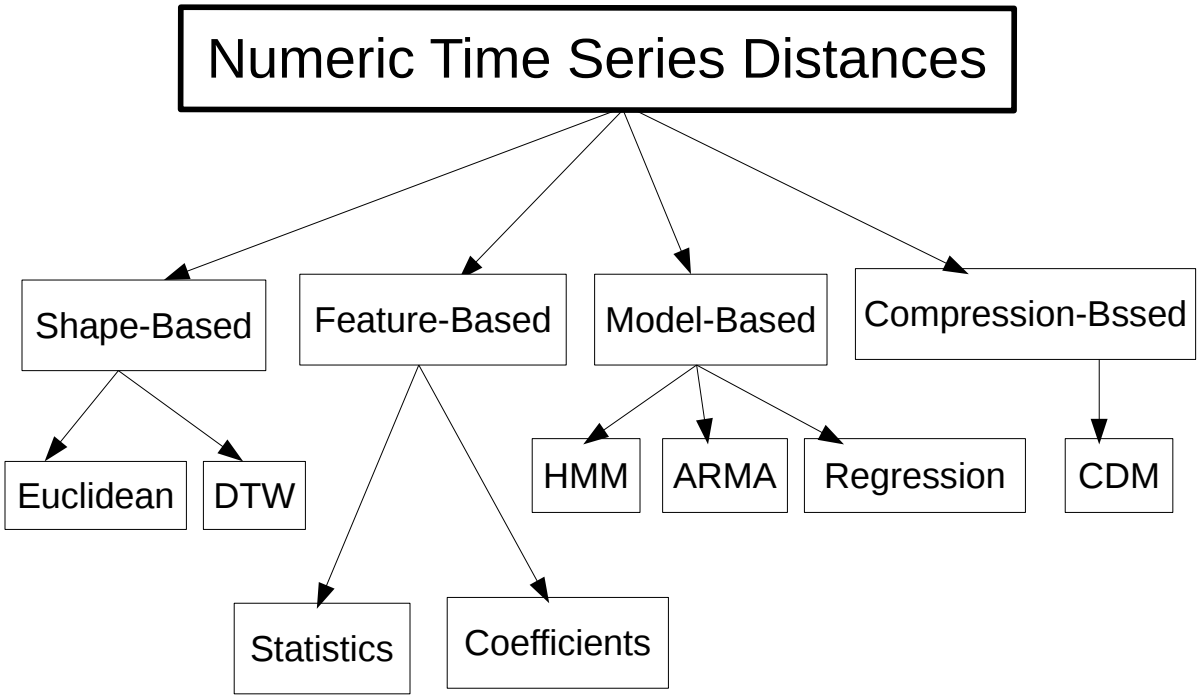


Figure 2.2: Categories of numeric time-series distances

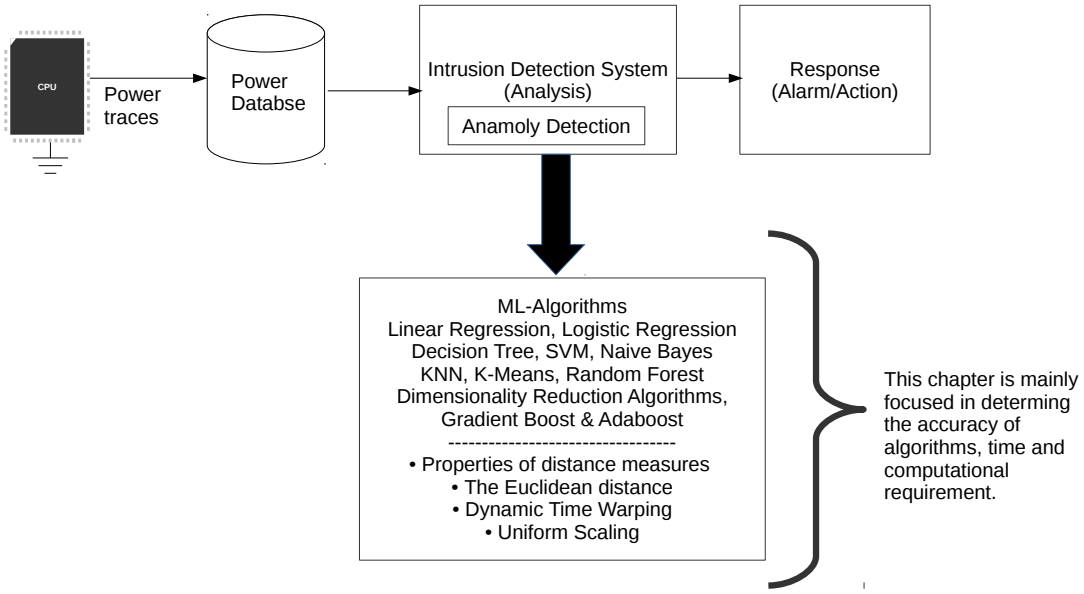


Figure 2.3: Possible use case (IDS)

2.2 Machine Learning Algorithms for Time Series Classification

Machine Learning[1] is a field of computer science that gives computers the ability to learn without being explicitly programmed [45] [43]. Alternatively, machine learning a.k.a predictive learning explores the study of algorithms that can learn and make the prediction from data. It is a data-driven prediction/classification approach, through building a model from a series of past data. Predictive modeling can be divided into two classifications approaches –Regression and pattern classification. Regression models are based on the analysis of relationships between variables and trends in order to make predictions about continuous variables while pattern classification assigns discrete class labels to particu-

[1]Major portion of this background concept is extracted from [43], and API's are used from [44, 1].

lar observations. Pattern classification can be analyzed in two subcategories: supervised learning and unsupervised learning. In supervised learning, dataset and class labels of the respective data-set are known in advance while in unsupervised learning prediction/classification should be made on unlabelled instances of data samples.

2.2.1 Types of machine learning

$$\begin{cases} \text{Supervised learning} \begin{cases} \text{Classification} \\ \text{Regression} \end{cases} \\ \text{Unsupervised learning} \begin{cases} \text{Discovering clusters} \\ \text{Discovering latent factors} \\ \text{Discovering graph structure} \end{cases} \end{cases}$$

Machine learning basically has 3 elements [46]:
ML Model = Representation + Evaluation + Optimization
A model is represented as a conditional probability distribution $P(y|\vec{x})$(classifier) or a decision function $f(x)$. The set of distribution or decision function is the hypothesis space of the model.

2.2.2 k-Nearest Neighbors

k-NN is a simple instance based non-parametric learning algorithm also known as lazy algorithms [47] which is based on the principle that the dataset with similar properties lies in the proximity. Lazy-learning algorithms require less computation time during the training phase than eager-learning algorithms but more computation time during classification [48]. The k-NN finds the k nearest instances to the test instance and determines its class by identifying the single most frequent class using distance metric. Several distance metrics have been proposed and tested over the years. Distance metric used in the classification affects the classification accuracy of the model. Some of the k-NN distance metrics are:

- Minkowsky:

$$d((x_1, ...x_n), (y_1, ..., y_n)) = \sum_{i=1}^{n-1} (x_i - y_i)^2 - (x_n - y_n)^2 \tag{2.2}$$

- Manhattan:

$$d((x_1, ...x_n), (y_1, ..., y_n)) = \sum_{i=1}^{n-1} |x_i - y_i| = \sum_{i=1}^{n-1} |y_i - x_i| \tag{2.3}$$

- Euclidean:

$$d((x_1, ...x_n), (y_1, ..., y_n)) = \sum_{i=1}^{n-1} |x_i - y_i| = \{\sum_{i=1}^{n-1} |y_i - x_i|\}^{1/2} \tag{2.4}$$

2.2.3 Naive Bayes Classifier

Naive Bayes classifier is a linear classifier which is based on Bayes probability theorem [49], and the adjective "naive" comes from the fact that the features in a dataset are mutually independent. The assumptions of mutually independent feature can lead to poor performance of NB. This model cannot accurately predict the non-linear dataset because of unrealistic independence assumption [50]. The base probability model of NB classifier is Bayes' Theorem which can be summarised as:

$$\text{posterior probability} = \frac{\text{conditional probability} \cdot \text{prior probability}}{\text{evidence}} \tag{2.5}$$

It calculates the probability of a sample being in a particular class i given its observed feature values.

Let x_i be the feature vector of class sample i, $i \in \{1, 2, ..., n\}$; ω_j be the representation of class j $j \in \{1, 2, ..., m\}$ and $P(x_i \mid \omega_j)$ be the probablity observing a sample x_i given that it belongs to class ω_j. Now, as inferred from equation 2.5 posterior probablity can be written as:

$$P(\omega_j \mid \mathbf{x}_i) = \frac{P(\mathbf{x}_i \mid \omega_j) \cdot P(\omega_j)}{P(\mathbf{x}_i)} \tag{2.6}$$

Assuming a d-dimensional feature vector, the class conditional probability is obtained as:

$$P(\mathbf{x} \mid \omega_j) = P(x_1 \mid \omega_j) \cdot P(x_2 \mid \omega_j) \cdot \ldots \cdot P(x_d \mid \omega_j) = \prod_{k=1}^{d} P(x_k \mid \omega_j) \tag{2.7}$$

The evidence P(x) is the probability of observing a feature pattern x independent of the class label which is calculated as:

$$P(\mathbf{x}_i) = P(\mathbf{x}_i \mid \omega_j) \cdot P(\omega_j) + P(\mathbf{x}_i \mid \omega_j^C) \cdot P(\omega_j^C) \tag{2.8}$$

Suppose we have a feature sample x_i, NB classifiers works as: classify sample x_i as ω_1 if $P(\omega_1 \mid \mathbf{x}_i) > P(\omega_2 \mid \mathbf{x}_i)$ else classify the sample as ω_2.

2.2.4 Support Vector Machine

Support Vector Machine is a classifier which performs classification by finding the hyperplane which maximizes the margin between the classes [51]. A line splitting the input samples into two classes is a hyperplane. In practice, SVM is implemented using a kernel which is the basis of learning hyperplane. SVM is typically used when samples of two classes are closer and harder to classify. The support vector machine searches for the closest points, which it calls the "support vectors". SVM connects the support vectors to draw the best separating line called a hyperplane that bisects and is perpendicular to the connecting line. Since we already have the hyperplane created, all the incoming input samples are separated by this hyperplane [52]. SVM works best for binary type classification problem.

Let $\mathbf{x}$ be a n-dimensional feature vector. y be the class (i.e., the output of the SVM); $y \in \{-1, 1\}$. P and Q be the parameters of the SVM which are learnt during the training phase. The class of the i^{th} sample $((\mathbf{x}^{(i)}, y^{(i)}))$ is determined as:

$$y^{(i)} = \begin{cases} -1 & \text{if } \mathbf{P^T x}^{(i)} + Q \leq -1 \\ 1 & \text{if } \mathbf{P^T x}^{(i)} + Q \geq 1 \end{cases} \tag{2.9}$$

SVM always maximizes the distance between two decision boundaries by taking $\min_{\mathbf{p}} \frac{\|\mathbf{p}\|}{2}$. SVM should always classify all the input samples which means:

$$y^{(i)}(\mathbf{P^T x}^{(i)} + b) \geq 1, \forall i \in \{1, \ldots, N\} \tag{2.10}$$

Equation 2.10 leads to the optimization problem. Optimization process involves the calculation of inner products. The function that maps the input samples to the inner product is called a Kernel. There are many kernel implementation in the literature to choose from, most commonly used ones are RBF Kernel, Linear Kernel, and Gaussian Kernel. Typically, [53]using a linear kernel when the number of features is larger than the number of observations and using a Gaussian kernel when the number of observations is larger than the number of features maximizes a classification accuracy and computational efficiency.

2.2.5 Decision Tree

DT are a non-parametric supervised learning algorithm for classification. DT works by learning decision rules inferred from the data features and creating a model to classify the future input data. DTs have been used in many research because of its easiness to understand and visualization property. With DTs, it is possible to validate a model using

statistical tests which increases the reliability of the model and also easy to interpret. One disadvantage of using DT is overfitting; learners can create an over-complex tree that does not represent the general notion of data precisely. Since DT's works on decision rule in every step, one small variation in data might result in generation of completely different tree. The cost of predicting is logarithmic in the number of data points used to train the tree. Total cost of entire tree is:

$$O(n_{features} n^2_{samples} log(n_{samples}))$$

There are various DT algorithms [54, 55, 56] such as: ID3, CART, C4.5 and C5.0. ID3 creates a multiway tree finding, for every node, the categorical features that will yield the largest information gain for a ceratin category. ID3 and C4.5 are almost similar except C4.5 removes the restriction that features must be categorical. C4.5 converts the trained tree into rules and determines the accuracy of rules to maximize the prediction. CART in the other hand supports numerical target variables and constructs binary trees using the feature that yields the largest information gain at each node.

Given a training vector $w_i \in R^n, i = 1, ..., l$ and a label vector $z \in R^l$, DT recursively creates a tree by grouping the samples with the same label together. Each node of the tree is associated with a particular set of records R that are categorized by a specific rule on a feature. Let's say a attribute P can be tested as $P \leq x$, the set of records R can be partitioned into two trees as: $R_{ltree} = \{r \in R : r(P) \leq x\}$ and $R_{rtree} = \{r \in R : r(P) > x\}$ Similarly, all the attributes of the records are partitioned recursively. The accuracy of the best split is measured out by measuring *impuritymeasures*. The impurity of the parent node has to decrease as we traverse down the tree. Let $(E_1, E_2, \ldots, E_n)$ be a split induced and impurity measure I.

$$\Delta = I(E) - \sum_{i=1}^{k} \frac{|E_i|}{|E|} I(E_i) \tag{2.11}$$

$$p_j = \frac{|\{t \in E : t[C] = c_j\}|}{|E|} \tag{2.12}$$

$$H(E) = 1 - \sum_{j=1}^{Q} p_j^2 \tag{2.13}$$

Where,P_j is a fraction of records in E of class c_j, and Q is the total number of classes.

2.2.6 Random Forest

Random Forest is one of the popular methods in machine learning. It is an ensemble type of machine learning algorithm called bootstrap aggregation or bagging. An ensemble method combines the predictions from different machine learning algorithms to make a more powerful and accurate classifier than any individual learning algorithms. Bootstrap method is used for estimating statistical quantities from the test samples. The Bootstrap Aggregation algorithm creates multiple different models from a single training data-set while random forest algorithm makes a small tweak to bagging and results in a highly accurate classifier.

Bootstrap method [57] is a statistical method for estimating a quantity from a data sample such as mean, variation, standard deviation, etc. Bootstrap method creates a random sub-samples of our dataset with replacement and calculates a statistical quantity of each subsample. Finally, it calculates the statistical quantity using the result of each sub-sample. For example, lets say we used three resamples and got the standard deviation values 2.3, 4.5 and 3.3. After calculating the average of these numbers, we could take the estimated sd of the data to be 1.10. These quantities are estimated as learned coefficients. Bootstrap algorithm is used to reduce the variation for the algorithms that have high variance. Bagging is used in DT algorithm to reduce the high variation and inturns creating a Random forest classifier.

The result of prediction from random forest [58] is uncorrelated or weakly correlated. In CART, when selecting a split point, the learning algorithm looks through all variable values in order to select the most optimal split-point that minimizes the error. The random forest algorithm changes this procedure so that the learning algorithm is limited to a random sample of features searches. The number of variables that can be searched in each split (p) is input to the classifier [59].

(2.14)

For the classification good split point p is:

$$\sqrt{q} \tag{2.15}$$

For regression good split point value p is:

$$\frac{q}{3} \tag{2.16}$$

Where p is the number of randomly selected features that are available to search at a split point and q is the number of input varibles

(2.17)

2.2.7 Neural Network

Neural Networks are the computing systems inspired by the biological neural networks; they learn to do task progressively looking at the previous data without being programmed for the specific task. NN is based on a collection of units called artificial neurons; each neuron can transmit the information to the following neurons. NN consists of layers of neurons, each layer performing different kinds of operation/calculation from first(input) layer to output layer. The main idea behind the NN operation is similar to the biological brain which has the capacity of backpropagation, information adjustment, and bidirectional information transfer. A neuron in NN receives input, changes its internal state according to the input, and produce output relying on the input data and internal state/activation. There are four essential components of neural network [60]:

- Neuron: A neuron which receives an input and consists of activation function, threshold, and output function/identity function.
- Connections and Weights: Each neuron has a connection, and connection has an assigned weight w_{pq}. Where p is the predecessor and q is the successor neuron.
- Propagation function: Propagation function computes the input to the successor neuron from the data of predecessor neuron.

$$F_j(t) = \sum_i Output_i(t) w_{pq} \tag{2.18}$$

- Learning rule: This is the learning algorithms to produce the desired output, which typically modifies the parameter of neural networks such as weights and thresholds.

Learning algorithms have gained in more attraction these days to find the best optimal solution possible. This requires us to define a cost function $C : F \to \mathbb{R}$ such that no solution has the cost less than the optimal solution, i.e., $C(S^*) \leq C(f) \forall f \in F$. There are many variants of a neural network; typically the variants are obtained by using different learning algorithms/rules. In this thesis, we use Multi-layer Perceptron (MLP) (MLP)[1]

for analyzing the power traces. MLP is a supervised learning algorithm that learns a function $f(\cdot) : P^n \rightarrow P^m$ by training on a dataset, where n and o are the numbers of the dimension for input and output respectively. Figure 2.4 shows an example of one hidden layer MLP.

2.2.8 Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) is a classic classifier with quardatic decision surface. Linear discriminant analysis is supervised dimensionality reduction technique which works by projecting the data in the direction of maximum separation between the classes. Both LDA and QDA are derived from a probabilistic models which model the class conditional distribution of the data $P(X|y = k)$ for each class p. Predictions uses Bayes rule:

$$P(y = p|X) = \frac{P(X|y = p)P(y = p)}{P(X)} = \frac{P(X|y = p)P(y = p)}{\sum_l P(X|y = l) \cdot P(y = l)} \tag{2.19}$$

We select the class p which maximizes this conditional probability. For QDA, P(X—y) is modelled as a multivariate Gaussian distribution with density:

$$p(X|y = k) = \frac{1}{(2\pi)^n |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(X - \mu_k)^t \Sigma_k^{-1}(X - \mu_k)\right) \tag{2.20}$$

For this classifier, we need to estimate the class priors P(y=k), the class means μ_k, and the covariance matrices.

2.2.9 k-Nearest Neighbors and Dynamic Time Warping (DTW) Algorithms

DTW is an algorithm to find the optimal match between two time series which may vary in speed. Time series are wrapped non-linearly in time dimension independent of the non-linear variation along the series which may need compression or expansion in time in order to find the best mapping. DTW algorithm is first introduced by [18], and being used in time series data mining and sequence classification [19, 20, 21, 22, 23]. [24] has shown the efficient online sequence learning using unsupervised convolutional the neural network, but lately, DTW has gained in popularity in time series classification with the introduction of 1-Nearest Neighbor and faster computation with tighter lower bound [25, 26, 27]. [28, 29,

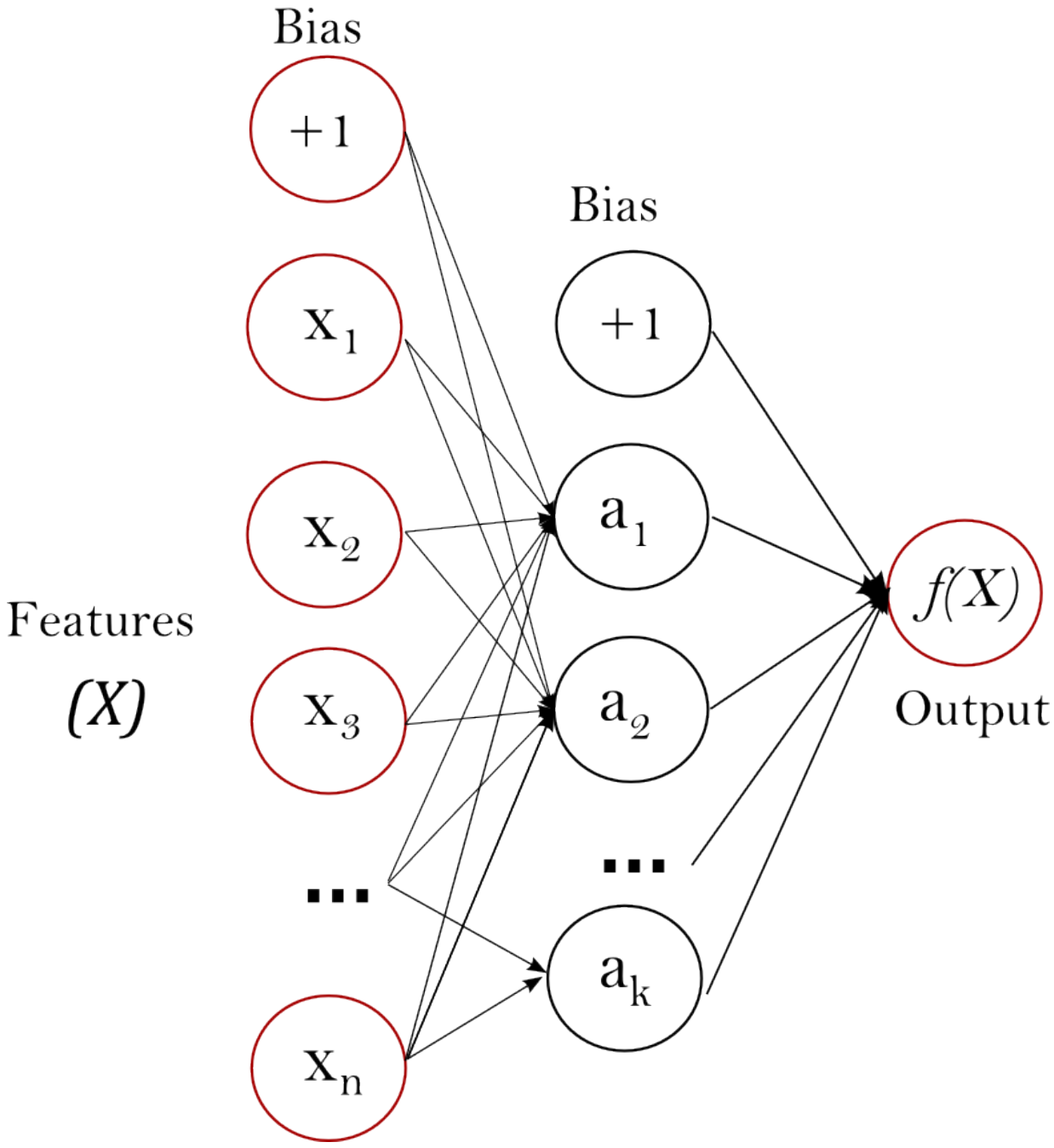


Figure 2.4: One hidden layer MLP [1]

30, 31] have shown that DTW combined with 1-Nearest Neighbor achieves near-optimal performance in time series classification.

The DTW algorithm compares two time series A and B where $A = (a_1, a_2, a_3, \ \cdots \ , a_m)$ and $B = (b_1, b_2, b_3, \ \cdots \ , b_n)$ of length m and n, respectively. To compare the similarity between two time series A and B, we need local distance matrix between each element of the series. The distance between (a, b) is small if they similar to each other. To measure the local cost between each element in time series let us define an optimal value function $D(i, j)$ as the distance between $A(1 : i)$ and $B(1 : j)$ that has the mapping path from $(1, 1)$ to (i, j). The optimal minimum path can be calculated using the dynamic programming approach as:

$$D(i,j) = |A(i) - B(j)| + min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases} \tag{2.21}$$

where $D(1, 1) = |A(1) - B(1)|$

Finally, the calculated D(m,n) is the optimal path. The warping path obtained from this calculation is subjected to several constraints [30] such as the warping path must start and end at diagonally opposite corner of the matrix, steps in the matrix are restricted to adjacent cells and points should be monotonically spaced in time.

k-NN is a non-parametric [61] method of classification which outputs the class by a majority vote of its neighbors, with the object being assigned to the class with most votes among its k-nearest neighbors. Typical K-NN uses Euclidean distance as a measure of distance while DTW uses the dynamic time warping distance like score as a measure of distance. 1-nearest neighbor merely outputs the closest matching class. 1-nearest neighbor with DTW calculates the distance along the warping path and outputs the closest match.

Chapter 3

System Model

In this chapter, we explain our system setup which is used for power data acquisition. We address the source code instrumentation techniques in section 3.2 and data preprocessing steps in section 3.3.

3.1 Powertraces Capture Setup

As described in section 1, our proposed technique uses the concept of non-intrusive side channel power analysis to determine the code block being executed by the MCU. We capture the power consumption by adding a shunt resistor in series with the power-in line going to the MCU so that a voltage proportional to the instantaneous power consumption is produced. The shunt resistor is chosen such that the voltage drop that it causes is small enough that the normal functionality of the MCU is not affected. Since the produced voltage is in the order of a few millivolts, we added an analog input stage to amplify it for further processing. The signal is then digitized by an analog to digital converter (ADC), which samples at 14-bit resolution at a frequency of 2 MHz. The converted samples are captured using a Saleae Logic Analyzer. Figure 3.1 shows the power capture setup of the experiment. Port_Bit_Marker in Figure 3.1 is used for the training phase of the classifier: it allows us to segment the power trace into power trace segments corresponding to BBs. To this end, the MCU port_bit (Port_Bit_Marker) toggles the logic level at the beginning of each BB.

In our experiments, we use a custom designed non-preemptive multitasking operating system for AVR microcontroller that supports basic multitasking with yield/context

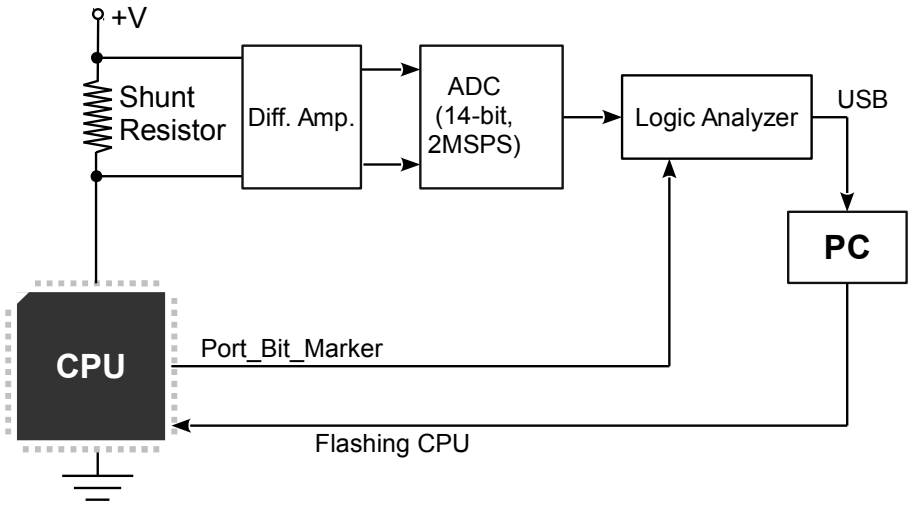


Figure 3.1: Capture setup

switching and Round-Robin scheduling. To show the efficacy of our approach on the industrial real-time system, we chose four application programs to run each as one task: Cruise Control, Water-level Control, Adaptive Differential Pulse-code Modulation (ADPCM) encoding, and Cyclic Redundancy Check (CRC) computation. Cruise Control and Water-level Control are obtained directly from the sample SCADE [62] models and run without any modification in the program.[1] ADPCM and CRC are obtained from MiBench [63].

To carry out the experimental evaluation of our technique, the source code is instrumented to allow us to obtain the power consumption information of each BB (section 3.2 discusses this in more detail), which is therefore used as the training data for our classifiers. Instrumentation done in this phase does not have any effect on the functionality and the execution flow of the program. Moreover, the code instrumentation does not introduce undesirable overhead in the captured power traces. During classification, we use a non-instrumented version of a program where the power trace of a complete program is fed as input to the resulting trained system which gives the execution trace.

The power trace associated with each BB may exhibit variations due to the context in which it is executed (the input data and state). Thus, we should train the system with power traces of a set of execution instances that is statistically representative of the power

[1] Except for the instrumentation necessary for the experiments.

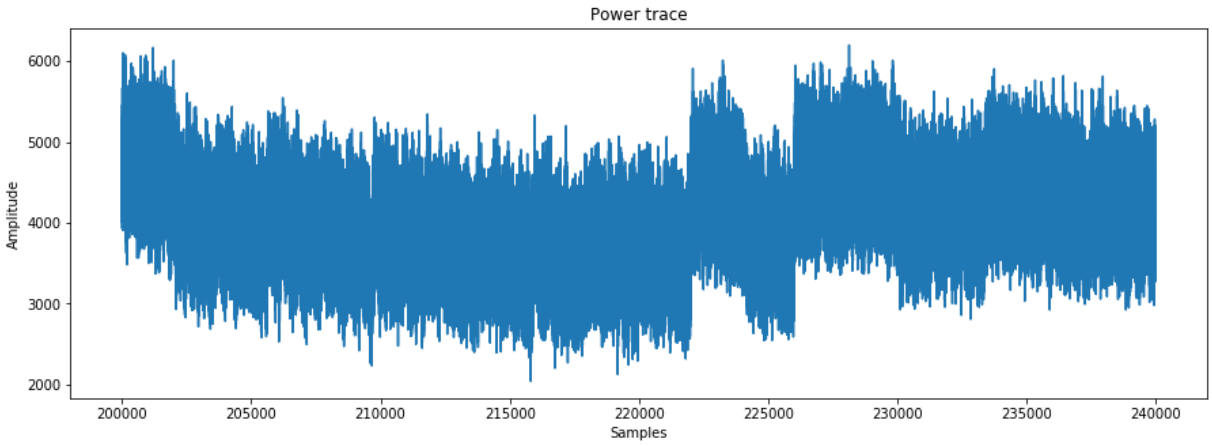
traces generation process. To this end, we used randomization for all program inputs and replicate the experiment multiple times. At the end of the training phase, we make sure that we have the power-trace data for all the BB of a complete system. Plot(a) in figure 3.2 shows the power trace and plot(b) shows the marker used to fragment the traces of each BB.

3.2 Source Code Instrumentation

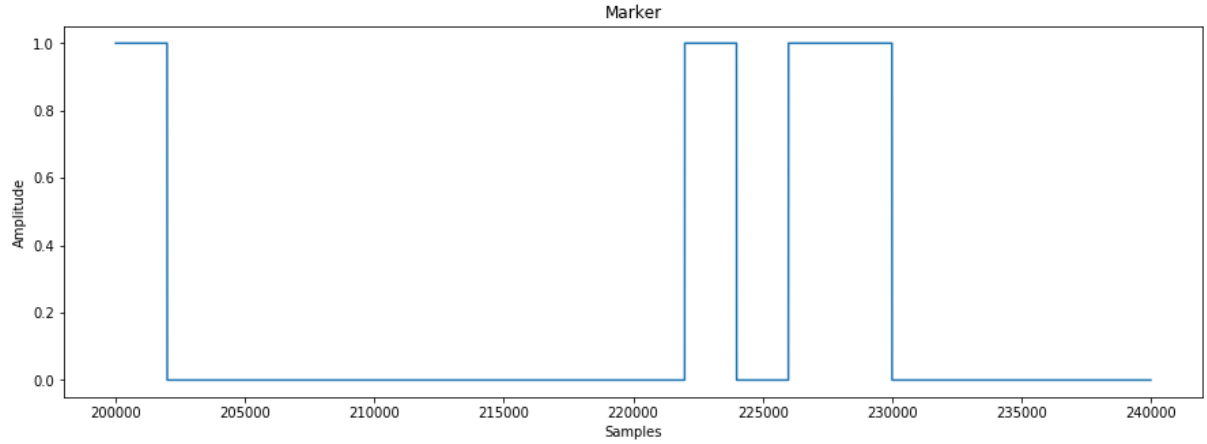
We used the LLVM optimizer and analyzer [64], -dot-cfg pass, to extract the CFG of the programs. Sometimes the BB produced by LLVM are too short(1-2 lines), having very few samples in power trace fragment, for the classifier to work. To address this issue, we merge short CFG nodes to create a large CFG node(single entry, single exit node) without deviating from the original control flow of the program. Appendix A.3 shows the part code to automate the instrumentation process.

We created two instrumented versions; one that executes on the target MCU, and another version that runs offline on a workstation. *Printf instrumented version* is the instrumented code that runs on the workstation, and *flip_port instrumented version* is the instrumented code that runs on the target. In *Printf instrumented version*, we instrument the source code by adding a print statement at the beginning of each BB which prints the ID of the BB that is currently executing and the yield information at the end of BB. In *flip_port instrumented version*, we instrument the source code by adding the FLIP_PORT_BIT statement in the exact same location as we did for the printf instrumented version. This version executes on the target and uses a general purpose input/output (GPIO) pin to signal transitions between BBs including context switch by toggling the port bit. The instrumentation simply places a pin-toggle statement at the beginning of each BB. This information is captured using the 16^{th} bit of logic analyzer as shown in Figure 3.1. Listings 3.1 and 3.2 show the instrumentation technique as we discussed above [3]. *Printf instrumented version* gives the actual trace which is processed to get the ground truth for our classifier. Execution path of program changes as input changes. For both versions of the program to follow the same execution path, we need same input for both of them. To achieve this, randomized.h is generated using a pseudo-random number generator which produces the random values to initialize the inputs. We use the resulting randomized.h in both versions such that the printf version and flip port version follow the same execution path every time.

[3]Complete instrumentation is shown in appendix A



(a)



(b)

Figure 3.2: Powertrace with marker

```
CruiseControllerfunction()
{
  /*variables for cruise control */
  printf ("CruiseControlNode0x13b5fe0\n");
  DetectIfPedalIsPressed();
  printf("yield\n");
}
```

Listing 3.1: Printf instrumented version

```
CruiseControllerfunction()
{
  /*variables for cruise control */
  FLIP_PORT_BIT;
  DetectIfPedalIsPressed();
  FLIP_PORT_BIT;
  yield();
}
```

Listing 3.2: Flip_port instrumented version

Since we run the printf instrumented versions of each program (task) individually, we need to combine the actual traces of both tasks to get an overall actual trace as it happens during the execution on the MCU. As an example to illustrate this process, suppose that the trace (obtained from the printf instrumented version) of cruise control is:

$$\text{A} \rightarrow \text{B} \rightarrow \text{yield} \rightarrow \text{C} \rightarrow \text{yield} \ ,$$

and the trace of water level control is:

$$\text{D} \rightarrow \text{yield} \rightarrow \text{E} \rightarrow \text{F} \rightarrow \text{yield}.$$

Given the Round-Robin scheduler, the overall (global) trace of the complete system is:

$$\underbrace{A \rightarrow B}_{\text{T1}} \rightarrow \underbrace{yield}_{\text{C.S}} \rightarrow \underbrace{D}_{\text{T2}} \rightarrow \underbrace{yield}_{\text{C.S}} \rightarrow \underbrace{C}_{\text{T1}} \rightarrow \underbrace{yield}_{\text{C.S}} \rightarrow \underbrace{E \rightarrow F}_{\text{T2}} \rightarrow \underbrace{yield.}_{\text{C.S}}$$

For the systems having more than two tasks, the overall trace can be obtained similarly by considering the round-robin scheduling and cyclically moving through the tasks. Alternatively, we can get the actual trace by simulating the OS (with same tasks in flip port version) with an IDE such as AVRSTUDIO. We also verified the correctness of actual trace by simulation. Figure 3.4 shows the CFG of the overall system running on the device where subgraph on the left side represents the CFG of the cruise control program and subgraph on the right represents the CFG of the water-level controller. Figure 3.3 shows the power traces of few basic block in the multitasking system.

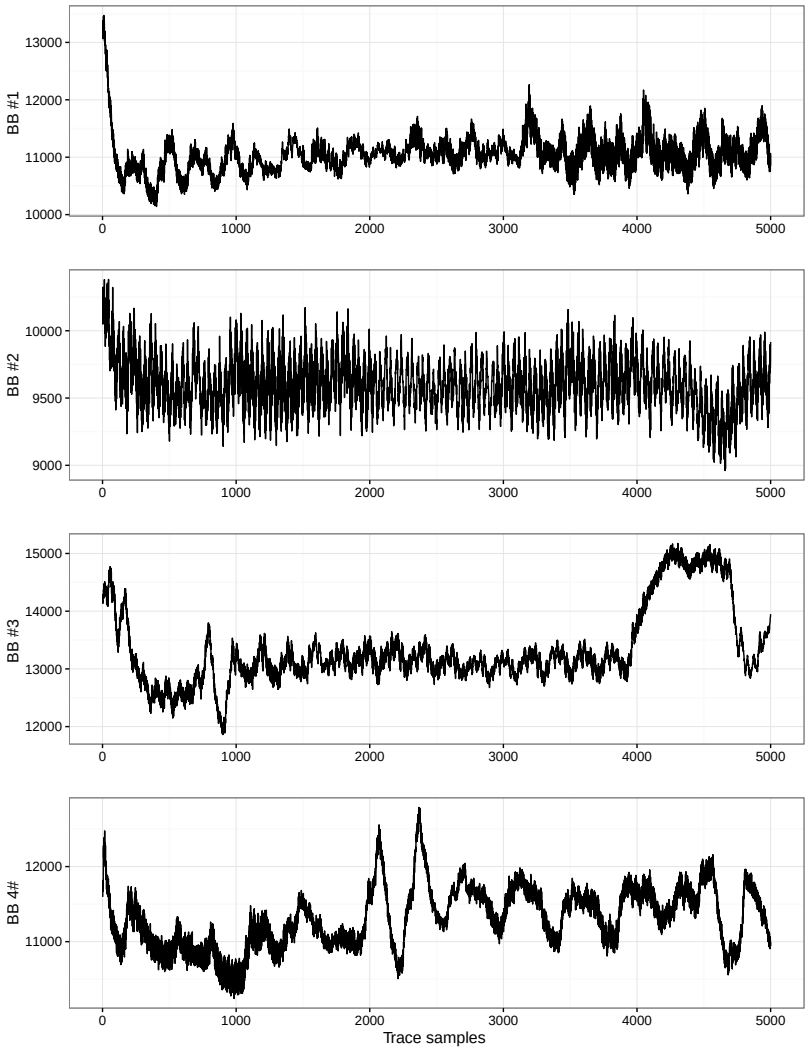


Figure 3.3: Power-traces of the BBs

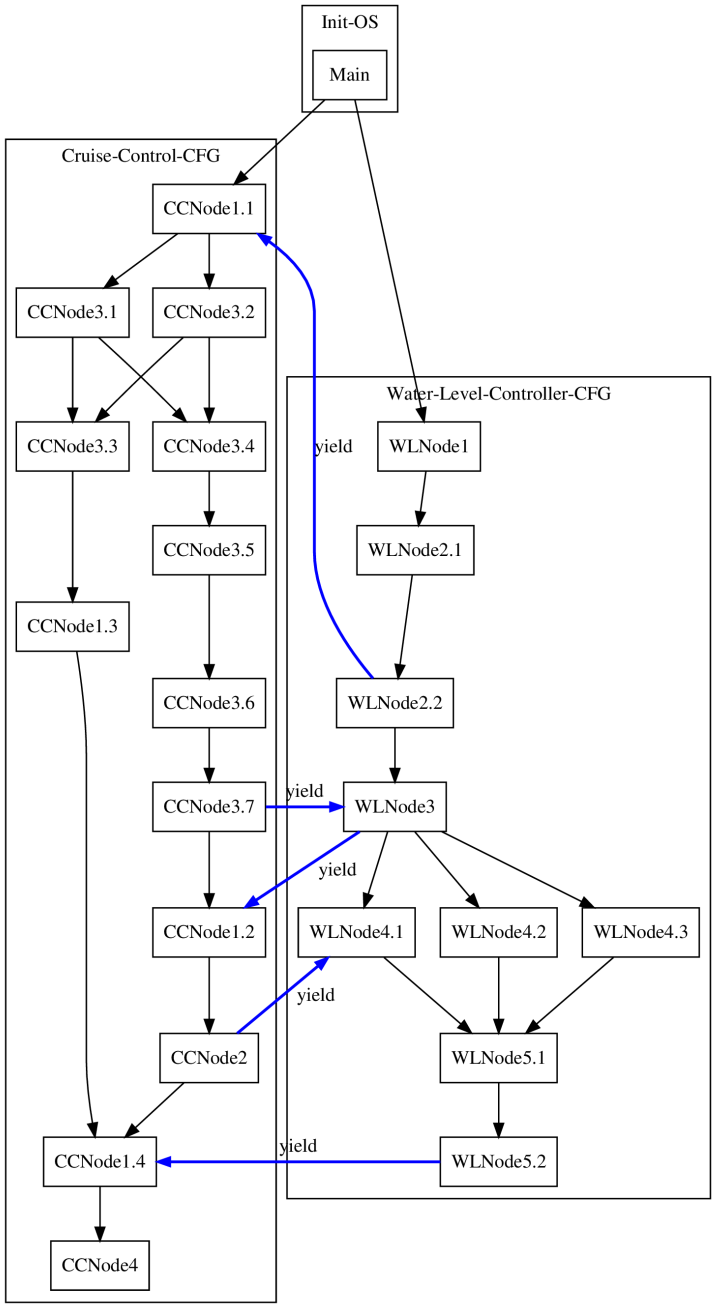


Figure 3.4: CFG of the system with two tasks (one instance)

3.3 Capturing Power Traces and Preprocessing

We captured the power trace using the logic analyzer whose 16^{th} bit was connected to a *port_bit* of MCU and bits 1-15 delivered the power data. We generate traces for each BB from the entire trace using FLIP_PORT_BIT (each logic level change signifies either the transition of a BB or a context switching). We replicate the trace capture of the system approximately 1000 times to get 1000 traces of each basic block. At this point, we have traces of each BB with the label(ground truth). Since powertrace is a time series data, any suitable machine learning algorithm can be used to classify. We will focus on feature-based and shape-based method of classification in chapter 4.

Chapter 4

Classification Techniques in Power Trace

In this chapter, we discuss the classification techniques in power trace data which is captured as a time series data. As described in section 2.1.5, time-series classification can be broadly divided into four categories –shape-based, feature-based, model-based, and compression-based. In this chapter, we focus on the feature-based and shape-based model. For feature-based classification, we extract various features from time series and train a classifier with those features. For the shape-based method, we calculate the dynamic time warping distance and use $k-$NN in DTW distances.

4.1 Experimental Setup

In this section, we provide more insight on tools used, benchmark programs and evaluation metrics.

4.1.1 Tools Used

We use ATMega2560 MCU clocked at 1 MHz with STK600 evaluation board in our experiment. Saleae Logic Analyzer [65] was used to capture traces into the laptop as explained in 3.1. Lenovo T450 laptop equipped with Intel®core^{TM} i5-5200U CPU @ 2.20GHz was

used to capture traces. We use a custom designed non-preemptive multitasking operating system for AVR microcontroller that supports basic multitasking with yield/context switching and Round-Robin scheduling.

4.1.2 Benchmark Programs

To show the efficacy of our approach on the industrial real-time system, we chose four application programs to run each as one task: Cruise Control, Water-level Control, ADPCM encoding, and CRC computation. Cruise Control and Water-level Control are obtained directly from the sample SCADE [62] models and run without any modification in the program.[1] ADPCM and CRC are obtained from MiBench [63]. Table 4.1 shows the number of BBs in each function. We tested all the classification algorithms with the same data.

Table 4.1: Benchmark programs and number of BB

Task Name	Function	#of BB
CruiseControl	CruiseStateManagement	7
CruiseControl	Controller-Node	4
CruiseControl	CruiseSpeedManagement	1
CruiseControl	ThrottleCmd	1
WaterLevel	DecideAlarm	2
WaterLevel	Controller-Node	2
WaterLevel	DecideOutFlow	3
WaterLevel	DecideInFlow	2
ADPCM	Encoder	4
ADPCM	Decoder	4
CRC	CRC_Computation	2
Total	–	32

[1] Except for the instrumentation necessary for the experiments.

4.1.3 Experiment

By replicating the experiment 1000 times over the same configuration/setup, settings, and the environmental condition, we determine how large error should be deemed to be statistically significant. Since we are initializing the input randomly, the CFG is different in each run due to which some BB executed less number of times. We run an additional experiment to make 1000 traces for each basic block. We train the system with 1000 traces of each BB and classify continously. In continuous classification, we ran the experiment on traces captured for thirty minutes which is roughly 900 execution of each basic block. For the experimental purpose, we calculated the time it took to run all four programs of multitasking system in order. On an average, it took 0.6 sec to execute all four program(one cycle) in order. We classified continuously on the traces captured for 30 minutes which is approximately 1080 cycle –considering 25 BB for each cycle 27000 traces in total.

The length of CFG, number of yields, and number of tasks are the factors considered in the experiment. To show the efficacy of our technique in the real-time systems, we tested our classifier with a multiple number of yields(8, 12, 15, and 17) and a multiple number of tasks(2, 3, and 4).

4.1.4 Evaluation Metrics

We use the positive predictive value (precision in Equation (4.1)) to evaluate the performance of the classifier. Since our classifier always outputs one of the options all the time, whether it is true positive or false positive; there is no notion of recall, specificity, and fall-out in our experiment. Positive predictive value (PPV/P) is defined as the ratio of total True positive (TP) instances to the sum of True Positive instances and False Positive (FP) instances. In our case, TP are instances where the classifier correctly determines the basic block currently executing in the MCU while FP are instances of incorrect classifications.

$$P = \frac{TP}{TP + FP} \tag{4.1}$$

where P denotes the precision, TP is the total number of instances with correct prediction (true positives), FP is the total number of instances with an incorrect prediction (misclassifications or false positives).

Measuring the Precision: To measure the precision, we take the actual execution trace running in offline mode, i.e., printf instrumented version, as the ground truth. We then compare the output of the classifier with the actual trace to determine the true

positives (correct classifications) and false positives (misclassifications). Let us consider the instance where classified sequence is: $BB_A \rightarrow BB_B \rightarrow yield \rightarrow BB_C \rightarrow BB_E$ and the actual sequence is $BB_A \rightarrow BB_B \rightarrow yield \rightarrow BB_C \rightarrow BB_D$. So, $FP = 1$, $TP = 4$ and precision $P = 80\%$ as calculated from equation 4.1. Average precision is calculated with several runs using arithmetic mean. Similarly, to calculate the precision of individual BB, we use TP and FP from several runs. Let us consider one BB CruisControlNodeA_1 which is accurately classified in 9 runs out of 10. So, $FP = 1$, $TP = 9$ and the precision P of CruisControlNodeA_1 is 90%. We calculate precision of all the BB similarly.

4.2 Feature-based Classification

In this section, we address the classification algorithm comparison by extracting features from the power-trace time-series data. A feature is an individual measurable property which statistically defines a time series. We extract following features from the time series and train the classification algorithm:

- amplitude
- maximum
- max_slope
- median
- median_absolute_deviation
- percent_close_to_median
- minimum
- skew
- weighted_average
- std

All the classification algorithms are run with the features extracted. We will discuss the time and computational requirement of most widely used classification techniques such as k-NN, SVM, Gaussian Method, DT, Random Forest, Neural Network, and Quadratic Discriminant Analysis. To this end, we compare the accuracy vs. time and computational requirement trade off of all the classification methods mentioned above. Figure 4.1 shows the plot of extracted feature for one BB.

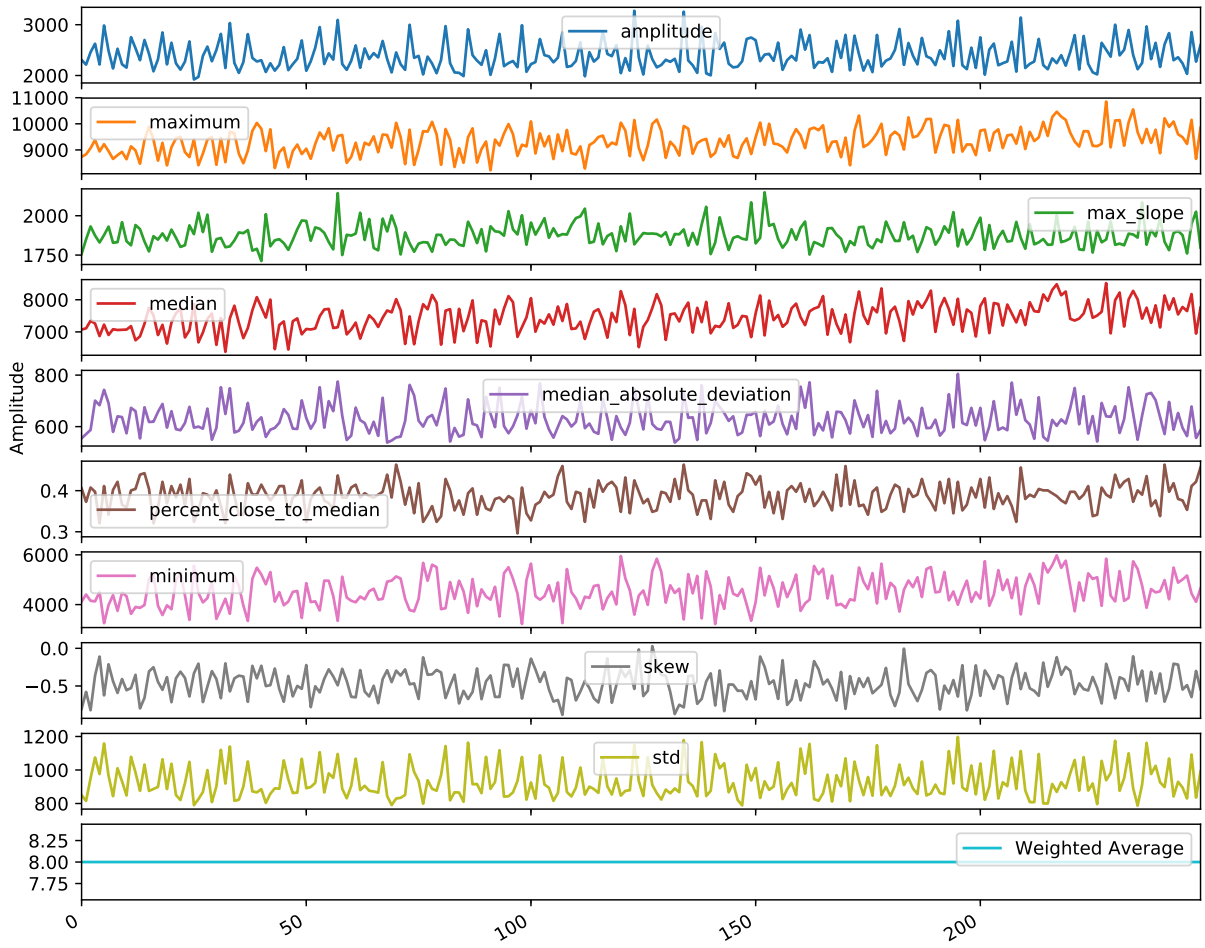


Figure 4.1: Power trace features plot

4.2.1 Results –Feature-based Classification

In this section, we discuss the results of feature-based classification algorithms. We ran the experiment with a different number of task and a different number of yield as described in section 4.1.3. In feature-based classification, we found that varying the number of task and number yield does not affect the precision. Table 4.5 shows the precision comparison of different methods. Testing time is calculated per 100 traces classification.

k-Nearest Neighbors

We use Euclidean distance (equation 2.4) as the measure of distance to classify the power trace.

- Precision of the model: 78%
- Testing time: 0.10 Sec

Support Vector Machine

We evaluated this classification technique using two kernels: a) Linear Kernel b) RBF Kernel. SVM linear kernel use the equation 2.9 to perform the classification. Results of Linear Kernel:

- Precision of the model: 76%
- Testing time: 10.27 Sec

Results of RBF Kernel:

- Precision of the model: 37%
- Testing time: 1.08 Sec

Gaussian

The evidence P(x) is the probablity of observing a feature pattern x independent of the class label which is calculated using Equation 2.5 and Equation 2.8.

- Precision of the model: 73%
- Testing time: 0.09 Sec

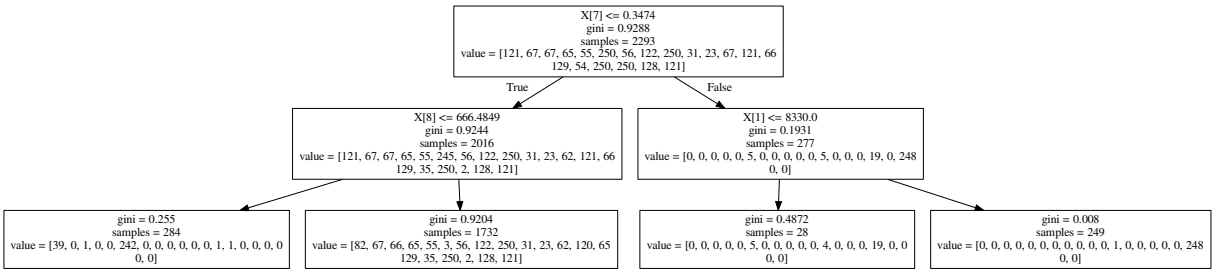


Figure 4.2: DT generated by the model with max_depth=2

Decision Tree and Random Forest

We use an optimised version of the CART algorithm from scikit learn. Impurities (Gini Index) and entropy are calculated using Equation 2.11, 2.12, and 2.13. We observed that the model with max_depth= 95 could produce maximum classification accuracy. Figure 4.2 shows the DT generated by the model with maximum_depth=2. We use max_depth=10 and n_estimators=10 for Random Forest.

- Precision of the model: 81%
- Testing time: 0.19 Sec

Classification result of Random Forest Classifier:

- Precision of the model: 89%
- Testing time: 0.20 Sec

Neural Network

We use MLP algorithm that trains the model using backpropagation as discussed in 2.2.7.
We use MLPClassifier(hidden_layer_sizes=(15,), random_state=1, max_iter=10, warm_start=True).

- Precision of the model: 38%
- Testing time: 0.77 Sec

Quadratic Discriminant Analysis (QDA)

Quadratic decision boundary is generated by fitting class conditional densities to the data using Bayes rule. To achieve this, the class conditional probability is calculated using Equation 2.19.

- Precision of the model: 72%
- Testing time: 0.10 Sec

4.2.2 Discussion –Feature-based Classification

In this section, we analyzed different classification algorithms in power trace data. The results of this experiment indicate that common classification techniques we tested are not so useful for our data. Since the power-trace data is different to usual time-series data in many sense, it may be difficult to incorporate those features in presented classification algorithms. Usually, time series data are fixed length and perfectly synchronized in time which is not the case in power-traces. Power-traces of different BB has different execution time and hence different length of traces. This poses a problem in the classification algorithm which works on similarities of features extracted.

Although, we got a reasonable accuracy (88%) with Random Forest classifier the training time of this classifier is very high (10 times more than K-NN). However, with a large sample size, caution must be applied, as the findings might not be able to generate the similar accuracy. Training time of K-NN classifier is almost zero but testing time is quite high in comparison to other tested classification algorithms. Contrary to expectations, neural network in our data resulted in very low accuracy (38%). It is difficult to explain this result, but it might be related to the temporal information of the time-series data at any time interval. These results provide further support for the need of shape-based classification techniques such as DTW. In the next section, we will discuss the use of DTW classifier in the context of execution reconstruction of the cooperative operating system.

4.3 Shape-based Classification

In this section, we use DTW as the distance metric used by the nearest neighbors (NN) classification technique. The time complexity of each DTW comparison is O($n \cdot m$) where n and m are the lengths of each time series. Due to this, the time taken to calculate the

Table 4.2: Overall precision of the system

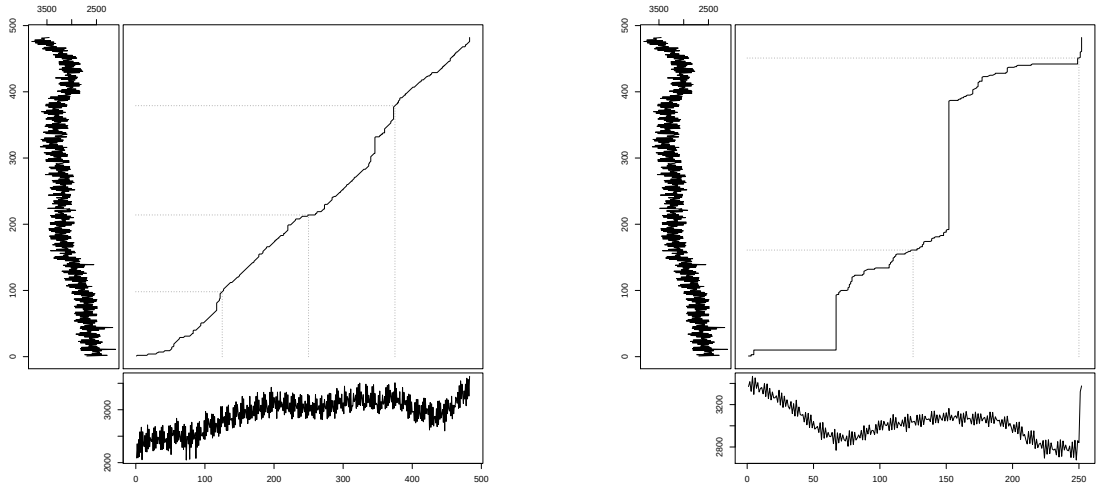
Application (Task)	**Precision (%)**
Water-level Controller	97.28
Cruise Controller	95.64
ADPCM	97.33
CRC	98.00
Yield	98.26
Average	97.30

warping distance of testing time series with *all* the training time series is considerably large. To reduce the number of comparisons, we average the power traces and generate one power trace for each BB. While this in principle means that we use the nearest centroid rule and lose the benefits of the 1-NN with DTW technique, it has been shown that the performance is similar for these two techniques [66]. Though they use warping as part of the averaging process, this is necessary for the general case, where the average of different time series instances may produce a result completely dissimilar to all the averaged samples. This is not the case in our system, as different instances of execution of a BB still execute the same operations, and it is only the data that varies, which has an effect with the characteristics of added noise. Figure 4.3 shows the warping path of the matching and the unmatching sequences. Plot (a) in Figure 4.3 is the warping path of the matching time series, and plot (b) is the warping path of dissimilar time series. Two similar time series have a path close to the diagonal of the matrix while dissimilar time series have paths that deviate from the diagonal.

Since the classification process is continuos, we need to determine the sizes of the traces as we continue to classify. We extract all possible sizes from the start point of the trace according to lengths of basic blocks in the master database and compare them against each trace of the master database to output the optimally aligned trace. We determine the next segmentation point from the CFG and repeat the process.

4.3.1 Results –Shape-based Classification

Table 4.2 shows the precision of a four-task multitasking system. From this table, it is evident that our technique can accurately produce an execution trace of a multitasking



(a) DTW path for similar traces

(b) DTW path for dissimilar traces

Figure 4.3: Warping path of similar and dissimilar traces

Table 4.3: Precision with different number of tasks

	Number of Tasks		
	2	**3**	**4**
Precision (%)	96.50	96.83	97.30

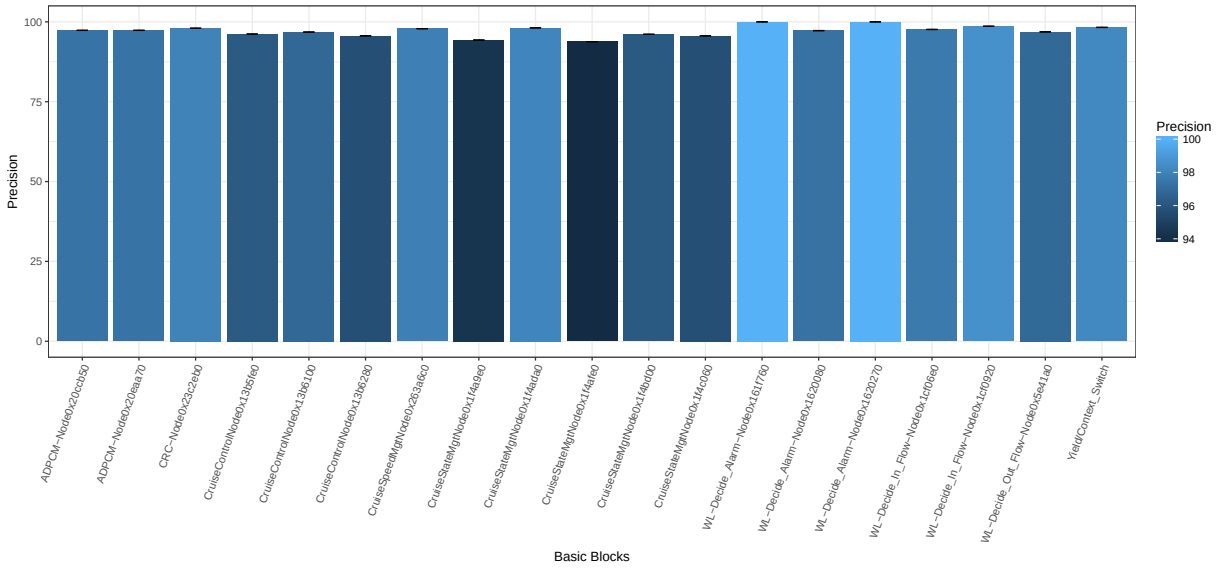


Figure 4.4: Precision of some BB

Table 4.4: Precision with different number of yields

	Number of Yields			
	8	**12**	**15**	**17**
Precision (%)	97.3	97.24	97.14	97.23

system that is running on the embedded device. We also ran our experiment with two, three, and four tasks, and with multiple numbers of yields. Table 4.3 presents the precision of an overall system with a multiple number of tasks and Table 4.4 presents the precision of an overall system with a multiple number of yields. From Table 4.4, it can be seen that the number of context switches in the application does not affect the precision; however, the precision changes by ±1% when varying the number of tasks (Table 4.3). The major cause of misclassification is due to the similar type of instruction being executed in the BB. Interestingly, the total of 230 context switches in a multitasking system has 226 accurate classifications. A probable explanation for the highly accurate classification of BB 'yield' is that its power consumption profile is unique with respect to all other BBs. Figure 4.4 shows the precision of individual BBs. We can see from Figure 4.4 that the precision of some BBs is 100% while an average precision of an overall system is 97.3%.

4.4 Shape-based Method(DTW) vs. Feature-based Method

Table 4.5: Classifiers' Precision (Expressed in Percentage)

	Shape-based	**Feature-based Classification**					
Number of Task	DTW	k-NN	DT	RF	QDA	Gaussian	SVM
2	96	78	81	88	72	73	76
3	96	78	81	88	72	73	76
4	97	78	81	88	72	73	76

As seen in Table 4.5, the precision of the shape-based method, i.e., DTW, is consistently more in comparison to feature-based methods. Since DTW allows the elastic shifting along the time-axis to accommodate the shape changes in time, it is more robust for time-series data which varies in shape. In the feature-based methods, while extracting features from a time series, we lose temporal information which is accounted in DTW. A possibility of elastic shifting of extracted features along the time axis could improve the classification accuracy of feature-based methods.

Chapter 5

Discussion, Future Work and Conclusions

This chapter presents the discussion on all the work presented in this thesis followed by concluding remarks.

5.1 Discussion and Future Work

As mentioned in the literature review, run-time monitoring, intrusion detection and debugging of the embedded software are always the central issues in the product development. The primary question in this research was to analyze the suitable mechanism for non-intrusive run-time monitoring of embedded software. In this end, prior studies that have noted the usability of side channel analysis as a method of non-intrusive tracing, however, they do not consider highly sophisticated code and conditions with multitasking system. In this thesis, we work upon the side channel power analysis to solve the problem of run-time monitoring, intrusion detection and debugging of the cooperative multitasking system.

An initial objective of the research was to include the multitasking support in the non-intrusive program tracing technique and to identify the suitable classification algorithm for power-traces. Although the area of machine learning is growing at a fast pace; very little information was found in the literature on the question of a suitable classification algorithm for power-traces. Chapter 4 set out with the aim of assessing the machine learning classification algorithms. The results of this assessment indicate that no already implemented classification techniques can be applied to our data. Usually, time series data are fixed

length and perfectly synchronized in time which is not the case in power-traces. Power-traces of different BB has different execution time and hence different length of traces. This nature of data poses a problem in feature-based classification techniques. On the question of classification category of the time-series data, the result from section 4.2.1 gives a substantial proof that we need a shape-based classification for our data. The maximum classification accuracy we got from the experiment of eight widely used machine learning algorithms was 89% with random forest classifier. To obtain this result with random forest classifier, we needed to determine many dependent parameters of the algorithm, which in general are determined using trial and error method(could be a limitation for traces from different MCUs). It is difficult to explain this results, but this might be related to the correlated feature of the similar relevance of the data. There are, however, other possible explanations such as overfitting of the data, biased nature of random forest with the entries with more levels, etc. Since this algorithm was not converging to maximum accuracy for all the data we tested, we decided not to go further with this method. The accuracy we got from the K-NN classifier was reasonable impressive considering time and computation constraints.

During our work on power profile classification, we could identify at least one limitation: BBs with similar power profiles were the primary cause of misclassification. For example, consecutive additions block could be mistaken for one multiplication. This paves the way for future work where finer granularity in CFG can be exploited for more accurate classification. Although we realized the possibility of using faster DTW classifiers such as [26], instead of traditional DTW classifier, we intend to undertake it as future work. There is a scope of improvement in classification time. It is essential to bear in mind that, for the classification to work in continuous real-time traces we need to compare all the traces in the database to find the optimal match which takes more time and computational resources. We believe that further research and investigation in this problem would be beneficial in the use of this technique in future research. Further research on efficient time-series searching and pattern matching would be useful to reduce the resources required for classification. Our work can also be extended to work in a more general anomaly detection for real-time multitasking systems. With the prior knowledge of CFG and the power trace of all the nodes/BB on the program, any external interfering program is likely to produce a significantly different control flow path and power traces, hence assisting in anomaly detection.

5.2 Conclusion

In this thesis, we presented a novel technique for run-time execution reconstruction in multitasking systems; assessed the classification algorithms for power-trace time series data; and implemented an online classification algorithm using bandwidth reduction technique. The proposed technique can be extended to a broad range of microcontroller (MCU) platforms. For the purpose of experiment, we chose the AVR ATMega2560, which is widely used in cyber-physical systems these days. Our work can be extended to work in a more general anomaly detection of a real-time multitasking system. With the prior knowledge of CFG and the power trace of all the nodes/BB on the program, any external interfering program is likely to produce a significantly different control flow path and power traces, hence assisting in anomaly detection. To list out the major contributions of this thesis;

- We built upon the previous work on non-intrusive program tracing using side power channel analysis, by demonstrating preemptive multitasking classification. We deployed a 1-NN combined with DTW and static code analysis to classify the traces.
- We assessed eight widely used machine learning classification algorithm with power-trace data and showed the comparison of time and computational resources for all the algorithms.

We strongly believe that our work provides a good starting point for future research on non-intrusive run-time tracing of an operating system.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] C. Moreno and S. Fischmeister, "Non-intrusive Runtime Monitoring Through Power Consumption: A Signals and System Analysis Approach to Reconstruct the Trace," in *International Conference on Runtime Verification*, pp. 268–284, Springer, 2016.

[3] C. Moreno, S. Fischmeister, and M. A. Hasan, "Non-intrusive program tracing and debugging of deployed embedded systems through side-channel analysis," *ACM SIGPLAN Not.*, vol. 48, no. 5, p. 77, 2013.

[4] C. Moreno, S. Kauffman, and S. Fischmeister, "Efficient Program Tracing and Monitoring Through Power Consumption – With A Little Help From The Compiler," in *Design, Automation, and Test in Europe (DATE)*, 2016.

[5] Liu, Yannan and Wei, Lingxiao and Zhou, Zhe and Zhang, Kehuan and Xu, Wenyuan and Xu, Qiang, "On Code Execution Tracking via Power Side-Channel," in *ACM Conference on Computer and Communications Security*, pp. 1019–1031, ACM, 2016.

[6] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a Side Channel Based Disassembler," in *Transactions on Computational Science X*, pp. 78–99, Springer Berlin Heidelberg, 2010.

[7] M. Msgna, K. Markantonakis, and K. Mayes, "The B-side of Side Channel Leakage: Control Flow Security in Embedded Systems," in *International Conference on Security and Privacy in Communication Systems*, pp. 288–304, Springer, 2013.

[8] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu, and W. Xu, "WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices," in *USENIX Workshop on Health Information Technologies*, USENIX, 2013.

[9] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Secure Embedded Processing through Hardware-Assisted Run-Time Monitoring," no. 1, pp. 178–183, 2005.

[10] K. Deng, A. W. Moore, and M. C. Nechyba, "Learning to recognize time series: combining ARMA models with memory-based learning," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pp. 246–251, Jul 1997.

[11] B. Bakshi and G. Stephanopoulos, "Representation of process trendsiv. induction of real-time patterns from operating data for diagnosis and supervisory control," *Computers & Chemical Engineering*, vol. 18, no. 4, pp. 303–332, 1994.

[12] C. Antunes and A. L. Oliveira, "Temporal data mining: An overview," in *KDD Workshop on Temporal Data Mining*, pp. 1–13, 2001.

[13] S. Zhong and J. Ghosh, "Hmms and coupled hmms for multi-channel eeg classification," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2, pp. 1154–1159, IEEE, 2002.

[14] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Information processing and technology," ch. Feature-based Classification of Time-series Data, pp. 49–61, Commack, NY, USA: Nova Science Publishers, Inc., 2001.

[15] R. J. Povinelli, M. T. Johnson, A. C. Lindgren, and J. Ye, "Time series classification using gaussian mixture models of reconstructed phase spaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 779–783, June 2004.

[16] M. W. Kadous, "Learning comprehensible descriptions of multivariate time series.," in *ICML*, pp. 454–463, 1999.

[17] M. W. Kadous and C. Sammut, "Classification of multivariate time series and structured data using constructive induction," *Machine Learning*, vol. 58, pp. 179–216, Feb 2005.

[18] R. Bellman and R. Kalaba, "On Adaptive Control Processes," *IRE Transactions on Automatic Control*, vol. 4, no. 2, pp. 1–9, 1959.

[19] W. Euachongprasit and C. A. Ratanamahatana, *Efficient Multimedia Time Series Data Retrieval Under Uniform Scaling and Normalisation*, pp. 506–513. Springer Berlin Heidelberg, 2008.

[20] T. Kahveci, A. Singh, and A. Gurel, "Similarity searching for multi-attribute sequences," in *Proceedings 14th International Conference on Scientific and Statistical Database Management*, pp. 175–184, 2002.

[21] T. Kahveci and A. Singh, "Variable length queries for time series data," in *Proceedings 17th International Conference on Data Engineering*, pp. 273–282, 2001.

[22] J. Gu and X. Jin, *A Simple Approximation for Dynamic Time Warping Search in Large Time Series Database*, pp. 841–848. Springer Berlin Heidelberg, 2006.

[23] C. Ratanamahatana and E. J. Keogh, "Making Time-Series Classification More Accurate Using Learned Constraints," in *SDM*, 2004.

[24] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Comput.*, vol. 28, pp. 2474–2504, Nov. 2016.

[25] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast Time Series Classification Using Numerosity Reduction," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 1033–1040, ACM, 2006.

[26] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm," *Knowl. Inf. Syst.*, vol. 47, pp. 1–26, Apr. 2016.

[27] B. D. Fulcher, M. A. Little, and N. S. Jones, "Highly comparative time-series analysis: the empirical structure of time series and their methods," *Journal of The Royal Society Interface*, vol. 10, no. 83, 2013.

[28] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, 2008.

[29] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, pp. 40–48, 2010.

[30] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, (New York, NY, USA), pp. 262–270, ACM, 2012.

[31] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Min. Knowl. Discov.*, pp. 1–55, nov 2016.

[32] Aleph One, "Smashing the stack for fun and profit," *Phrack magazine*, 1996.

[33] Solar Designer, ""return-to-libc" Attack," *Bugtraq*, Aug 1997.

[34] Mouser Electronics, "Choosing the optimal low power mcu," 2016. http://ca.mouser.com/applications/low_power_choosing_mcu.

[35] Texas Instruments, "CMOS Power Consumption and Cpd Calculation," 1997.

[36] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology – CRYPTO' 99*, pp. 388–397, 1999.

[37] F. E. Allen, F. E., Allen, and F. E., "Control flow analysis," in *Proc. a Symp. Compil. Optim.* -, vol. 5, (New York, New York, USA), pp. 1–19, ACM Press, 1970.

[38] T. Popp, S. Mangard, and E. Oswald, "Power Analysis Attacks and Countermeasures," *IEEE Design Test of Computers*, vol. 24, no. 6, pp. 535–543, 2007.

[39] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross, "Control-flow bending: On the effectiveness of control-flow integrity.," in *USENIX Security*, vol. 14, pp. 28–38, 2015.

[40] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*, AusPDC '10, pp. 61–70, Australian Computer Society, Inc., 2010.

[41] G. Gracioli and S. Fischmeister, "Tracing Interrupts in Embedded Software," in *Proc. of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pp. 137–146, June 2009.

[42] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, ACM, 2003.

[43] K. P. Murphy, *Machine learning: a probabilistic perspective*. 2012.

[44] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[45] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, July 1959.

[46] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, pp. 78–87, Oct. 2012.

[47] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006.

[48] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, (Amsterdam, The Netherlands, The Netherlands), pp. 3–24, IOS Press, 2007.

[49] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s.," *Philosophical Transactions*, vol. 53, pp. 370–418, 1763.

[50] S. Raschka, "Naive bayes and text classification I - introduction and theory," *CoRR*, vol. abs/1410.5329, 2014.

[51] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, pp. 1155–1178, 2007.

[52] A. Shashua, "Introduction to machine learning: Class notes 67577," *CoRR*, vol. abs/0904.3664, 2009.

[53] N. Ayat, M. Cheriet, and C. Suen, "Automatic model selection for the optimization of svm kernels," *Pattern Recognition*, vol. 38, no. 10, pp. 1733 – 1745, 2005.

[54] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," 1999.

[55] S. L. Salzberg, "C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993," *Machine Learning*, vol. 16, pp. 235–240, Sep 1994.

[56] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[57] T. Hastie, *The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations*. New York: Springer, 2001.

[58] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 832–844, Aug 1998.

[59] J. Brownlee, "Bagging and random forest ensemble algorithms for machine learning," 2016-04-22.

[60] A. Zell, *Simulation neuronaler Netze*. Bonn Paris Reading, Mass. u.a: Addison-Wesley, 1994.

[61] D. J. Hand, P. Smyth, and H. Mannila, *Principles of Data Mining*. Cambridge, MA, USA: MIT Press, 2001.

[62] F.-X. Dormoy, "SCADE 6: A Model Based Solution for Safety Critical Software Development," in *European Congress on Embedded Real Time Software*, 2008.

[63] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Workload Charact. 2001. WWC-4. 2001 IEEE Int. Work.*, pp. 3–14, 2001.

[64] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, (Washington, DC, USA), pp. 75–, IEEE Computer Society, 2004.

[65] Saleae, "Python Library to Control a Saleae Logic Analyzer," 2016 –. [Online; accessed ¡today¿].

[66] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Faster and More Accurate Classification of Time Series by Exploiting a Novel Dynamic Time Warping Averaging Algorithm," *Knowl. Inf. Syst.*, vol. 47, no. 1, pp. 1–26, 2016.

[67] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," 2012.

[68] K. Lamichhane, C. Moreno, and S. Fischmeister, "Non-intrusive program tracing of non-preemptive multitasking systems using power consumption," in *Proc. of Design, Automation, and Test (DATE)*, (Dresden, Germany), 2018.

[69] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1019–1031, ACM, 2016.

[70] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," *CoRR*, vol. abs/1701.07681, 2017.

[71] V. Banciu, E. Oswald, and C. Whitnall, "Reliable Information Extraction for Single Trace Attacks,"

[72] N. Kavvadias, P. Neofotistos, S. Nikolaidis, C. Kosmatopoulos, and T. Laopoulos, "Measurements Analysis of the Software-Related Power Consumption in Microprocessors," *IEEE Trans. Instrum. Meas.*, vol. 53, pp. 1106–1112, aug 2004.

[73] T. Giorgino, "Computing and visualizing dynamic time warping alignments in r: The dtw package," *Journal of Statistical Software*, vol. 31, no. 1, pp. 1–24, 2009.

[74] A. Silberschatz, P. B. Galvin, and G. Gagne, *Applied Operating System Concepts*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[75] S. Sun, Z. Yan, and J. Zambreno, "Experiments in Attacking FPGA-Based Embedded Systems using Differential Power Analysis,"

[76] Institute of Electrical and Electronics Engineers., IEEE Circuits and Systems Society., IEEE Computer Society., and IEEE Solid-State Circuits Council., *IEEE transactions on very large scale integration (VLSI) systems.* Institute of Electrical and Electronics Engineers, 1993.

[77] T. Popp, S. Mangard, and E. Oswald, "Power Analysis Attacks and Countermeasures," *IEEE Des. Test Comput.*, vol. 24, pp. 535–543, nov 2007.

[78] B. Miller, F. Vahid, and T. Givargis, "RIOS: A Lightweight Task Scheduler for Embedded Systems,"

[79] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, pp. 561–580, Oct. 2007.

APPENDICES

Appendix A

Code Instrumentation

A.1 Instrumentation: Print Version

```
CruiseControllerfunction()
{

  bool variables_definition;

  /*more variables for cruise control */

  printf ("CruiseControlNode0x13b5fe0\n");

  /*Check the pedal position */

  DetectIfPedalIsPressed;
  variable_1 = FindSpeedLimit(SpeedInput);


  Manage_the_cruise_state_polling_control_variables
  Check_if_brake_is_pressed;
  Check_if_accelerator_is_pressed;
  Check_all_the_other_parameters;

  /*Update all the control variable*/

  Update_the_cruise_state;

  Update_the_state();

```

```
  Regultion_Check_If_it_is_on_off_satndby
  IfLimitIsReached {

    printf ("CruiseControlNode0x13b6100\n");

    UpdateSpeed_management;
  }
  else if (Just_Initalization) {

    printf ("CruiseControlNode0x13b6160\n");

    Update_speed_to = ZeroSpeed;
  }

  printf ("CruiseControlNode0x13b6280\n");

  ControlTheThrottleOfTheCruiseControlSystem;
  Update_speed and state_management;


  printf("yield\n");
}
```

Listing A.1: Instrumentation: Print Version

A.2 Instrumentation: FLIP_PORT Version

```
CruiseControllerfunction()
{

  bool variables_definition;

  /*more variables for cruise control */

  FLIP_PORT_BIT;

  /*Check the pedal position */

  DetectIfPedalIsPressed;
  variable_1 = FindSpeedLimit(SpeedInput);


  Manage_the_cruise_state_polling_control_variables
  Check_if_brake_is_pressed;
  Check_if_accelerator_is_pressed;
```

```
  Check_all_the_other_parameters;

  /*Update all the control variable*/

  Update_the_cruise_state;

  Update_the_state();

  Regultion_Check_If_it_is_on_off_satndby
  IfLimitIsReached {

    FLIP_PORT_BIT;

    UpdateSpeed_management;
  }
  else if (Just_Initalization) {

    FLIP_PORT_BIT;

    Update_speed_to = ZeroSpeed;
  }

  FLIP_PORT_BIT;

  ControlTheThrottleOfTheCruiseControlSystem;
  Update_speed and state_management;

  FLIP_PORT_BIT;
  yield();
}
```

Listing A.2: Instrumentation: FLIP_PORT_BIT Version

```
        if (min_block_length < 2)
        {
            cerr << "WARNING --- Consider Min block length >= 2" << endl;
        }

        string line;
        vector<string> src_code(1, " ");
        while (getline(src_file, line))
        {
            src_code.push_back(line);
        }
     ostringstream cmd;
    cmd << "clang -c -g -emit-llvm -o " << arg[1] << ".o " << arg[1];
```

```
14
15      if (system(cmd.str().c_str()) == -1)
16      {
17          cerr << "Error executing " << cmd.str() << " to extract the CFG" <<
    endl;
18          return 1;
19      }
20
21      cmd.str("");
22      cmd << "opt -dot-cfg-only " << arg[1] << ".o 2>&1";
23
24      FILE * cfg_cmd = popen (cmd.str().c_str(), "r");
25      char buf[1024];
26      vector<string> dot_files;
27      while (fgets(buf, sizeof(buf), cfg_cmd) != NULL)
28      {
29          const string line(buf);
30          if (line.find(".dot'") != string::npos)
31          {
32              istringstream in(line);
33              string discard, dot_file;
34              getline (in, discard, '\'');
35              getline (in, dot_file, '\'');
36              dot_files.push_back (dot_file);
37          }
38      }
39
40
41      ofstream instrumented_portbit ((string(arg[1]) + ".instr_fpb.c").c_str()
    );
42      ofstream instrumented_printf ((string(arg[1]) + ".instr_print.c").c_str
    ());
43
44      if ((!instrumented_portbit) || (!instrumented_printf))
45      {
46          cerr << "Could not write to output file" << endl;
47          return 1;
48      }
49
50      vector<string> src_instr_fpb (src_code),
51                     src_instr_print (src_code);
52
53          // First instrumented line (will need to place an #include before
    this one)
54      int first_line = src_code.size() - 1;
```

```
55
56
57      Control_flow_graph get_cfg (const string & dot_file, const vector<string
        > & src_code);
58
59      for (vector<string>::const_iterator f = dot_files.begin(); f !=
        dot_files.end(); ++f)
60      {
61          Control_flow_graph cfg = get_cfg(*f, src_code);
62          cfg.collapse_short_nodes (min_block_length);
63
64          for (Control_flow_graph::const_node_iterator node = cfg.nodes_begin
            (); node != cfg.nodes_end(); ++node)
65          {
66              if (node->line() > 0 && node->line() < first_line)
67              {
68                  first_line = node->line();
69              }
70
71              if (node->length() > 1)
72              {
73                  src_instr_fpb[node->line()] = "FLIP_PORT_BIT; " +
                    src_instr_fpb[node->line()];
74                  src_instr_print[node->line()] = "printf (\"" + prefix + node
                    ->id() + "\\n\"); " + src_instr_print[node->line()];
75              }
76          }
77
78          ofstream out_dot (((*f) + ".instr.dot").c_str());
79          if (out_dot)
80          {
81              cfg.write_dot (out_dot);
82          }
83      }
84
85      cout << "SEARCHING for empty line from line " << first_line << "
        backwards" << endl;
```

Listing A.3: Intrumentation of source code