

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Towards Sustainable IoT in Energy-Constrained Edge Networks: Context-aware
Anomaly Detection Frameworks Using an Efficient Data Collection Approach**

AHMAD SHAHNEJAT BUSHEHRI

Département de mathématiques et génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
GÉNIE INDUSTRIEL

Novembre 2024

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Towards Sustainable IoT in Energy-Constrained Edge Networks: Context-aware
Anomaly Detection Frameworks Using an Efficient Data Collection Approach**

présentée par **Ahmad SHAHNEJAT BUSHEHRI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Soumaya YACOUT, présidente

Samira KEIVANPOUR, membre et directrice de recherche

Gabriela NICOLESCU, membre et codirectrice de recherche

Heng LI, membre

César MARCON, membre externe

DEDICATION

*To my beloved parents, **Soudabeh** and **Morteza**, whose unconditional love, endless sacrifices, and unwavering support have been the foundation of all my achievements. You gave me everything I needed to dream, strive, and succeed.*

*To my dear brother, **Hamed**, whose love, support and encouragement have been a constant source of strength. Your kindness and presence mean the world to me.*

*To my grandparents, **Zahra**, **Ahmad**, **Fatemeh**, and **Karim**, who always dreamed of seeing this day. Your love and faith in me have guided me through every step of my life.*

Your influence remains, and your memory continues to inspire and uplift me every day.

*To **Khadijeh**, **Mohammad**, and **Ali Akbar**, whose love, guidance, and care have been like that of parents to me. Your persistent support and generosity have profoundly shaped my journey, and I am forever grateful for your presence in my life. FOREVER!*

*To **Sina**, my cousin, whose memory lives on in our hearts. Your warmth and the joy you brought to those around you will never be forgotten.*

*To **Parnian**, whose presence has been a quiet source of calm and light along the way. Your love and kindness have given me the support and peace I needed through every step.*

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to the following individuals and groups for their invaluable support, guidance, and kindness, which have had a profound impact on my Ph.D. journey:

- Samira and Gabriela, for their exceptional mentorship, insightful feedback, and encouragement throughout this journey. Your guidance has been a cornerstone of my academic and personal growth.
- Jamshid Sabet Hemmatabadi, Prof. Nasser Mozayani, and Prof. Hashem Mashhoun the teachers who changed my life. I owe them any progress I may achieve and remain eternally grateful.
- Adel and Ashkan, for their exceptional technical expertise and innovative solutions were pivotal to the success of this work.
- Mohammad Esmaeel, Sahba, and Mohsen three of the nicest people I have ever met. Their kindness and companionship enriched my life in countless ways.
- Felipe, for his invaluable guidance, friendship, and constructive feedback, which played a critical role in shaping this research.
- Francois Juteau, for being a true friend during both the happiest and most challenging moments of this journey.
- Anas, an old friend whose friendship has remained strong despite the passage of time. His support has always been a source of comfort.
- Prof. Michel Dagenais, one of the most influential figures in my professional career. I learned so much from his profound knowledge and guidance, which left a lasting impact on my growth.
- Amin Nikanjam, Hamed Maaref, and Ehsan Rezagholizadeh for their effective support, friendship, and collaborative spirit.
- Vahid, the best friend I could have ever asked for. His deep presence in my heart, cheerful memories, and support have inspired me to become a better person.
- The members of Poly Circle Lab and HES Lab, for creating an inspiring and collaborative environment that encouraged growth and exploration.

RÉSUMÉ

L'utilisation généralisée de systèmes embarqués multi-capteurs dans les réseaux IoT alourdit considérablement la consommation énergétique à la périphérie du réseau. Les approches actuelles de détection de la surconsommation d'énergie reposent sur des dispositifs externes tels que des prises intelligentes. Elles sont principalement conçues pour une surveillance énergétique globale au niveau des bâtiments plutôt que pour les dispositifs informatiques. Ces approches ne parviennent généralement pas à répondre aux besoins énergétiques précis des nœuds IoT individuels, où la collecte de données précises est cruciale. Cette thèse comble une lacune essentielle dans la surveillance de l'efficacité énergétique en introduisant un cadre d'analyse des performances adapté aux nœuds IoT à ressources limitées à la périphérie. Plus précisément, il permet la surveillance interne et externe des opérations énergivores dans les capteurs et services à des fins de dépannage.

Premièrement, le traçage est l'une des techniques efficaces pour collecter des données granulaires issues de systèmes complexes multi-threads. Un traçage de programme consiste en une série d'événements écrits par des sondes. Les techniques de sondage traditionnelles dans les systèmes multi-threads reposent fortement sur l'instrumentation en espace noyau plutôt que sur des techniques efficaces en espace utilisateur dans les processus en cours d'exécution. Par conséquent, le succès et les performances de l'insertion et de l'exécution des sondes dépendent fortement de l'efficacité de celles-ci. Les techniques de sondage existantes sont limitées en portée car elles sont coûteuses en termes de surcharge et intrusives lors de l'exécution. Pour résoudre ce problème, nous introduisons un cadre appelé NOProbe (sonde basée sur NOP), implémentant deux nouvelles techniques d'instrumentation en espace utilisateur pour l'architecture x86. Au cours de nos expériences, NOProbe a atteint des taux d'efficacité de sondes allant de 97% à 99%, surpassant de loin les approches conventionnelles des outils existants comme GDB, qui ont obtenu des taux allant de 0% à 11%.

Deuxièmement, la plupart des solutions de surveillance énergétique actuelles dépendent des compteurs intelligents, qui ne mesurent que la consommation d'énergie globale d'un appareil. Il est généralement difficile d'obtenir des informations spécifiques sur les opérations internes des nœuds IoT. En utilisant l'approche de collecte de données introduite par NOProbe, une nouvelle architecture pilotée par l'IA est proposée pour collecter des données de traçage fines à partir de l'état interne des nœuds suspects sur le plan énergétique à la périphérie. Le résultat de cette étape est alimenté dans un cadre d'analyse des données pour détecter toute anomalie comportementale, telle qu'une consommation d'énergie excessive par rapport aux

seuils. Enfin, en fonction de l'identification des nœuds énergivores par l'étape de détection d'anomalies, certains capteurs et services peuvent être arrêtés pour répondre aux attentes essentielles d'autres éléments. Avec des taux de précision allant jusqu'à 95%, et des scores de précision et de rappel aussi élevés que 96% et 95%, le modèle détecte systématiquement les appels système clés dans les cas de surconsommation d'énergie. Des appels système tels que *poll*, *futex*, *recvmsg*, et *listen* ont montré de fortes corrélations avec une consommation énergétique élevée, car ils sont fortement impliqués dans des opérations intensives en temps ou en I/O qui nécessitent une puissance de traitement importante.

Troisièmement, les approches actuelles d'optimisation énergétique se concentrent principalement sur la réduction de la consommation énergétique des unités de traitement, via le cyclage d'activité, l'échelle dynamique de tension ou des conceptions matérielles à faible consommation. Cependant, ces stratégies ne prennent souvent pas pleinement en compte la communication réseau comme une source majeure de déperdition d'énergie dans les nœuds IoT. La collecte de données grossières provenant des opérations de communication de données externes peut également être une approche complémentaire pour révéler les comportements de surconsommation parmi les éléments de la périphérie. À cette fin, nous avons développé un nouveau cadre de détection d'anomalies énergétiques des nœuds IoT à l'aide de l'analyse de transmission des données. Un tel cadre d'analyse des performances permet aux dispositifs d'adapter leur configuration en fonction de la qualité dynamique du signal et des paramètres de transmission. Enfin, un retour d'information sur la stabilisation énergétique donne aux nœuds un aperçu pour modifier leur configuration de transmission pour les communications futures. Sur la base des résultats de la détection d'anomalies énergétiques, ce cadre a atteint des performances élevées avec une précision de 94,8%, un score de précision de 95,2%, un score F1 de 95,0%, surpassant d'autres solutions au prix d'une augmentation de 14% et 17% du temps d'entraînement et de test, respectivement.

Ces contributions proposent collectivement des cadres de détection d'anomalies énergétiques contextuellement intelligents, orientés vers un réseau périphérique contraint en énergie et durable grâce à une solution de collecte de données efficace.

ABSTRACT

The widespread use of multi-sensor embedded systems in IoT networks burdens energy consumption at the network’s edge. Existing energy over-consumption detection approaches rely on external devices like smart plugs. They are primarily designed for broad, building-level energy tracking rather than computing devices. These approaches typically fail to address the fine-grained energy demands of individual IoT nodes, where precise data collection is essential. This thesis addresses the gap in energy monitoring by introducing a performance analysis framework tailored for resource-constrained IoT nodes at the edge. Specifically, it enables internal and external monitoring of energy-intensive operations in sensors and services for troubleshooting purposes.

First, tracing is a technique for collecting granular data out of complex multi-threaded systems. A program trace consists of a series of events written by probes. Traditional probing techniques in multi-threaded systems rely on kernel-space instrumentation rather than efficient user-space ones in running processes. Hence, the success and performance of probe insertion and execution are affected by the efficiency of the probe. Existing probing techniques are limited in scope as they are costly in overhead and intrusive in execution. To solve this challenge, we introduce a framework called NOProbe (NOP-based Probe), implementing two new user space instrumentation techniques for the x86 architecture. Through our experiments, NOProbe achieved success rates for inserting fast probes using 5-byte call instructions at 97% to 99% of target probe locations, far surpassing conventional approaches in existing tools like GDB, which achieved rates between 0% to 11%.

Second, most existing energy monitoring solutions depend on smart meters, which only measure the overall energy consumption of a device. Providing insights into specific operations within IoT nodes is usually hard to achieve. Using the data collection approach introduced by NOProbe, a new AI-driven architecture is proposed to collect fine-grained tracing data from the internal state of energy-suspicious nodes on the edge. The output of this step is fed to a data analytic framework to detect any behavioral anomalies such as over-threshold energy consumption. Finally, depending on identifying the energy-hungry nodes by the anomaly detection step, some sensors and services might be halted, to meet the essential expectations for others. With accuracy rates of up to 95%, and precision and recall scores as high as 96% and 95%, the model consistently detects key system calls in energy over-consumption. System calls such as *poll*, *futex*, *recvmsg*, and *listen* showed strong correlations with high energy consumption, as they are heavily involved in time-intensive or I/O operations that

demand significant processing power.

Third, current energy optimization approaches primarily focus on reducing the energy consumption of processing units, through duty cycling, dynamic voltage scaling, or low-power hardware designs. However, these strategies often fail to fully consider network communication as a dominant source of energy drain in IoT nodes. Collecting coarse-grained data from the external data communication operation can also be a complementary approach to reveal the over-consumption behavior among edge elements. To this end, we developed a novel framework for energy anomaly detection of IoT nodes using data transmission analysis. Such a performance analysis framework enables devices to adapt their configuration according to the dynamic signal quality and transmission settings. Finally, energy stabilization feedback gives nodes insight into changing their transmission configuration for future communication. Based on the result of anomalous energy detection, this framework achieved high performance with 94.8% accuracy, 95.2% precision score, and 95.0% F1 score, overperforming other solutions at the cost of 14% and 17% increase in training and testing time, respectively.

These contributions collectively propose context-aware energy anomaly detection frameworks toward a sustainable energy-constrained edge network using an efficient data collection solution.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	ix
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF SYMBOLS AND ACRONYMS	xv
CHAPTER 1 INTRODUCTION	1
1.1 Challenges for Energy Performance Analysis on Edge	3
1.2 Research Objectives	7
1.3 Contributions	8
1.4 Industrial Application	10
1.5 Thesis Outline	11
1.6 Publications	12
CHAPTER 2 LITERATURE REVIEW	13
2.1 General-purpose Energy Consumption Modeling	13
2.1.1 Dynamic Energy and Thermal Models in Multi-core Systems	14
2.2 Edge/cloud-based Energy Consumption Modeling	18
2.2.1 Node-Based Solutions	18
2.2.2 Cloud-Based Solutions	20
2.3 Instrumentation	23
2.3.1 Dynamic Instrumentation	24
2.4 Tracing	25
2.5 ML-based Anomaly Detection Approaches	26
2.5.1 Unsupervised Learning	26
2.5.2 Supervised learning	28
CHAPTER 3 METHODOLOGY	30

CHAPTER 4	ARTICLE 1: "NOPROBE: A NOP-BASED DYNAMIC BINARY INSTRUMENTATION FRAMEWORK FOR EFFICIENT DATA COLLECTION USING BINARY REWRITING"	34
4.1	Introduction	35
4.2	Background	38
4.2.1	Instrumentation Approaches	39
4.2.2	Cross-modification Challenges	42
4.2.3	Approaches for Safe Code Insertion	43
4.2.4	Compiler Optimization and Semantic NOPs	44
4.2.5	Out-of-Line Execution (OLX) Buffer	48
4.3	Related Work	49
4.3.1	Instrumentation Tools	49
4.4	NOProbe Design and Architecture	51
4.4.1	Assumptions	51
4.4.2	Multi-strategy Probing	52
4.4.3	Static Analysis	61
4.4.4	Safe Patching Setup (lock-redirect-load-arm)	62
4.5	Results	65
4.5.1	NOPs Accessibility	66
4.5.2	Probe Effectiveness	67
4.5.3	Probe Insertion Performance	69
4.5.4	Probe Execution Performance	71
4.6	Conclusion	73
CHAPTER 5	ARTICLE 2: "DEEP LEARNING-DRIVEN ANOMALY DETECTION FOR GREEN IOT EDGE NETWORKS"	75
5.1	Introduction	75
5.2	Background and Related Works	78
5.2.1	Node-based Solutions	78
5.2.2	Cloud-based Solutions	80
5.3	Problem Statement	82
5.4	Data Description and Mathematical Analysis	83
5.4.1	Feature Correlation	84
5.4.2	Mathematical Analysis	85
5.5	Methodology	87
5.5.1	Data Labeling	88

5.5.2	Data Normalization	89
5.5.3	Feature Importance	90
5.5.4	Loss Function Design	91
5.5.5	Deep Unsupervised Learning Model for Anomaly Detection	92
5.6	Experiment	93
5.6.1	Experiment Design	94
5.6.2	Result and Discussion	95
5.7	Conclusion	100
CHAPTER 6 ARTICLE 3: "ANOMALOUS ENERGY DETECTION FOR RESOURCE- CONSTRAINED EMBEDDED SYSTEMS USING TRACING DATA ANALYSIS"		102
6.1	Introduction	102
6.2	Literature Review	104
6.2.1	Dynamic Instrumentation	104
6.2.2	Tracing	104
6.3	Methodology	106
6.3.1	Data Labeling	107
6.3.2	Proposed Classification Framework	108
6.4	Experiments and Results	109
6.4.1	Experimental Framework	109
6.4.2	Results of Experiments and Discussion	109
6.5	Conclusion and Future Work	113
CHAPTER 7 DISCUSSION		115
7.1	Limitations and Future Research	118
CHAPTER 8 CONCLUSION		122
REFERENCES		124

LIST OF TABLES

Table 2.1	Energy saving mechanisms in IoT domain	23
Table 4.1	Optimization-level-activated flags at -O2 and -O3 for code alignment in GCC	45
Table 4.2	NOPs, 1-byte, and 2-byte pseudo-NOPs in x86 architecture [1]	45
Table 4.3	manually-activated flags used for compiler-assisted techniques	50
Table 4.4	Evaluation Machine Specifications	65
Table 4.5	Number of successful fast probing across different benchmarks	69
Table 5.1	Relevant energy saving mechanisms in IoT domain	82
Table 5.2	Parameters in energy consumption modelling	86
Table 5.3	Protocol datasheet	89
Table 5.4	Comparison between the baseline clustering algorithms and the pro- posed modified VAE	99
Table 6.1	Labeling strategy for three datasets	108
Table 6.2	Structure of datasets	109
Table 6.3	The most highly variable features of all datasets and the associated descriptions	110
Table 6.4	Experimental results for the first dataset	113
Table 6.5	Experimental results for the second dataset	113
Table 6.6	Experimental results for the third dataset	113

LIST OF FIGURES

Figure 1.1	Sensor devices communicating to Cloud through gateways (drones) .	2
Figure 1.2	SDs communicating to edge devices (helicopters) through gateways (drones)	2
Figure 1.3	Illustrative overview for articles	10
Figure 3.1	Edge computing scenario with communication-intensive IoT nodes and processing-intensive edge devices (helicopters)	30
Figure 3.2	Collecting data from an instrumented kernel, applications, virtual machines, and containers	31
Figure 3.3	A context-aware energy anomaly detection framework using internal data collection	33
Figure 3.4	A context-aware energy anomaly detection framework using external data collection	33
Figure 4.1	Non-thread-safe patching of the short instruction no. 1 with a 5-byte probing instruction	36
Figure 4.2	General definitions for patching a long instruction before (top figure) and after (bottom figure) the operation	38
Figure 4.3	Original instruction(s) replacement by a probing instruction in Trap-based, Branch-based, and Tranch-based approaches	40
Figure 4.4	Function prologue layout before and after the instrumentation	41
Figure 4.5	Loop entry layout before and after the instrumentation	42
Figure 4.6	Cache straddling	42
Figure 4.7	Code alignment using NOPs	46
Figure 4.8	NOProbe execution flow with a 2-byte <i>jmp</i> as probing instruction . .	53
Figure 4.9	Embedding a 5-byte <i>call</i> in an 8-byte NOP-padding	56
Figure 4.10	Inserting a 5-byte <i>call</i> in a 7-byte NOP-padding	56
Figure 4.11	Probe insertion at <i>II</i> using the conventional and the proposed tranch-based approaches on three popular instruction patterns	60
Figure 4.12	Lock the probe site and redirect threads to the OLX buffer	63
Figure 4.13	Patching the probe site with a 5-byte probing instruction while ensuring serialization	64
Figure 4.14	The distribution of reachable and unreachable NOPs at different lengths in LLVM-ar and Clang	66
Figure 4.15	The inter-NOPs distance distribution in LLVM-ar	67

Figure 4.16	Probe insertion cost versus the number of running threads	70
Figure 4.17	Probe insertion cost versus the average function size of multiple binaries	71
Figure 4.18	Probe execution overhead for Uprobe (trap-based), NOProbe, GDB (trap-based and branch-based), Dyntrace (trap-based and branch-based), and Dyninst	72
Figure 5.1	Covariance matrix of features	84
Figure 5.2	IoT nodes communicating with the gateway at the edge	85
Figure 5.3	The proposed energy-aware Framework	88
Figure 5.4	The significance of the features' impact on the energy consumption of nodes	91
Figure 5.5	A comparison between performance metrics and execution time for 2-layer VAE and the proposed 2-layer modified VAE using different activation functions: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time	96
Figure 5.6	A comparison between performance metrics and execution time for 3-layer VAE and the proposed 3-layer modified VAE using different activation functions: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time	97
Figure 5.7	A comparison between performance metrics for 2-layer VAE and the proposed 3-layer modified VAE using ReLu activation function: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time	98
Figure 6.1	The complete architecture for the proposed anomaly detection framework	107
Figure 6.2	Heat map- The correlation between features in the first dataset . . .	112
Figure 6.3	Heat map- The correlation between features in the second dataset . .	112

LIST OF SYMBOLS AND ACRONYMS

IoT	Internet of Things
DPM	Dynamic Power Management
DVFS	Dynamic Voltage-Frequency Scaling
DEM	Dynamic Energy Management
DTM	Dynamic Thermal Management
PMC	Performance Monitoring Counters
LLVM	Low-Level Virtual Machine
TX	Transmit
RX	Receive
RSS	Received Signal Strength
DPC	Dynamic Power Control
TPC	Transmit Power Control
RSSI	Received Signal Strength Indicator
LQI	Link Quality Indicator
ML	Machine Learning
DBI	Dynamic Binary Instrumentation
RTOS	Real-Time Operating System
LSTM	Long Short-Term Memory
AQP	Approximate Query Processing
LOF	Local Outlier Factor
MNN	Mutual K Nearest Neighbors
RL	Reinforcement Learning
LR	Logistic Regression
SVM	Support Vector Machine
DT	Decision Tree
KNN	K-Nearest Neighbors
GMMLDA	Gaussian Mixture Model
LDA	Linear Discriminant Analysis
PSO	Particle Swarm Optimization
SD	Sensor Devices
NOProbe	NOP-based Probe
SMP	Symmetric Multi-Processing
Tranch	Trap-branch approach

MSB	Most Significant Byte
GPF	General Protection Faults
NOP	No-Operation
PC	Program Counter
JNS	Jump if Not Signed
OLX	Out of Line eXecution
JCC	Jump Condition Code
IP	Instruction Pointer
I1	Instruction 1
ELF	Executable and Linkable Format
IPI	Inter-Processor Interrupt
IRET	Interrupt RETurn
TSC	Time Stamp Counter
IPA	In-Process Agent
IPC	Inter-Process Communication
RCU	Read-Copy-Update
DM	Decision-Making
MCU	Micro-Controller Unit
P2P	Point-to-Point
MSE	Mean Squared Error
MSLE	Mean Squared Logarithmic Error
T	transmission Time
P	transmission Power
D	transmission Distance
VAE	Variational Auto-Encoder
ReLu	Rectified Linear unit
AI	Artificial Intelligence
AIoT	AI-integrated IoT
DL	Deep Learning
QoS	Quality of Service
RTOS	Real-Time Operating System
DAG	Directed Acyclic Graph
CSMA	Carrier Sense Multiple Access
CA	Collision Avoidance
SN	Sensor Networks
MQTT	Message Queuing Telemetry Transport

TLS	Transport Layer Security
SSL	Secure Sockets Layer
DTLS	Datagram Transport Layer Security
Intel PT	Intel Processor Trace
IIoT	Industrial Internet of Things

CHAPTER 1 INTRODUCTION

Data collection and processing in the traditional cloud computing scenario is performed in a central location. Sensor devices send their data to cloud servers through gateways (see drones in Fig. 1.1), where it is analyzed, and decisions are made. This model facilitates scaling while sharing resources, but it also has some limitations such as delays in processing that incur high latency. Constraints on data transfer in this model are imposed due to its high bandwidth consumption. Moving to edge computing, a better compromise on resource consumption is devised by processing data closer to where it is collected. This decentralization makes the system more efficient and responsive.

Industrial domains provide a unique opportunity for taking advantage of edge computing by employing network virtualization, wireless sensor technologies, and smart decision-making approaches. An Internet of Things (IoT) scenario on edge is defined through a set of wireless sensor networks connecting a collection of widely distributed monitoring sensor devices (SD) [2]. One of the main advantages of these resource-constrained SDs located in hostile environments [3] is forwarding the collected data to the edge gateways [4] for further analysis [5] and decision-making (DM) [6]. Such tiny SDs also constrain hardware resources, such as stripped RAM size, low-rank processing capability, and limited energy harvesting [7]. However, they are well-adopted in autonomous systems for steady data collection, stable transmission [8], continuous diagnosis [9], and real-time analysis [10].

The emergence of edge computing in IoT introduced edge devices (see unmanned helicopters in Fig. 1.2) closer to the network edge than to cloud data centers. Edge devices feature heterogeneous hardware/software capabilities such as operating systems, service stacks, and architectural interoperability. Those devices that feature higher computation power, larger memory space, and enhanced potentials than the other nodes are designated as edge servers. The recent development of edge computing has revolutionized the capabilities of edge devices, enabling them to offer much more complex and flexible functionalities such as local real-time decision-making, transmission relay, content aggregation, and edge caching [11] [12]. Considering the computation power of edge devices, edge computing makes a suitable context for optimized utilization of resources [13] [14]. Network bandwidth [11], communication latency, and data migration are the three major resources that are effectively influenced by the emergence of the edge computing domain. Since the distance between the edge network and the data collection point (SD) is minimized, the bandwidth is largely preserved while the response time is shrunk. Therefore the quicker the response gets to the user, the higher the

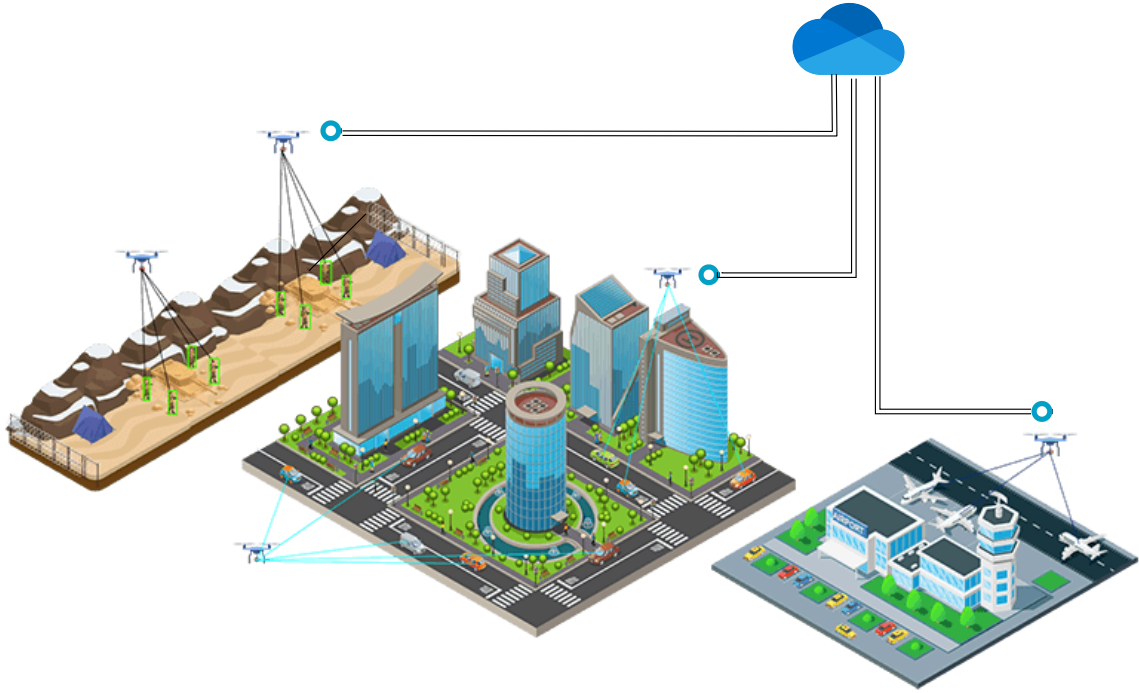


Figure 1.1 Sensor devices communicating to Cloud through gateways (drones)

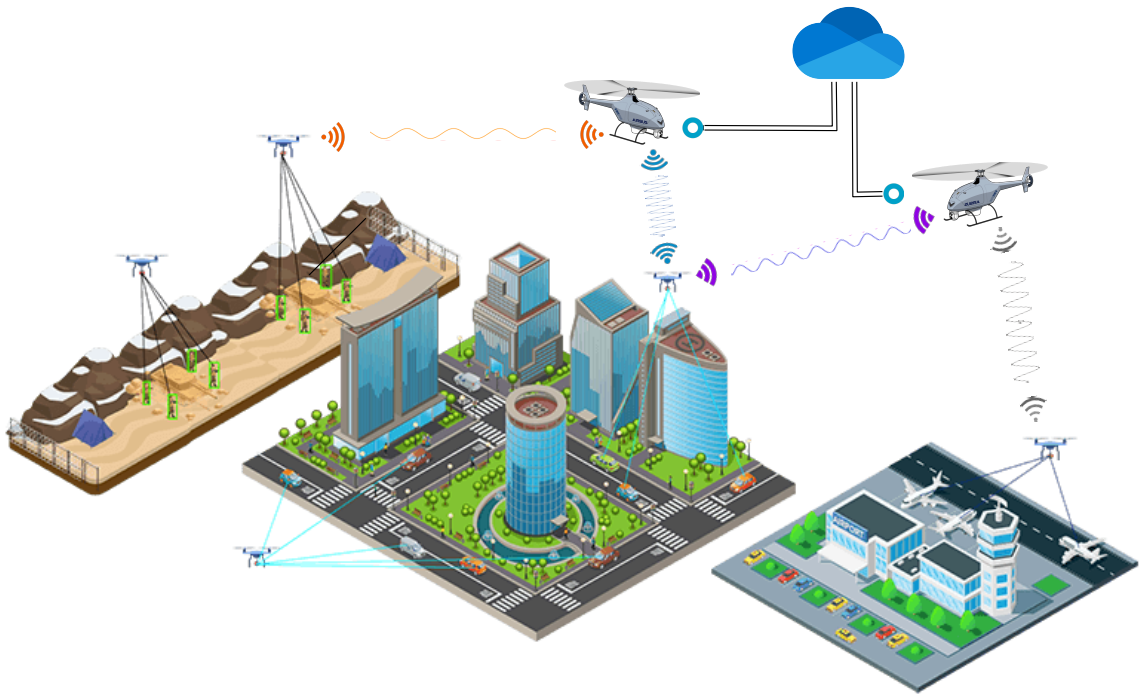


Figure 1.2 SDs communicating to edge devices (helicopters) through gateways (drones)

user satisfaction in time-critical services would be. This becomes even more beneficial for the services in the mobile edge computing (MEC) domain to be provided with close-to-real-time responses [15].

1.1 Challenges for Energy Performance Analysis on Edge

Energy consumption is counted as one of the most challenging constraints for service provisioning in SDs and edge devices. Energy over-consumption in SDs occurs in cases like amplifying the transmission power for fading/shadowing avoidance and increasing the rate of forwarding data in a congested multi-hop network eventually leading to energy failure. Similarly, when SDs are located in remote areas and distant from edge gateways, their vulnerability to increased path loss becomes more pronounced, particularly in terms of energy efficiency. Similarly, as resource-constrained edge devices get more complex, they are resistant to resolving unexpected performance anomalies [16]. Since there are many reasons for system energy performance dropping such as service resource overloading, it would be a complicated task to inspect and detect the system execution flaws manually.

The mismanagement of energy consumption in both sensors and edge devices can result in overheating. This can occur because, in multiprocessor systems, concurrent tasks are often distributed non-uniformly, creating regions of high energy demand that generate localized heat. Non-uniform heat generation can cause spatial temperature gradients across the chip, as different areas of the system experience varying loads depending on task allocation. The spatial temperature gradients and temporal thermal gradients besides the thermal cycles are three main factors that degrade the performance and the efficiency of a system during its lifetime. Ignoring overheating may also cause performance, durability, and reliability loss. For example, raising the temperature by ten to twenty degrees in metallic materials would shorten the useful life of the components by sixteen times [17].

Most software-layer energy optimization models perform at the levels of source code, algorithm, and software architecture [18] [19]. The outcome of these energy minimization approaches is categorized into multiple groups for embedded computing devices [20]. The first is called Dynamic Power Management (DPM) [21]. DPM temporarily halts any idle element of the system to avoid wasting energy, like processing cores. In the second group, Dynamic Voltage-Frequency Scaling (DVFS) permits CPUs to switch between different frequency-voltage levels to scale down energy consumption [22]. To save a significant amount of energy expended on generic tasks in the third group, CPUs are customized to meet the requirements of the tasks on a multiprocessor system-on-chip [23]. The same customization happens for the fourth group but for the cache-based memory access [24]. By mapping tasks to different

processors, the fifth group performs load balancing across all the cores which results in the utilization enhancement of processing elements on a multiprocessor system-on-chip [25].

In general, energy consumption evaluation techniques are categorized into measurement-based, simulation-based, and model-based strategies [26]. Unfortunately, the first two approaches do not comply with the nature of dynamic service execution on remote devices. The first strategy requires physical access to nodes at measurement time while the second one cannot pinpoint runtime transient errors with confidence. On the other hand, the third approach can track application energy consumption during software execution without relying on energy measurement hardware or simulators. The third approach uses a behavioral system analysis to examine the energy workload efficiency on a machine automatically [27]. By monitoring the running service responses to the system actions, this approach can diagnose potential energy anomalies.

Operating at different layers, model-based energy evaluation approaches leverage resource utilization (physical hardware), program analysis (user space), and system calls (kernel space). Utilization-based modeling estimates the energy consumption of a system by considering hardware parameters, such as CPU load, disk read/write accesses, and network transmission rate. Discovering the correlation between hardware utilization and energy consumption forms the basis of this approach. Program-analysis-based modeling utilizes code evaluation in user space that mainly performs on the assembly instruction level. As the code instructions are executed sequentially by the processor, the consumed energy by an individual instruction is proportional to the number of cycles it takes to execute [28] [29]. Therefore, the overall energy consumption of a running code is obtained by the collection of the consumed energy for the execution of comprised instructions in the code. In the context of system-call-based modeling, system calls play a key role in representing the resource allocation demand within the system. System calls are the primary mechanism for user space applications to request access to kernel-managed hardware resources.¹ Consequently, system-call-based energy modeling captures energy state variations associated with system call invocations and the hardware interactions they trigger. These variations depend on the type of the required resource without having prior knowledge about system workload.

System-call-based and utilization-based modelings take advantage of the tracing mechanism during runtime as a fundamental technique in data collection. They require tracing tools to collect hardware counters and system call invocation data. Generally, practitioners use tracing tools to discover the target process execution state, collect granular performance-related

¹However, it is important to note that user-space applications can also interact directly with hardware components in user space (non-kernel time slices) through memory-mapped I/O, direct access to device buffers, or specialized libraries bypassing system calls.

events, and observe performance problems in run-time. Tracing should not significantly perturbate the system; otherwise, many issues cannot be captured properly. Imposing negative performance impact, consuming additional resources, introducing concurrency overhead, disturbing timing orders, and creating Heisenbug are some of those issues. Tracing fulfills data collection by providing a comprehensive view of how a program behaves during runtime, enabling various analysis and optimization tasks.

Tracing-based data collection for an application in run-time requires binary modification techniques to inject probe points into the executable. Instrumentation may be performed on the source code statically before the execution, or on the binary dynamically after the execution has started. In most cases, a probe or tracepoint is a fast user-space function call or a slow kernel-based trap inserted into the code. It could be inserted into the source code at compile-time, or into the binary before or after the execution has started. Dynamic binary instrumentation (DBI) is the process of manipulating an executable code, at the instruction level granularity, to offer some functionalities like tracing. There is no need to recompile the target program as dynamic instrumentation uses dynamic code injection mechanisms. Multiple techniques can be used for DBI to insert probes and user-supplied analysis routines at run-time into the code. As a separate process, the instrumentation code injector program controls the target and provides runtime support while injecting code for tracepoints. The runtime support is handled by an instrumentation agent that is loaded in the same address space as the running target program. The instrumentation agent controls the tracepoints and receives commands from the instrumentation code injector program. For example, the GNU Debugger [30] can inject an in-process agent into a running program to insert low-overhead conditional tracepoints. It is worth noting that DBI as a source code-independent technique is used not only for debugging and tracing but also in other domains like software security and testing [31].

The x86 is one of the most popular platforms in production systems. Unlike ARM and MIPS architectures with fixed-length instructions, x86 supports variable-length instructions. Instructions in x86 can take one to several bytes to accommodate a wide range of operations and addressing modes. In most cases, common instructions have shorter and more complex encoding. This may shrink the code size, permit a smaller instruction cache, and save clock cycles, and energy [32]. However, DBI tool developers have been struggling with multiple challenges associated with this architecture that add complexity to instrumentation operations by introducing extra memory access and performance bottlenecks. As an example, variable-size instructions are not aligned to either word, cache line, or even page boundaries [33] [34]. This can lead to cache coherence issues, particularly for code cross-modification in multi-core environments. To prevent such problems, Intel proposed a costly

cache coherency control mechanism [35]. Another challenge for DBI with variable-length instructions is the length of probing instruction. It may overlap subsequent neighbor instructions if its size (2-byte or 5-byte) exceeds that of the original instruction at the probe site. If a branch lands on the overlapped instruction(s), the processor will try to interpret the bytes in the middle of the probing instruction as a valid instruction. Such branches land in an unintended memory region leading to crashes, security vulnerabilities, or undefined program behavior. The concurrency in multi-threaded systems compounded with variable-size instructions also adds significant complexity to DBI in x86 architecture. For instance, when a probing instruction replaces several original instructions, additional measures must be taken for a thread that is running, preempted, or interrupted in the patching area. In light of the existing limitations, this thesis investigates a new user-space dynamic instrumentation framework as a data collection structure for x86 architecture. We look over novel strategies to instrument any multi-byte instruction in optimized codes. We also explore solutions to discover compiler-agnostic strategies targeting the weakness of current jump-based techniques in trampoline placement.

Applying existing energy optimization solutions on a peer-to-peer network is not effective in reducing energy consumption for SDs and edge devices along with alleviating the consequences. These approaches do not consider transmission energy consumption as the primary cause of battery depletion for SDs in a wireless sensor network [36]. On the other hand, it remains challenging to develop a unified energy performance analysis framework that accommodates diverse architectures, computational capabilities, energy profiles, and workload types. Each edge device, from simple micro-controllers to advanced multi-core processors, has its energy consumption characteristics. Although sensor devices are typically designed as a single processor board, most existing approaches are devised for multiprocessor system-on-chips. For example, over-heat generation is much more prone to happen in multiprocessors than single processor SDs as the former suffers from non-homogeneous heat dissipation in its processing elements. The second issue is that multi-core processors often use Dynamic Voltage and Frequency Scaling (DVFS) to optimize energy consumption based on workload, while this technology does not apply to simpler devices. As a result of this diversity, there is no one-size-fits-all energy model that can fit all architectures. These devices have varying computational capabilities, which further complicates energy modeling. During local computations, high-performance devices might consume a substantial amount of energy, whereas less powerful devices might use a greater level of energy during communication tasks because of offloading requirements. It can also be difficult to model energy accurately because of the dynamic and unpredictable nature of workloads at the edge. Edge devices are susceptible to unexpected spikes in computational tasks like those in processing-intensive data streams.

Energy models are required for these workload variations with adaptive and responsive capabilities against real-time fluctuations.

Additionally, the operational contexts of edge devices such as whether they are battery-powered or not, determine their energy profile. While battery-powered devices give higher priority to energy efficiency (to extend operational lifetime), plug-in devices target performance maximization instead. Such pluralism in planning requires specific context-aware energy management strategies tailored to the individual needs of each device.

By achieving comprehensive data collection, we can extract fine-grained real-time monitoring data from SDs and edge devices to evaluate energy footprint. By applying DBI on edge devices and collecting internal traces for system call invocations in each time window, we aim to devise an energy anomaly detection framework. As the detailed energy information of a running device can reveal its internal status at each instant, it can also provide a rough prediction of the energy state of the device shortly and help authorities indicate the best course of action in monitoring the system's performance. Apart from the overloaded edge device with high energy consumption, SDs may also become points of failure in an edge network. A software update causes SDs to enter a high-power mode more frequently than necessary during transmission. Consequently, it reduces the network operational time which recursively leads to a rapid depletion of SD batteries. Similarly, By collecting energy and other transmission data (link quality parameters and configuration settings) together with internal tracing data, we aim to develop a new AI-based model to detect anomalies in energy behavior. This model must avoid supervised learning solutions since data labeling techniques for energy over-consumption detection [37] are not practically feasible in continuous learning for IoT domains. We believe this model may prolong the survival time of the whole network reliably [38].

1.2 Research Objectives

The major objective of this thesis is to develop innovative frameworks and methodologies for improving data collection, energy efficiency, and anomaly detection in resource-constrained IoT elements for edge networks. Through DBI, this research addresses limitations in current methodologies by creating the NOProbe framework capable of efficient data collection with minimal overhead. Additionally, the thesis seeks to advance anomaly detection models, specifically tailored for edge networks, by leveraging both granular tracing data and coarse transmission and link quality data analysis. By integrating these approaches, the research intends to improve the performance of energy monitoring, thus contributing to the sustainability and reliability of IoT infrastructures.

Furthermore, we can formulate the following specific objectives as follows:

- To develop a low-overhead and comprehensive data collection framework
 - Existing probe insertion techniques are limited in scope as they are costly in overhead and intrusive in probe execution. We introduce a framework called NOProbe (NOP-based Probe), implementing two new user space instrumentation techniques for the x86 architecture. NOProbe exploits several complementary strategies to achieve a high success rate in the insertion of fast jump-based probes for short instructions (<5 bytes) compared to other user-space approaches that are unable to instrument them.
- To develop a context-aware energy anomaly detection framework using external transmission and link quality data collection
 - Obtaining accurate energy data from IoT nodes using current techniques without external devices is unfeasible. We introduce a deep-learning-driven framework for energy anomaly detection of IoT nodes using data transmission analysis. We redesign the conventional loss function by enhancing the impact of our dominant features to improve the reliability of the anomaly detection framework.
- To develop a context-aware energy anomaly detection framework using internal tracing data collection
 - Most of the energy over-consumption detection approaches make use of smart plugs or are designed for smart buildings rather than computing devices. We introduce a new architecture to detect any behavioral anomalies such as over-threshold energy consumption using system call analysis.

1.3 Contributions

Following the research objectives stated above, this thesis presents the following original contributions, to the field of system performance analysis:

- Develop a multi-strategy user-space dynamic instrumentation framework as a data collection structure called NOProbe. It can insert a fast probe at almost any instruction location in a program built with compiler optimizations (such as -O2) or even without it (stripped binaries). Through our experiments, NOProbe achieved probe effectiveness

rates² ranging from 97% to 99%, far surpassing conventional approaches in existing tools like GDB, which achieved rates between 0% and 11%.

- In the first strategy, we exploit existing paddings– inserted by compilers for code alignment optimization purposes– as strategic placement points for installing trampolines. To achieve this goal, two new algorithms are introduced to assign paddings to probes within a specified range. One algorithm prioritizes performance and follows a greedy approach, while the other focuses on optimizing the global padding assignment.
 - In the second strategy, we utilize pseudo-NOP instructions to explore a wider range of memory space for the trampoline address allocation.
 - Develop an improved user-space mechanism (lock-redirect-load-arm) that can safely and concurrently patch a basic block of instructions in multi-threaded programs.
- Propose a new energy-aware framework for the detection of energy consumption anomalies that might affect IoT nodes’ lifetime in a wireless sensor network. This framework collects peer-to-peer communication energy and link quality data and uses a deep-learning approach to detect the abnormal energy behavior of the SDs. Based on the result of anomalous energy detection, this framework achieved high performance with 94.8% accuracy, 95.2% precision score, and 95.0% F1 score, overperforming other solutions at the cost of 14% and 17% increase in training and testing time, respectively.
 - Propose a learning solution for anomaly detection based on an unsupervised algorithm. We design a new loss function for the conventional variational autoencoder by analyzing the feature importance of the data using a linear regression model
 - Evaluate the proposed approach by using a public dataset
 - Implement the proposed approach with other baseline clustering algorithms and unsupervised deep-learning models, including multi-layer VAEs, to make a comparison between their performance and that of our proposed solution
 - Propose a new framework for anomaly detection in power consumption of energy-restricted embedded systems. With accuracy rates of up to 95%, and precision and recall scores as high as 96% and 95%, the model consistently detects key system calls in energy over-consumption. System calls such as *poll*, *futex*, *recvmsg*, and *listen* showed strong correlations with high energy consumption, as they are heavily involved in time-intensive or I/O operations that demand significant processing power.

²The number of successfully inserted fast probes using 5-byte call instruction (without using a trap-based approach) divided by the total number of target probe locations.

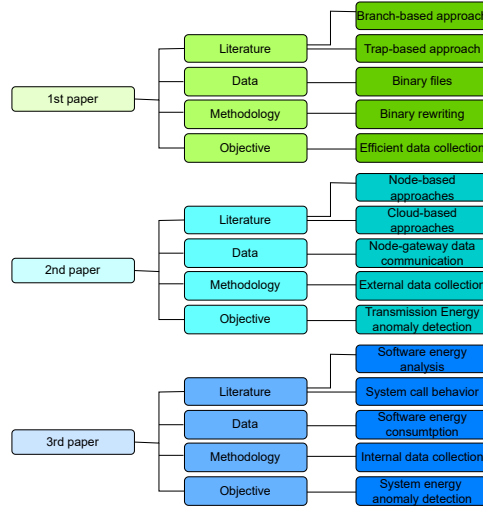


Figure 1.3 Illustrative overview for articles

- Propose a learning solution through classification for anomaly detection using the supervised tree-based and Logistic regression algorithms
- The proposed approach is evaluated by using three public datasets
- The proposed approach is also implemented with some other classification algorithms to observe its performance

The structure of the three papers is illustrated in Fig. 1.3.

1.4 Industrial Application

A fleet of drones is deployed to inspect a vast pipeline system. By using sensors, these drones can detect leaks, temperature changes, and signs of corrosion, and communicate with edge gateways to relay their findings. During the inspection operation, energy consumption is monitored to ensure efficient operation and maximize flight time.

During one of these routine flights, a couple of drones suddenly encountered fast battery depletion while processing images for leak detection. One of the possible root causes of such unexpected energy loss could be a software glitch (such as a loop running more often than it should) or a malfunctioning sensor. To detect the issue, we use NOProbe to instrument the embedded operating system of drones together with suspicious installed software. Through NOProbe, we collect detailed tracing data such as resource utilization (such as CPU, RAM, disk, and NIC usage), in addition to system calls, and function stacks. The internal anomaly

detection framework analyzes the tracing data and concludes that inefficient memory allocation has put unnecessary strain on the CPU.

Another set of drones in the fleet experiences a quick battery drain on the different parts of the pipeline passing through rugged terrain. A potential root cause for this scenario is explained by a high rate of packet losses, delayed transmissions, and persistent re-transmission. Upon collecting transmission parameters and communication quality data with their edge gateway, the external anomaly detection framework identifies the anomaly as congested communication channels and multi-path interference. It suggests switching to an empty or less congested channel using frequency hopping techniques. As a second solution, it recommends boosting transmission power temporarily to strengthen the signal and overcome the interference in wilderness areas.

Apart from the energy over-consumption reported in the other two cases, a subset of drones records remarkably low energy usage compared to their counterparts in similar conditions after completing their assigned tasks. Although energy conservation is desirable for our drones in general, it also raises concerns about their operational effectiveness. By using NOProbe and collecting detailed tracing data, the internal anomaly detection framework diagnoses the issue in those drones. They have been idling for significant periods instead of actively processing sensor data and making transmission to edge gateways. Further analysis detects the root cause as a misconfigured energy-saving mode within their operating systems. Even though we expected the mode to be triggered during low-activity periods to reduce power usage during low-activity periods, the misconfigured mode erroneously caused the drones to enter a low-power state prematurely. The internal anomaly detection framework recommends deploying a software update accordingly.

1.5 Thesis Outline

The rest of the proposal is organized as follows. In Chapter 2, we present the literature review. We introduce the concepts necessary to understand this manuscript and review the different research works related to our objectives. In Chapter 3, we describe the design and the implementation details of the methodology we followed to achieve our research objectives. We present the articles we wrote as part of this research, in Chapters 4, 5, 6. Subsequently, in Chapter 7, we discuss the obtained results in light of the previously defined research objectives. Finally, Chapter 8 concludes the thesis.

1.6 Publications

The thesis is built on the foundation of the papers that have been published or submitted, as mentioned here:

- Ahmad Shahnejat Bushehri, Anas Balboul, Adel Belkhiri, Samira Keivanpour, Gabriela Nicolescu, Michel R. Degenais, "NOProbe: A NOP-based Dynamic Binary Instrumentation Framework for Green Data Collection Using Binary Rewriting" submitted to IEEE Transactions on Dependable and Secure Computing.
- A. Shahnejat Bushehri, A. Amirnia, A. Belkhiri, S. Keivanpour, F. G. de Magalhães and G. Nicolescu, "Deep Learning-Driven Anomaly Detection for Green IoT Edge Networks," in IEEE Transactions on Green Communications and Networking, vol. 8, no. 1, pp. 498-513, March 2024, doi: 10.1109/TGCN.2023.3335342.
- A. S. Bushehri, S. Keivanpour, M. Azam and G. Nicolescu, "Anomalous energy detection for resource-constrained embedded systems using tracing data analysis," 2021 IEEE International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa, 2021, pp. 1-8, doi: 10.1109/ICECET52533.2021.9698794.

CHAPTER 2 LITERATURE REVIEW

This chapter introduces the key concepts in our research context and reviews related work in the field. It aims to guide our research within its broader setting, emphasizing the novelty and significance of our contributions.

2.1 General-purpose Energy Consumption Modeling

To make a stable environment-friendly and low-power design for real-time resource-constrained edge elements, energy consumption should be cut to enable the services to last longer. There are several energy-aware empirical, reasoning, and reconfiguration approaches that study various issues at different code, design, and run-time levels. Some comprehensive empirical studies performed at the multi-stage software life cycle [39] [40] conclude that it is hard to minimize the energy consumption of software products for developers due to the lack of software energy optimization knowledge. They fail as most of them do not have a clear estimation of how large the amount of consumed energy would be for developed applications. Moreover, there are still several incorrect interpretations held even by skilled professionals regarding the principles of energy consumption mitigation [40]. Recent research reveals that the limitations of tools in the green computing area at both the code level and software architecture level may avoid reaching the optimization point [41]. In addition, pinpointing where the energy hotspots are, is a factor that has been used by several experimental strategies to discover the pieces of code affecting the overall energy footprint of a program [42] [43] [44]. The goal is to make the most effective use of energy spent by applications by looking over the code. They aim at identifying the energy profile for various energy-consuming concerns [45]. The authors in [46] and [47] displayed that a slight manipulation in the energy model at the design level would influence energy consumption. Here at this level, having multiple design solutions and comparing the energy consumed by each of them is considered a challenging part while seeking compensation for other quality factors. There are also other applicable techniques addressing the energy-oriented design patterns [42]. The studies in [48] [49] show how model-based energy efficiency analysis affects various consumption characteristics of software architecture. Organizing these models helps us realize the major connection between all those design patterns and the consumed energy in practice. Apart from the code level, method level [50], and architectural level power analysis, other approaches keep track of the energy consumption during run-time [51] or setting up profiles of power usage at each instant of time dynamically [52].

As most of the industrial edge servers feature a heterogeneous architecture for multiprocessor systems-on-chips, it is intended to explain the MPSoC-based energy/thermal models in detail.

2.1.1 Dynamic Energy and Thermal Models in Multi-core Systems

To have long-lifetime battery-powered edge nodes, it is required to replace the energy-intensive execution models with energy-aware frameworks. All the classic research has been involved in energy optimization at the levels of systems, architecture, or circuits so far. The outcome of these energy minimization approaches are categorized in the multiple cases [20]. The first is called Dynamic Power Management (DPM). To avoid wasting energy, DPM temporarily halts any idle element of the system like processing cores. Dynamic voltage-frequency scaling (DVFS) as the second case, permits CPUs to switch between different frequency-voltage levels to scale down the energy consumption. To save a great amount of energy expended on generic tasks on the third class, CPUs are customized to meet the requirements of the tasks on a multiprocessor system-on-chip. The same customization happens for the fourth category but this time for the cache-based memory access. By application task mapping to different processors, the fifth class performs load balancing across all the cores that results in utilization enhancement of processing elements on a multiprocessor system-on-chip.

There is also another factor to consider: over-heat generation, which stems from mismanagement of energy consumption in multiprocessor system-on-chip. This effect is much more prone to happen in multiprocessors rather than single processors as the former suffers from non-homogeneous heat dissipation in its processing elements. Moreover, the non-uniform heat generation leads to the spatial temperature gradients over the whole chip. The more the load on a system for concurrent processing, the higher temporal heat generation/dissipation occurs that may impose the temporal thermal gradients on a small area on the chip. Both the spatial temperature gradients and temporal thermal gradients besides the thermal cycles are three main factors that degrade the performance and the efficiency of a system during its lifetime [17]. Ignoring overheating in edge nodes may also cause a significant loss in performance, durability, and reliability. For example, raising the temperature by ten to twenty degrees in metallic materials would degrade the useful life of the components by sixteen times.

Next, we survey the current techniques for dynamic optimization of energy and thermal behavior on multicore embedded systems. Those techniques are divided into three classes: Dynamic Energy Management (DEM), Dynamic Thermal Management (DTM), and mixture management. For a deep categorization of the energy and thermal optimization techniques, each class is again grouped into two reactive and proactive subclasses. The management

in reactive solutions is applied as a reaction that only responds to a newly emerging state of the system (such as power consumption or temperature) after it has happened. The statistics regarding the state values are often discovered utilizing performance monitoring counters (PMCs). On the other hand, the proactive solutions would continuously anticipate the unseen future states (such as workload and performance metrics) through an estimation model and devise a plan to overcome the negative effects.

Dynamic Energy Management

Considering the constraints in the performance evaluation of multiprocessor system-on-chip, several researches have been conducted to optimize the energy consumption. To optimize resource management, there is lots of work on DVFS together with task mapping that typically makes use of offline, online, or hybrid optimization. Such approaches manage the consumption during runtime through reactive or proactive controlling mechanisms. In the case of reactive methods, a deep design space exploration of hardware and applications is needed for offline optimization. DVFS and task mapping are used in strategies presented in [53] [54] [55]. By introducing a resource model in [53], the authors enhanced the accuracy of the current models by taking into account the energy cost of mode switching in runtime in addition to time. In this approach, by distributing chunks of a given application to the available processing elements, it ensures obtaining the longest system lifetime, while keeping the application performance unchanged. This is called a software partitioning problem that could be defined and solved in the form of an integer linear programming problem [54]. Although some other solutions generate a set of mappings per application, they consume a huge amount of resources for the sake of higher throughput. Such a compromise is not valid in practice since the expending time in this way makes these approaches unreliable during runtime, mainly in the case of concurrent running applications. For taking care of dynamic workloads, authors in [56], and [57] use online optimization to process running applications dynamically. Without the need for application recompilation, performance monitoring counters are hired for an online algorithm in [56] to conserve a remarkable amount of energy. Given the memory resource utilization, a dynamic execution time prediction model and a corresponding DVFS technique have been proposed in [57]. All the online methods listed above, have efficient performance in most of the cases unless for the scenarios in which the decision-making step should be carried out instantly with no prior application knowledge [58]. Tackling this limitation is possible by using a combination of purely online and offline optimization approaches which is called hybrid. The work in [59] uses the hybrid method including a reactive control mechanism. It aims to map threads to specific cores and apply DVFS considering power constraints. Similarly, the thread-to-core mapping in [60] is achieved based upon utilization that

is followed by DVFS based on the surplus power of a 64-core platform. Normally, the heterogeneous platforms are more likely to be in use in industrial machinery as they benefit from a balanced ratio of energy consumption versus the performance [61] [62] [63] [64]. To take full advantage of the parallelism facilities in hardware, most of the above-mentioned solutions prefer to account for multithreaded applications. Although heterogeneous platforms use more than one type of processor core, most works assign the entire application to just one kind of core [61] [63] [64]. After considering all the possible simulated application-to-processing-core mapping by CMP\$im simulator in [61], the one predicted with the highest performance is chosen using a performance impact estimation technique. The mapping problem could be easily solved but there is no more advantage in obtaining a high performance-energy ratio for mapping threads onto non-uniform processing cores at the same time. As an attempt to close this gap, authors in [65] worked on an imitation learning framework that can specify the frequency, number, and type of active cores in non-homogeneous multi-core processors. S. Dey et al. in [66] present a newly developed strategy for managing power consumption during runtime. The next step is followed by transforming the source code of the running application into the intermediate representation code through a Low-Level Virtual Machine (LLVM). Then the outcome turns into an image type constantly to be fed into a convolutional neural network model for data classification. The result would be categorized into three groups: memory-intensive, computation-intensive, and hybrid. According to the resulting classification output and for the sake of runtime power optimization, DVFS is applied to the multi-core processors.

To manage power consumption dynamically, proactive controlling mechanisms have been employed in practice in [67], [68], and [69]. Having high energy-saving performance there, [68] utilizes it to apply online reinforcement learning for the DVFS method in frame-based applications. In [69] a runtime spatial mapping of streaming applications to a heterogeneous multiprocessor system-on-chip is proposed to minimize the energy consumption of the entire application. To handle dynamic application workloads and system energy consumption in [55], a hybrid scenario-based run-time task mapping algorithm is proposed for homogeneous multicore platforms. Authors in [61] use a task mapping approach considering the memory load intensity as the indicator for the task classification. For heterogeneous MP-SoCs, a domain-specific hybrid task mapping algorithm is proposed in [58] that is performed on a system-level simulator. It is strictly dependent on offline analysis though. Both proposed approaches in [58] and [61] are carried out without DVFS implementation and consequently do not take advantage of its energy conservation feature. To aim at saving energy in heterogeneous multicore platforms, an energy-efficient thread assignment technique has been proposed with varying operating frequencies in [70]. Although using the DVFS technique

avoids consuming considerable energy in [71], [68], [62], [63], and [64], there are still large gaps to be closed. For example in [63], the configuration space exploration is fulfilled for only one application. The outcome is then applied to the concurrent execution of applications. This approach is infeasible as the actual configuration space is dramatically larger. The scenario discussed in [64], is not a comprehensive case as the single-threaded applications are chosen that address single-type cores for executing applications. In contrast, heterogeneous cores are investigated for concurrent execution as well as thread mapping on non-similar type cores in [62]. Nevertheless, there is substantial offline/online exploration involved in generating regression models in this approach. The overhead of such costly operations negatively affects the system performance as it is required for all Dynamic Voltage-Frequency Scaling levels in addition to the thread mappings. Moreover, their work lacks the online periodic adjustment for Dynamic Voltage-Frequency Scaling levels, even though it plays a key role in load variation adaptation and energy conservation. Considering that the works in [71], and [67] made an effort to overcome this issue, they still suffer from a major dependency on offline results. Authors in [72] target the dependency problem by relying on an online mapping approach adapted to application arrival/completion times, together with workload/performance variations.

Dynamic Thermal Management

Even though most reactive and proactive thermal management approaches have investigated the general-purpose processor, digital signal processor, or network on chips with dozens or hundreds of cores so far, multicore platforms have not been widely addressed in this domain. However, some analyses are performed in [20], [73], and [74] to improve the dynamic thermal performance in mobile systems. To reduce the energy consumption of gaming applications on heterogeneous multi-processing processors in [73], a proactive approach including DVFS is used in addition to a frame- and thread-based strategy that predicts the gaming workload to apply a smart allocation of threads to cores. Alternatively, reactive approaches typically manage the thermal behavior by defining a certain safe threshold to investigate if the temperature of cores stays in the safe zone or not. In case of violation, an adjustment response intervenes to take the system back into a safe margin. These correction responses cover a variety of actions such as running a cooling system, switching off a module, or scaling down the core frequency. As another example of the reactive mechanisms, a runtime thermal optimization method is proposed in [74]. This temperature monitoring system maps a core frequency to a temperature for the best performance adjustment while keeping the operating temperature under control. By reducing the design space exploration based on a heuristic approach, the authors in [20] delimit the operating frequency that enables them to propose

a runtime thermal management approach with the most applicable frequency to meet the thermal constraint.

2.2 Edge/cloud-based Energy Consumption Modeling

Minimizing energy footprint can be performed at the node and cloud levels. At the node level, depending on the communication scenario, such as the location of nodes and the signal quality, data transmission is the primary cause of energy consumption of an IoT node in wireless sensor networks [75]. At the cloud level, data centers as powerful processing units are the decisive elements in the cloud computing domain. They host several power supply blocks, control systems, data communication, storage, and cooling systems, in addition to security components. Evaluating energy consumption in such complex systems is challenging as it requires a deep investigation of various connected elements [76]. We present the above-mentioned energy-saving approaches at the node and cloud levels in the following sections.

2.2.1 Node-Based Solutions

For IoT sensor nodes, extensive resource exhaustion occurs in transmit (TX) and receive (RX) states [77]. Several solutions are offered to remedy the high energy consumption rate and increase the life expectancy for sensor nodes in an IoT scenario. Without working on reliable energy-saving strategies for IoT nodes, a considerable strain will be put on the energy harvesting sources to provide sufficient supply for operational activities. There are four primary categories for network-based energy-saving approaches on IoT nodes: sleep/wake-up protocols, radio optimization, energy-efficient routing, and data reduction [78]. As we focus on dynamic link quality and transmission settings in this work, the first two approaches are investigated here to represent the role of protocols in the transmission energy consumption of a node.

Radio Optimization

is a set of approaches applied to a radio module as the most energy-hungry element in a sensor network to optimize the required energy for data communication. This type of radio optimization manages the energy consumption regime by manipulating the radio parameters. To this end, several optimization techniques for power transmission, modulation, and antenna direction are suggested. Moreover, a cooperative communication strategy and cognitive radio approach deliver substantial improvement to suppress the appetite for the remaining energy in wireless sensor communication [79]. The cognitive radio, a prominent strategy against

energy over-consumption in retransmissions (caused by packet loss), enables a transceiver to distinguish and avoid the occupied channels and access the vacant channels for data communication. Based on channel states, IoT nodes can intelligently pick a proper channel in the wireless spectrum and accordingly regulate the TX/RX parameters [80]. By comparing the cognitive-radio-enabled sensor networks with the classic ones, researchers [81] indicated a 13% improvement in the network's lifespan. In cooperative communication strategies, a virtual multi-antenna array is achieved by using multiple single-antenna nodes and sharing antennas with them, which results in a remarkable enhancement of the received signal strength (RSS) [82]. Evaluating these strategies shows that a large amount of energy could be saved for contributors in a wireless sensor network. This regime designates a set of nodes as relays and uses them to distribute the communication load by reducing the distance between IoT nodes and the gateway. Hence, the required energy for data collection through the gateway would be cut significantly, and the lifetime of the sensor network would be extended.

Selecting proper antennas affects the energy performance of sensor networks significantly. For example, replacing an omnidirectional with a directional antenna limits the unwanted interference caused by retransmissions [83]. The considerably enhanced energy comes from the fact that if a transmission is done toward a specific direction, the neighbor communications could also be accomplished without interfering with others [84]. Although the advantages of removing omnidirectional antennas have been demonstrated, there are many practical difficulties in direction adjustment and identifying proper parameters for directional antennas. One of the solutions for the radio modulation parameter selection could be the modulation optimization strategy that contributes to energy minimization [85] [86]. By carefully setting the modulation parameters in a wireless sensor network, a balanced compromise between the transmission power and the inter-node distance can be devised to regulate an appropriate circuit/transmission power consumption. In radio communications, the transmission power is managed using the Dynamic/Transmit Power Control (DPC/TPC) mechanism [87]. While maintaining the necessary link quality parameters, such as the Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI), DPC reduces the power of a radio transmitter. By avoiding interference, it results in network life span extension as well [88]. However, applying radio optimization techniques such as adaptive power adjustment at a high rate makes communications vulnerable to packet loss.

Sleep/Wake-Up Protocols

A practical solution for prolonging the life span of event-driven wireless sensor networks is to change the behavioral pattern of IoT nodes for data communication with the edge. The

idea behind behavioral tailoring in event-driven networks is to intermittently let the IoT nodes switch to sleep/awake mode. While an event is scheduled, the nodes would be in the awake state (radio on) just before data communication is initiated, and they would be switched back to sleep mode (radio off) as soon as the data stops flowing (no-event availability). Offering such a mechanism effectively contributes to energy footprint optimization in a resource-constrained edge network. For establishing a connection in this scenario, the sender and receiver should acquire a priori knowledge about the sleep-wake schedule of the neighbors (synchronization step). Given that information, the endpoint switches to the active mode right before the pre-determined moment and is subsequently involved in the data communication. To conduct the synchronization step, a topology control protocol can be used while adjusting the communication duty cycle between individual IoT nodes. The duty cycling technique offers an energy efficiency scheme by enabling nodes to switch between sleep and awake modes at regular intervals [89]. As the amount of consumed energy in awake mode doubles compared to that of sleep mode [90], a tuned duty cycle makes the sleep period enlarged while keeping the idle listening mode remarkably limited [91]. Duty cycling turns the radio on by detecting the preamble of a low-level carrier. Optimizing the energy consumption could be carried on through the topology control techniques (connectivity- and location-oriented) [92] [93], as they minimize the number of essential IoT nodes to meet the network connectivity and coverage requirements. The connectivity-oriented protocols for wireless sensor networks toggle IoT nodes between ON and OFF states dynamically to verify inclusive coverage and guarantee connectivity among nodes. On the other hand, according to the physical location of the IoT nodes, location-oriented protocols decide which nodes are qualified to be put in ON mode and when this transition takes place in a wireless sensor network to ensure network connectivity and coverage. While this approach reduces energy consumption, it suffers from a critical weakness. Real-time data collection cannot be performed using this technique as it requires a high-duty cycle of an IoT node and, subsequently, results in high energy consumption.

2.2.2 Cloud-Based Solutions

Most proposed approaches addressing high energy consumption are grouped into idle server shutdown and workload consolidation. However, machine-learning-based solutions have recently seen an upward trend in energy optimization. Demirci in [94] presented the produced heat, virtual machine placement, task scheduling, and power distribution as the main contributors to power consumption, while the environmental control overhead and processing overload control are considered side effects. However, resource prediction without a priori knowledge of allocated tasks to the data centers usually comes with a large error margin.

Thus, an efficient predictive model can significantly enhance energy consumption by providing an optimized allocation approach. DeJaVu [95] is presented as a cloud-based resource management system. The authors made a classification for load consolidation through a supervised machine-learning approach. As a result, the system started to be trained by analyzing the allocation process for varying workloads. After this step, DeJaVu has already learned to classify any unseen workload in two ways. It checks if the load is similar to any of those previously seen or not. If so, the exact allocation applies. Otherwise, it is treated as a new sample to be learned. As another load forecasting strategy, a neural-network-oriented predictor is employed in [96] to forecast how large the cloud will be loaded in the future. According to that, the unnecessary running machines would be switched off to prolong the total uptime of the system. Such green scheduling solutions can keep the same productivity with fewer servers while consuming less than 60% of energy compared to the standard scenario. Other works [97] anticipate running tasks' future energy footprint and service level agreement metrics like response time using supervised machine learning. The insightful feedback provided by such estimation helps the maintenance team devise energy-efficient task scheduling for high-demanding data centers. Applying unsupervised machine learning (ML) [98] could estimate the required resources by virtual machines. By evaluating the resource estimation, this approach reduces the consumed energy in data centers by turning unnecessary machines into sleep mode. A centralized approach in [99] is used for resource allocation using Q-learning in Heterogeneous cloud radio access networks. Despite a considerable energy budget saving by this framework, their analysis reports that the interference mitigation is met and the quality of service is acceptably preserved among users.

Though the data analytics approaches discussed so far are related only to data centers, leveraging machine learning decision-making to resource-constrained edge devices is also a high-demanding trend. Due to the limited hardware/software capabilities of edge devices for IoT applications, mapping large-scale ML algorithms is not practical. However, an energy-efficient in-memory computing kernel is proposed for linear classification [100]. They use a split-data-aware mechanism to adjust voltage, process, and temperature changes. This results in an admissible balance between energy efficiency and accuracy features. On the other hand, the code size is another critical bottleneck for the resource-constrained edge devices. To reduce the disk occupancy in such edge devices, several techniques like compression and weight pruning are leveraged. As Facebook researchers discuss in [101], there are other developed optimization solutions for code size management in the decision-making process at the edge level. Starting with compiling applications that contain machine learning models to platform-specific object code by making use of TVM [102], and Glow [103] are some of those efforts. Moreover, using vendor-specific APIs from OS suppliers could be another solution

for large code size volume [104]. To meet the requirements of resource-constrained CPUs on edge devices, two libraries are integrated into Caffe2 for computation optimization (especially for multimedia processing) by authors in [101]. The first is NNPACK which is responsible for the 32-bit floating-point computations, while the second one QNNPACK, performs 8-bit fixed-point computations. These works explain well how big companies like Facebook are promoting energy consumption and operating temperature optimization in edge computing. Apart from optimizing the required energy for task processing, remarkable attention should be paid to reducing the tail energy. High tail energy is consumed as the NICs cannot give up working at a high-power state even after sending the packet is finished. Therefore, managing the tail energy of both third- and fourth-generation NICs is of great importance for energy optimization [105] [106]. Bonsai, a resource-aware tree-based ML framework is proposed in [107] for edge devices. By learning a sparse and shallow tree with powerful nodes, Bonsai reduces the model size and cost of prediction while keeping the same level of accuracy metrics. As a result, the performance of prediction is outstanding even on legacy microcontrollers, while the memory occupancy is limited to a few kilobytes and the energy consumption competency is quite high. Although the edge computing domain has been improving fast in recent years, implementing CPU- and GPU-based machine learning algorithms on resource-constrained edge devices puts them under a serious computational load in IoT scenarios. However, FPGA-based edge devices are chosen to be optimized for large neural networks [108]. By using multiple design strategies such as configurable DNN IPs and resource allocation across DNN layers, they succeed in achieving an efficient energy pattern on FPGA. While the processing unit plays a key role in the energy consumption of an edge device, a high transmission rate of NICs has a strong relation with the battery depletion rate of edge devices as well [109] [110]. Minimizing the transmission rate of a NIC employing data compression is also an applicable approach on edge devices. Reducing transmitted data between edge devices and the edge server could also be achieved through data filtering over routed data both at the source and intermediate nodes [111]. As another work, For improving the efficiency of deep learning algorithms such as response time and energy consumption on edge devices, an offloading technique is proposed in [112]. There are also some software accelerators like DeepEar and DeepX for deep learning execution that enable edge devices to notably reduce the edge device resource consumption (e.g. energy) required by low-power deep learning inferences [113]. As another solution for data compression, a combination of a lossless Differential Pulse Code Modulation and lossy temporal compression technique is presented in [114] to obtain a better compression rate and thus a higher energy perseverance rate. Similarly, other data reduction strategies are implemented by authors in [115], and [116] targeting the energy-constrained processing. Following an energy-efficient radio

Table 2.1 Energy saving mechanisms in IoT domain

	Approaches	Methodology	Strategy	Articles
Energy Saving Mechanisms	Node_based	Radio Optimization	Cognitive Radio	[128], [129], [130]
			Cooperative Communication	[128], [82]
			Directional Antenna	[131], [132]
			Modulation Optimization	[85], [86]
		Sleep/Wake-Up Protocols	Transmission Power Control	[87], [88]
			Duty Cycling	[89], [91]
			Topology Control	[133], [93]
	Cloud_based	Node Sleeping	Unsupervised Learning	[134]
			Supervised Learning	[135], [136], [137], [138], [139]

resource management approach in [117], a strategy for bandwidth occupation and scheduling is proposed by manipulating the corresponding computation capacity and channel states. As a result of this technique, the energy usage for edge devices is improved as well as the performance of the learning process. In addition to all the above-mentioned strategies for minimizing the energy consumption of edge devices, considerable attention should be attracted to the routing protocol energy footprint. Intelligent routing is considered one of the most critical elements in the packet loss rate, battery depletion rate, and the performance of the whole edge network [118].

Most introduced energy over-consumption detection approaches use external devices such as smart plugs [119] [120] [121] or are designed for smart buildings rather than computing devices [122] [123] [124]. Some other strategies follow general-purpose models that do not meet the constraints for edge nodes or use dynamic energy and thermal models with high overhead [125] [126] [127]. Edge-based energy modelings are mostly application-specific and may not necessarily target the overall energy consumption of the edge node. In some others, the energy requirements are known in advance based on contextual information such as location, time, and device capabilities. In others, the energy-controlling regime will mainly be deployed unacceptably late. A summary of the discussed energy-saving mechanisms is presented in Table 2.1.

Through our work, we target the limitations of the existing energy anomaly detection models to devise a non-intrusive and affordable mechanism to reduce the energy footprint of an edge network.

2.3 Instrumentation

The instrumentation process is one of the software debugging mechanisms in which a pre-selected monitoring code is added to an application code for performance evaluation purposes. Instrumentation and trace recording for embedded systems could be fulfilled by utilizing static instrumentation to track block code behavior. This is typically performed by changing the source code permanently which implies a remarkable overhead on CPU and RAM usage. For

instance, the Google XRay implementation requires compile-time code insertion to reserve space in the function boundaries [140] [141] [142]. This loads an overhead of 6% on the performance of the system [143]. In static instrumentation, the tracepoint is inserted into the source code, and then the updated source code is saved on disk. Thus, the tracepoint location is known by the run-time. If inserting an instrumentation routine before executing the code is essential, static instrumentation is the preferred choice. It facilitates using a wider range of instrumentation tools and precise probe sites within the code.

Although static instrumentation gives the power to manipulate almost every line of the program, it is not portable across different languages. It requires the availability of the source code for performing instrumentation and does not scale well in case of probing huge bunches of code. However, It provides a reasonably reliable and trouble-free technique to gather all the required information. Still, at the same time, it does not offer much versatility as dynamic binary instrumentation does. To tackle these issues, dynamic binary instrumentation could be exploited eventually.

2.3.1 Dynamic Instrumentation

Dynamic Binary Instrumentation (DBI) is known as a run-time executable modification technique that enables tracing to inspect the behavior of an executable. Through dynamic code injection mechanisms, dynamic instrumentation injects user-supplied analysis routines into the user-defined location(s). Transferring the control from a user-defined spot to the desired location [144] (for monitoring purposes) is done by using a jump or call instruction (*0xe9* or *0xe8*) or a trap (*0xcc*) as a probe. Without a need for recompilation, this approach audits the behavior of a program at any selected analytic point [145]. However, DBI is not bounded just to tracing and debugging. As DBI techniques are flexible and independent of source code or compiler support, they have been considered an important approach for software testing, software security, and temporal code verification [146]. One of the legacy DBI techniques is to overwrite the original instruction(s) with a trap or jump instruction. In the case of a trap, several transitions happen between user and kernel space that make the execution time of a probe 600 to 900 times slower than that of the jump-based technique [147] [148]. Using a jump instruction, the control would be kept in user space that avoids burdening further loads on performance-sensitive production systems. However, if the size of the probed instruction (the original instruction under investigation) is shorter than the size of the probing instruction (jump instruction), there is no choice than overwriting the subsequent neighbor instructions. This will corrupt the execution flow of the program and make the insertion process of a jump-based probe longer or even cause undefined behavior in the system. By

using a legacy jump instruction, keeping concurrency in multi-thread systems would be a cumbersome issue that degrades the performance of the system and the success rate of jump-based probe insertion dramatically. From embedded firmware instrumentation [149] and on-demand scalable testing frameworks [150] to trap-and-emulate hardware VMM of x86 [151], and malicious software detection [152], DBI techniques have been used in practice extensively.

2.4 Tracing

Event tracing as an effective facility between a process and the Linux kernel enables the user to collect the kernel activities or application-defined events out of an already-instrumented application/kernel. Moreover, since edge devices are mostly implemented on multitasking RTOS (Real-Time Operating System), their performance also depends on response time. Timing must not exceed a certain threshold. To meet the deadline, developers could make use of tracing. Tracing as an operative technique collects and discovers the target process execution state and analyzes the performance of elaborate components. A trace of a program includes a sequence of events written by probes. The tracing events could be served from a trace file or even in real-time to debug a kernel/application or to troubleshoot the performance issues that may occur in an application [153]. As system calls are part of tracing events, they can spot abnormal behavior of energy consumption in embedded systems such as edge devices. Detecting anomalies in Liu's works [154] [155] is done by extracting the histogram of different system call types. Using a bag of system calls [155], the authors in [156] investigate intrusion detection utilizing standard ML techniques. Canzanese et al. [157] utilize the number of short streams of system calls in a fixed time window to detect malicious processes. For process monitoring, the authors in [158] apply a deep LSTM-based (Long Short-Term Memory) architecture on a stream of system call count vectors. Selecting the best features for the classification of malware system call streams in [159] is done by constructing a neural network. Tandon et al. [160] [161] [162] use multiple values of multiple system call arguments to improve their rule-based method. The contribution of authors in [161] [163] is considering the name of the system calls in addition to the CPU cycles count, for anomaly detection in time- and event-driven processes using kernel traces. In [164], two neural networks (LSTM) are applied to textual attributes and response time in logs and traces to detect anomalies in cloud services.

However, tracing at the system call level is one of the most challenging approaches in anomaly detection techniques as coping with data collection becomes uncontrollable easily [164], due to the high trace generation rate. The time-delay embedding approach is not able to scale well as tens of thousands of system calls per second are generated just by a single process [165] [155].

Specifically, studies like [166] [155] answer this problem by dividing the chain of the system calls into shorter pieces and then applying feature extraction over a fixed time window.

In general handling effective high-velocity data streams requires real-time data processing techniques adaptable to pattern change such as streaming algorithms [167]. These algorithms are designed for real-time processing as continuous data arrives. Techniques like sliding window [168] [169], approximate query processing (AQP) [170], and online aggregation [171] allow for immediate analysis based on the most recent data. There are also frameworks like Apache Kafka [172], Apache Flink [173], and Apache Storm [174] offering reliable solutions for real-time data processing. These frameworks offer essential low-latency processing, fault tolerance, and scalability for handling real-time traffic management in edge computing environments.

2.5 ML-based Anomaly Detection Approaches

Various unsupervised and supervised ML-based approaches offer different ways to energy anomaly detection. Since each method contributes differently to detection, it is important to choose them based on the specific needs of every single edge network.

2.5.1 Unsupervised Learning

These are known as detection techniques by which the anomalous observations could be discovered as the least fit in an unlabeled dataset while most of the instances are considered normal [175]. By grouping similar data points or estimating density, techniques like k-means, DBSCAN, local outlier factor (LOF), and Isolation Forest can identify outliers. The unsupervised detection techniques face challenges such as generating false positives and difficulties with interpretability. They also struggle to adapt to evolving patterns and changing conditions as they do not take advantage of guidance provided by labeled data.

Clustering

In unlabeled datasets, by splitting the energy consumption data into multiple clusters, data is classified into anomalies and normal ones. It has extensive applications for anomaly detection in different domains like fraud social account detection [176], video surveillance [177], Internet of Things (IoT) [178], intrusion detection [179], transaction-based banking systems [180], and sensor networks [181]. In [182] a two-step anomaly detection strategy in a multi-user energy consumption monitoring system is presented. At first, an anomaly score is given based on the history of the user's consumption. Later, considering the other users' consumption in the

vicinity, the anomaly score would be reevaluated. Data integration of the neighbor users enabled them to split the anomaly consumption from the variation in normal use. In [183], the authors detect the anomalous itemsets of events considering their patterns of appearance. In addition to the frequent itemset mining, this work employs categorical clustering with clustering silhouette thresholding to analyze the data streams. Therefore, as the outcome is achieved through collective-event detection rather than single-event one, anomaly detection can be performed. An integrated scalable framework is proposed in [184] to detect anomalies in events on massive unlabeled data logs by using clustering and classification techniques. Anomaly detection in time-oriented data is used to discover unwelcome incidents. [185] puts forward a framework to detect two different types of anomalies based on their shape and amplitude. By introducing a fuzzy clustering and anomaly score, anomalies are captured in the original representation and autocorrelation representation of the time series. Using mutual K nearest neighbors (MNN) and K-means clustering algorithms, the processing applies to power data to cut the measurement samples in [186].

Advanced Learning Models

Deep learning models are increasingly used for anomaly detection as they can identify and learn complex patterns within data. Auto-Encoders is a type of neural network architecture. The architecture is composed of an encoder and a decoder. First, the encoder compresses the input data into a lower-dimensional latent space and captures the essential features by discarding less critical information. Second, the decoder reconstructs the data from this compressed representation and reproduces the original input to its best. Secondly, we have Ensemble learning models that enhance anomaly detection by combining multiple models to enhance overall performance. Ensemble learning can detect a wider range of anomalies, and reduce false positives/negatives by taking advantage of the strengths of combined models. Bagging, boosting, and stacking techniques are commonly used in these models to ensure more reliable anomaly detection. Wherever maintaining high detection accuracy is required, ensemble learning is effective, especially in the uncertain and dynamic nature of data. Thirdly, Reinforcement learning (RL) models learn in adaptive and dynamic systems to optimize their behavior based on feedback from the environment through trial and error. For anomaly detection purposes, these models can repeatedly learn to adjust thresholds, trigger alerts, or adapt their detection strategies dynamically as conditions change. This behavior fits particularly wherever data distribution and anomalous patterns can vary over time. Such intrinsic features allow RL models to maintain effective detection strategies despite the traffic workload condition.

Note that, in choosing an anomaly detection technique for an edge environment, we should consider the trade-offs between detection accuracy, computational efficiency, and resource constraints. Various techniques should be chosen according not only to scenarios, but also depending on the available resources, the desired level of performance, and the type and volume of data.

2.5.2 Supervised learning

Trained data is required to be categorized and annotated for supervised anomaly detection in energy consumption datasets including the labels for normal and abnormal consumption [187]. Most datasets in practice do not have labels for the collected data consumption. Therefore, supervised learning is not as adopted as unsupervised algorithms in operational projects even though they obtain much higher accuracy metrics. Such techniques comprised three main elements: lowering the dimensional space (Dimension reduction), Developing a predictive model (Model learning), and grouping similar samples (Binary classification). Of these techniques, classification is the most common, and it will be explored further in the following part.

Classification

Given multiple energy consumption, a classification method model identifies the category to which a not-already-seen energy consumption value belongs. The prediction concludes based on the values observed during the training model using normal and abnormal consumption labels. Energy-oriented classification algorithms are divided into two groups: linear classification (e.g. Logistic Regression (LR), Support Vector Machine (SVM)), and non-linear classification (e.g. Decision tree (DT), K-Nearest Neighbors (KNN)). The detected anomalous energy in [188] is classified by comparing the performance of KNN versus other classifiers. Another application of the KNN algorithm in anomaly detection is found in [189] [190] by using KNN-based heuristic algorithms. A mix of a genetic algorithm and SVM in [191] is presented to indicate irregular energy consumption and consumers. Similarly, SVM is proposed in [192] [193] to discover the data anomalies due to energy theft. The work in [194] proposed an adaptive method for anomaly detection of smart meters based on GMM-LDA (Gaussian mixture model linear discriminant analysis) clustering feature learning and PSO-SVM (Particle Swarm Optimization Support Vector Machine). By making use of data generated by sensors, the authors in [195] detect the anomalies in power consumption utilizing a DT regressor. Using the same supervised algorithm, fraud energy consumption is detected in [196]. An upgraded version of DT is proposed in [197] to identify the consumption data anomaly.

lies by comparing the normal/abnormal class densities. Logistic regression is also a popular anomaly detection approach as it is used in [188] in power generation plants.

Most of the introduced energy over-consumption detection approaches make use of external devices such as smart plugs or designed for smart buildings rather than computing devices. Some other approaches follow general-purpose models that do not meet the constraints for edge nodes or use dynamic energy and thermal models with a high amount of overhead. Some edge-based energy modelings are mostly application-specific and may not necessarily target the overall energy consumption of the edge node. In some others, the energy requirements are known in advance based on contextual information such as location, time, and device capabilities. In some others, the energy-controlling regime will mainly not be deployed sufficiently early.

Dealing with the limitations of modern edge network maintenance infrastructure, made researchers and governments pioneer several innovative techniques and novel context-aware green frameworks for IoT-enabled services. Accordingly, this cooperation would lead to a smarter edge networking model besides providing stable environment-friendly, and low-power mechanisms. We intend to discover a network resource over-consuming detection mechanism to diagnose the energy-intensive edge network nodes. Having a DBI framework for data collection purposes enables tracing on energy-intensive devices. It also follows the main steps to conduct a learning-based model for resource-constrained edge networks in an IoT domain. It will be done through practical monitoring of the internal and external behavior of edge elements.

CHAPTER 3 METHODOLOGY

In edge networks like Fig. 3.1, energy consumption evaluation for edge devices is essential due to their limited power resources, typically batteries. The type of tasks these devices perform determines the best approach for detecting and addressing energy over-consumption. There are two primary categories of edge elements based on their predominant activities: communication-intensive devices (SDs) and processing-intensive devices (edge devices). Each category benefits from different data collection strategies to diagnose energy anomalies effectively.

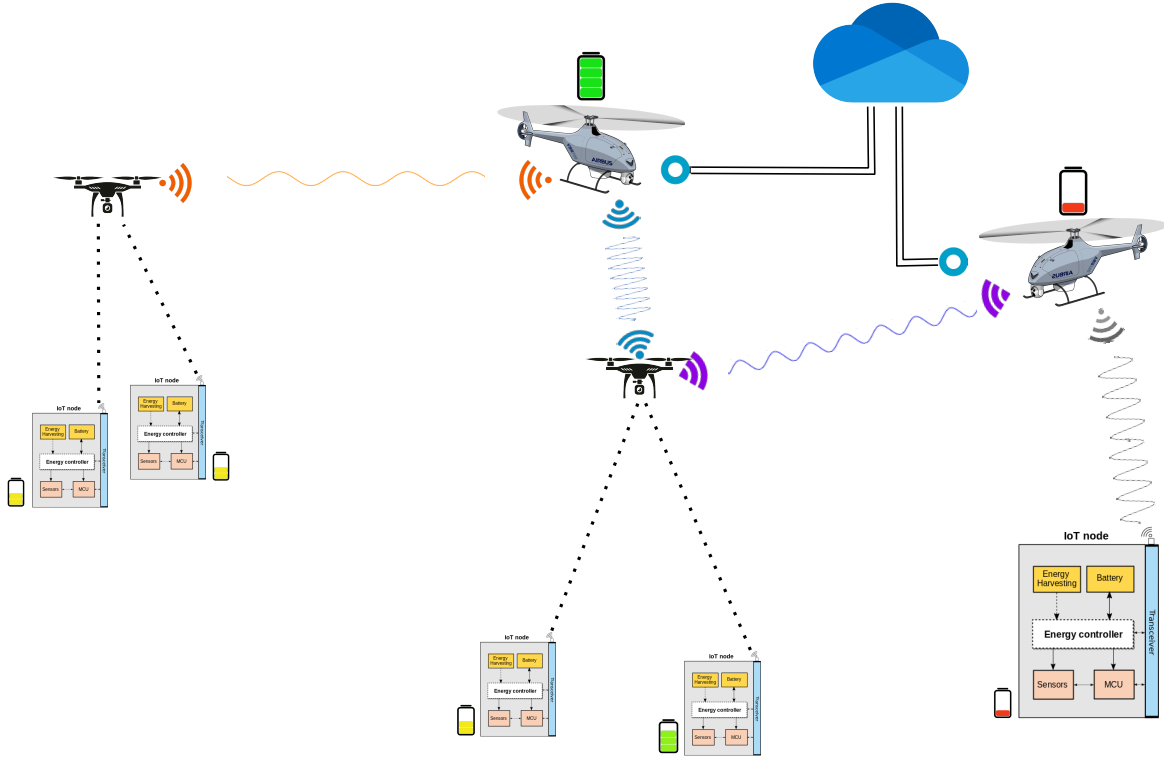


Figure 3.1 Edge computing scenario with communication-intensive IoT nodes and processing-intensive edge devices (helicopters)

Common communication-intensive devices can be found in IoT scenarios, where devices spend most of their time transmitting data to edge servers or other nodes. Therefore, their energy over-consumption often results from prolonged or frequent data communication. The optimal energy over-consumption detection strategy for nodes in this group is performed through

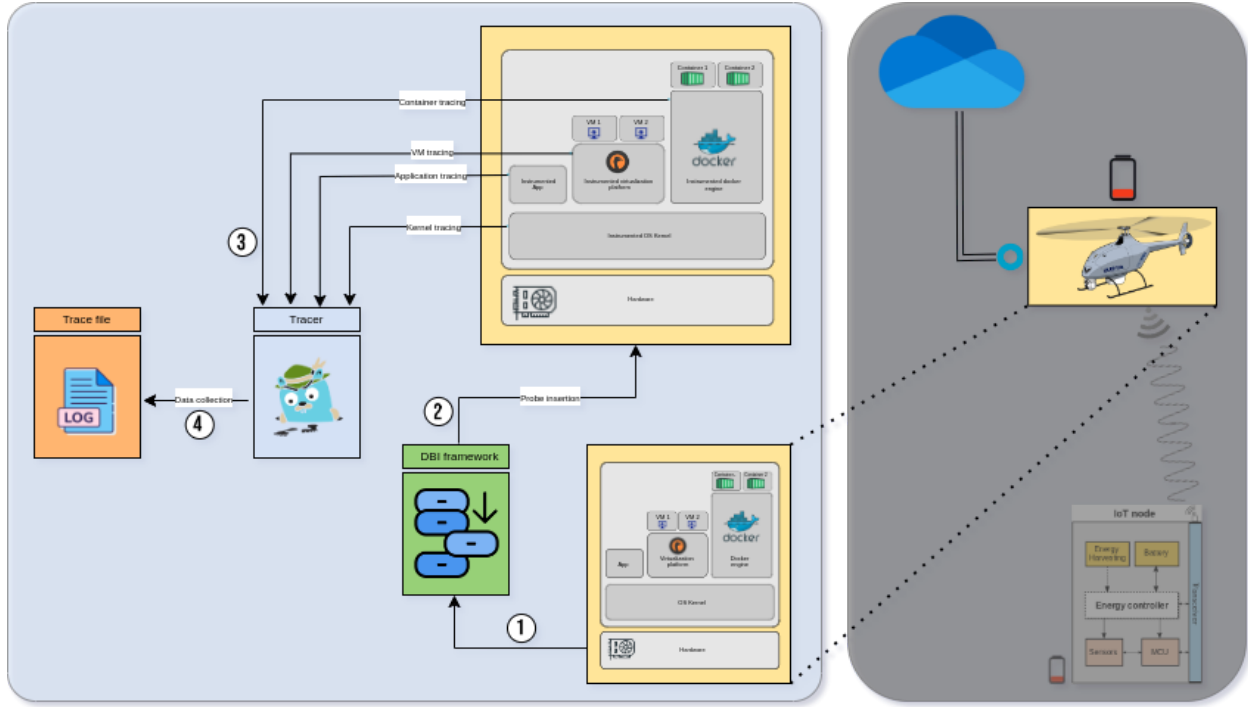


Figure 3.2 Collecting data from an instrumented kernel, applications, virtual machines, and containers

external data collection. To this end, network activity monitoring, connection quality check, signal strength assessment, and communication protocol efficiency verification could be potential methods for energy anomaly detection. On the other hand, processing-intensive devices spend most of their time performing CPU-/GPU-/RAM-/Disk-intensive computational tasks. Edge devices handling data processing, analytics, or machine learning tasks are the main representatives for this category. Their potential source of battery depletion is prolonged and intensive task processing, leading to high resource utilization. Identifying energy hotspots for such devices requires internal data collection strategies like resource utilization monitoring and fine-grained event tracing.

To manage the scenarios mentioned above, we have developed a smart context-aware framework to address the energy challenges that multi-sensor embedded systems face in urban and suburban areas. This framework is designed for on-demand performance analysis and energy management, incorporating various techniques to ensure prompt data collection and energy anomaly detection. Our goal is to enhance the operational longevity of energy-constrained edge networks, making them more responsive to changes in the energy level of the system. As the first step in managing energy over-consumption, we designed a novel data collection framework. Fig 3.2 shows this framework which allows us to systematically gather detailed

data from the kernel of uninstrumented embedded operating systems and/or applications in user space on a device while it operates. It enables us to develop a novel architecture in future steps to detect abnormal energy behavior of devices on the edge.

The lack of built-in instrumentation in kernels of many embedded operating systems, such as FreeRTOS, Contiki, TinyOS, RIOT OS, Zephyr, and ChibiOS/RT, can be attributed to several key factors [198] [199]. Primarily, these operating systems are designed to be lightweight, accommodating the constrained environments in which they operate. Keeping the operating system lightweight and free from unnecessary components helps ensure it runs with fewer bugs. Built-in instrumentation could introduce significant overhead in terms of memory usage and processing power, potentially undermining the system's performance and reliability. For example, FreeRTOS and TinyOS focus on maintaining a small footprint and fast context-switching times, which might be adversely affected by extensive instrumentation.

Additionally, the highly diverse and specific nature of applications in the embedded systems domain necessitates a run-time approach to monitoring and debugging. Built-in instrumentation may not accommodate the varied requirements across different applications. Therefore, it is often more effective for developers to implement custom instrumentation tailored to their specific needs.

NOProbe- the data collection framework proposed for the first objective- allowed us to systematically gather detailed data on a device's performance while it operates. Using NOProbe enables us to develop a novel architecture for evaluating energy in processing-intensive devices, where devices spend most of their time performing CPU-/GPU-/RAM-/Disk-intensive computational tasks. Edge devices handling data processing, analytics, or machine learning tasks are the main representatives for this category. The potential source of battery depletion for devices of this kind is considered as the prolonged and intensive task processing which leads to high resource utilization. Fig. 3.3 pictures the role of the proposed energy management framework proposed in this chapter against other elements on edge. Using NOProbe also allows us to create a novel framework for assessing energy consumption in communication-intensive devices, where devices frequently spend most of their time in data transmission to edge servers or other nodes. Energy over-consumption in communication-intensive devices often results from prolonged or frequent data communication. Fig. 3.4 pictures the role of the proposed energy management framework proposed in this chapter against other elements on the edge.

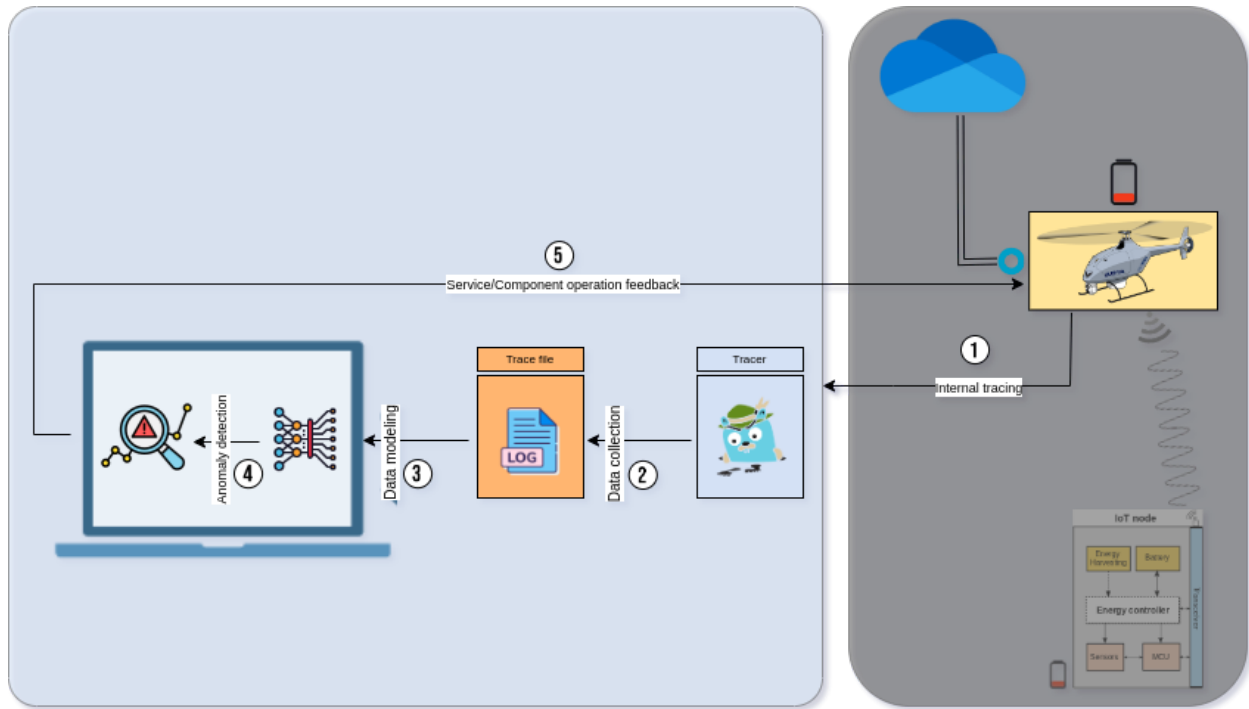


Figure 3.3 A context-aware energy anomaly detection framework using internal data collection

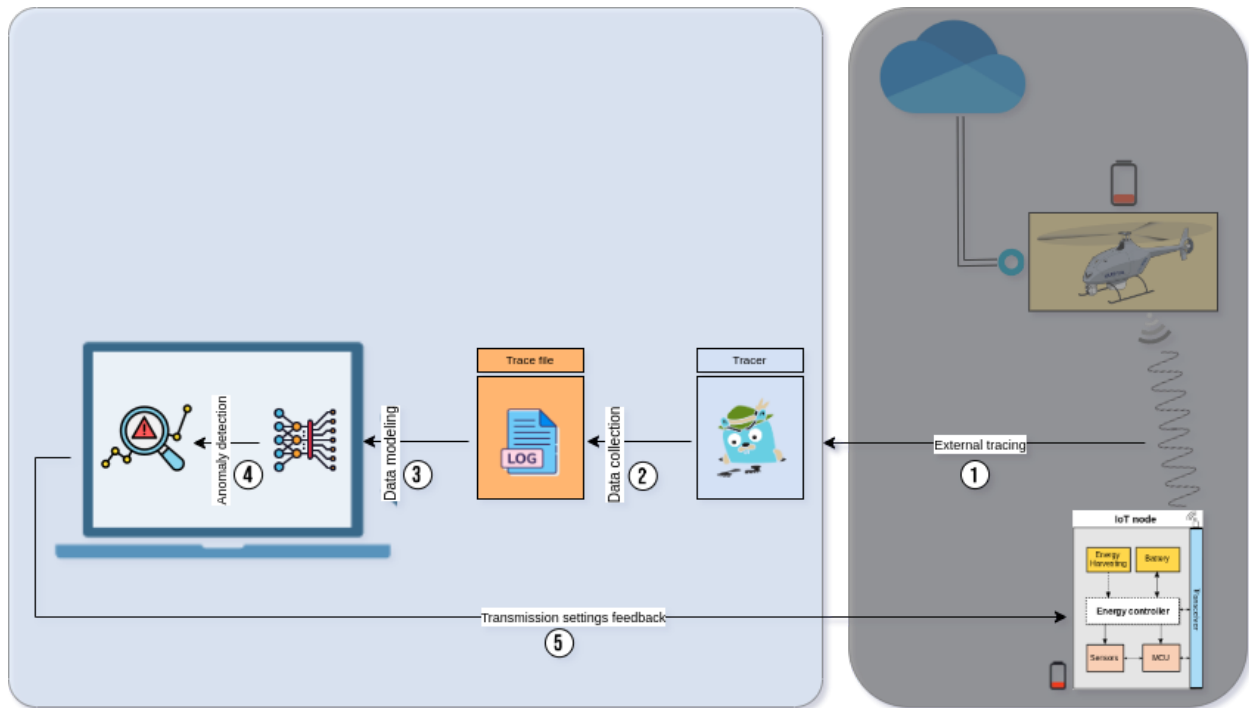


Figure 3.4 A context-aware energy anomaly detection framework using external data collection

CHAPTER 4 ARTICLE 1: "NOPROBE: A NOP-BASED DYNAMIC BINARY INSTRUMENTATION FRAMEWORK FOR EFFICIENT DATA COLLECTION USING BINARY REWRITING"

Authors Ahmad Shahnejat Bushehri, Anas Balboul, Adel Belkhiri, Samira Keivanpour, Gabriela Nicolescu, Michel Dagenais

Submitted on August 21, 2014 in IEEE Transactions on Dependable and Secure Computing

Abstract Tracing is a technique for analyzing the energy footprint of complex multi-threaded systems. It is often used to monitor the execution of energy-intensive operations by inserting probes in running processes. A jump-based probe redirects control flow from the probe’s location to a runtime-generated trampoline to fulfill the tracing purposes. Hence, the success and performance of probe insertion and execution are affected by establishing a proper trampoline placement. Existing techniques are limited in scope as they are costly in probe injection and intrusive in probe execution. We introduce a framework called NOProbe (NOP-based Probe), implementing two new user space instrumentation techniques for the x86 architecture. NOProbe exploits several complementary strategies to achieve a high success rate in the insertion of fast jump-based probes for short instructions (<5 bytes) compared to other user-space approaches that are unable to instrument them. Without compiler assistance, the first method leverages compiler code optimizations for runtime instrumentation. Besides employing a direct stripped-binary rewriting technique without compiler assistance, the second method operates independently of using compiler optimizations. These two methods provide a comprehensive solution adaptable to varied runtime instrumentation requirements.

In the first strategy, we exploit existing paddings– inserted by compilers for code alignment optimization purposes– as strategic placement points for installing trampolines. To achieve this goal, two new algorithms are introduced to assign paddings to probes within a specified range. One algorithm prioritizes performance and follows a greedy approach, while the other focuses on optimizing the global padding assignment. In the second strategy, we utilize pseudo-NOP instructions to explore a wider range of memory space for the trampoline address allocation. We also propose an improved algorithm (*lock-redirect-load-arm*) used by the NOProbe framework to have a thread-safe patching setup. The experimental results demonstrate that our approach achieves a higher success rate for fast probe insertion and maintains a very low execution overhead, outperforming competing techniques.

4.1 Introduction

As more applications are developed for computationally demanding tasks, such as model training, energy-constrained devices face challenges in extending their operational lifetime. In general, energy consumption evaluation techniques are categorized into measurement-based, simulation-based, and model-based strategies [200]. The first two approaches do not comply with the nature of dynamic service execution on remote devices. The first strategy requires physical access to nodes at measurement time while the second one cannot pinpoint runtime transient errors with confidence. On the other hand, the third approach can track application energy consumption during software execution without relying on energy measurement hardware or simulators. Operating at different layers, model-based energy estimation approaches leverage resource utilization (physical hardware), program analysis (user space), and system calls (kernel space).

Utilization-based modeling estimates the energy consumption of a system by considering hardware parameters, such as CPU load, disk read/write accesses, network transmission rate, etc. Discovering the correlation between hardware utilization and energy consumption forms the basis of this approach. Program-analysis-based modeling utilizes code evaluation in user space that performs on the assembly instruction level. As the code instructions are executed sequentially by the processor, the consumed energy by an individual instruction is proportional to the number of cycles it takes to execute [28] [29]. Therefore, the overall energy consumption of a running code is obtained by the collection of the consumed energy for the execution of comprised instructions in the code. In the context of system-call-based modeling, system calls play a key role in representing the resource allocation demand within the system. System calls are the only exterior gate for user space applications to hardware. Consequently, by keeping track of system call invocations made by processes within applications, the energy state changes can be unveiled. Such variations depend on the type of the required resource without having prior knowledge about system workload.

System-call-based and utilization-based modelings take advantage of the tracing mechanism during runtime. They require tracing tools for acquiring hardware counters and system call invocation information. In general, practitioners use tracing tools to discover the target process execution state, collect granular performance-related events, and observe performance problems in run-time. Tracing should not perturbate the system; otherwise, many issues cannot be captured. Imposing negative performance impact, consuming additional resources, introducing concurrency overhead, disturbing timing orders, and creating Heisenbug are some of those issues. Tracing an application in run-time requires binary modification techniques to inject probe points into the executable. Run-time binary modification is already used

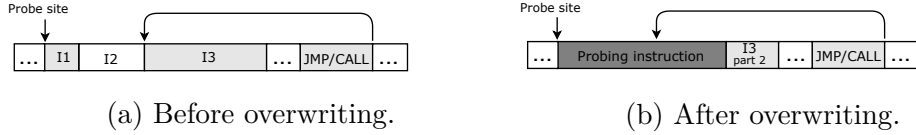


Figure 4.1 Non-thread-safe patching of the short instruction no. 1 with a 5-byte probing instruction

in various areas, such as cache simulators [201], dynamic kernel patching [202], and JIT compilers [203]. Many tracing tools, including DTrace [204], support binary modification for tracing and debugging purposes.

Instrumentation may be performed on the source code statically before the execution, or on the binary dynamically after the execution has started. In most cases, a probe or tracepoint is a fast user-space function call or a slow kernel-based trap inserted into the code. It could be inserted into the source code at compile-time, or into the binary before or after the execution has started.

Dynamic binary instrumentation (DBI) is the process of manipulating an executable code, at the instruction level granularity, to offer some functionalities like tracing. There is no need to recompile the target program as dynamic instrumentation uses dynamic code injection mechanisms. Multiple techniques can be used for DBI to insert probes and user-supplied analysis routines at run-time into the code. As a separate process, the instrumentation code injector program controls the target and provides runtime support while injecting code for tracepoints. The runtime support is handled by an instrumentation agent that is loaded in the same address space as the running target program. The instrumentation agent controls the tracepoints and receives commands from the instrumentation code injector program. For example, the GNU Debugger [30] can inject an in-process agent into a running program to insert low-overhead conditional tracepoints. It is worth noting that DBI as a source code independent technique is used not only for debugging and tracing but also in other domains like software security and testing [31].

The x86 is one of the most popular platforms in production systems. Unlike ARM and MIPS architectures with fixed-length instructions, x86 supports variable-length instructions. Instructions in x86 can take one to several bytes to accommodate a wide range of operations and addressing modes. In most cases, common instructions have shorter and more complex encoding. This may shrink the code size, permit a smaller instruction cache, save clock cycles, and energy [32]. However, DBI tool developers have been struggling with multiple challenges associated with this architecture that add complexity to instrumentation operations by in-

roducing extra memory access and performance bottlenecks. As an example, variable-size instructions are not aligned to either word, cache line, or even page boundaries [33] [34]. This can lead to cache coherence issues, particularly for code cross-modification in multicore environments. To prevent such problems, Intel proposed a cache coherency control mechanism [35].

Another challenge for DBI with variable-length instructions is the length of probing instruction. It may overlap subsequent neighbor instructions if its size (2-byte or 5-byte) exceeds that of the original instruction at the probe site. If a branch lands on the overlapped instruction(s), the processor will try to interpret the bytes in the middle of the probing instruction as a valid instruction. Such branches land in an unintended memory region leading to crashes, security vulnerabilities, or undefined program behavior (Fig. 4.1).

The concurrency in multi-threaded systems compounded with variable-size instructions also adds complexity to DBI in x86 architecture. For instance, when a probing instruction replaces several original instructions, additional measures must be taken for a thread that is running, preempted, or interrupted in the patching area. If the instrumenting agent thread in Fig. 4.1 writes the probing instruction starting at *I1*, while another thread is already interrupted at *I2*, it might cause the latter to see an invalid instruction when it is resumed.

In light of the existing limitations, we propose NOProbe in this paper. NOProbe is a new user-space dynamic instrumentation framework for x86 architecture to insert low-overhead probes almost anywhere within a process. We developed a novel strategy to instrument any multi-byte instruction in optimized codes and in the presence of nearby suitable NOPs. In addition, we propose an original compiler-agnostic strategy targeting the weakness of current jump-based techniques in trampoline placement. Apart from the probe insertion techniques, we also developed a patching algorithm that can atomically patch a basic block¹ in symmetric multi-processing (SMP) environments. Therefore, our instrumentation technique does not resort to a stop-the-world approach and installs the probe safely with limited intrusiveness.

Our main contributions to this paper are:

- We propose a multi-strategy user-space dynamic instrumentation framework called NOProbe. It can insert a fast probe at almost any instruction location in a program built with compiler optimizations (such as -O2) or even without it (stripped binaries).
- We developed an improved user-space mechanism that can safely and concurrently patch a basic block of instructions in multi-threaded programs.

¹A sequence of instructions that has no incoming branches except for the first instruction and no outgoing branches except for the last instruction

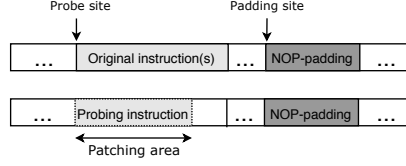


Figure 4.2 General definitions for patching a long instruction before (top figure) and after (bottom figure) the operation

- We implemented our solution in a user-space tracing tool (*uftrace*), developing it to dynamically attach to a running process and inject instrumentation.
- We analyzed and compared the probe performance of multiple existing instrumentation techniques.

The rest of the paper is organized as follows. Section 4.2 surveys existing approaches and explains the background concepts required for understanding dynamic binary instrumentation. Section 4.3 presents a short review of existing instrumentation tools and their limitations. In Section 4.4, we detail the design and architecture for the proposed NOPProbe framework. Section 4.5 presents our experimental results comparing NOPProbe with existing probing tools. Finally, Section 4.6 concludes the paper and suggests improvements for future investigations.

4.2 Background

The effectiveness of dynamic instrumentation in x86 binaries significantly depends on the chosen instrumentation approach. It can vary from kernel-based solutions to user-space strategies. Either way, code rewriting in this architecture must navigate safe cross-thread modification and code insertion challenges. To address the first, the literature proposes serialization mechanisms to enforce a strict order of execution for instructions, synchronize the processor’s internal state, and facilitate thread synchronization in multi-threaded environments. Addressing the latter involves multiple approaches, from intrusively stopping the entire threads to preloading customized libraries to adopting an efficient atomic modification approach.

We discuss the above-mentioned approaches to tackle various challenges related to concurrency, synchronization, integrity, consistency, and coherency when modifying code or data structures dynamically. In this context, NOPProbe leverages advanced binary rewriting techniques by considering the compiler optimizations and semantic NOPs to navigate these challenges and provide a reliable framework. The definitions for the concepts used throughout

the article are listed below and illustrated in Fig. 4.2:

- Probe site: The location at which a probe should be inserted.
- Original instruction(s): One or more instructions at the probe site that are replaced with the probing instruction.
- Probing instruction: The relative *jmp* or *call* branching instructions, replacing the original instruction(s) at the probe site, to transfer the control flow to a trampoline.
- Padding site: The location in which one or more NOP instructions would be replaced by a branching instruction to reach a probe handler.
- Patching area: The region where the original instruction(s) will be replaced, partly or completely, by the probing instruction.

4.2.1 Instrumentation Approaches

To insert a dynamic tracepoint at arbitrary instruction locations on x86 platforms, the following three approaches are commonly used:

- Trap-based approach
- Branch-based approach
- Tranch-based approach (Trap-Branch)

The trap-based approach (used in Uprobe and unoptimized Kprobe [205]) leverages a trap as a probing instruction. As you can see in Fig. 4.3a, the first byte of instruction *I1* has been substituted with one byte *INT* instruction, commonly referred to as *int3*, as a probing instruction. The hexadecimal opcode for this trap instruction is *0xcc*. Executing a trap instruction generates a *SIGTRAP* signal. This approach also allows for the use of illegal instructions instead of traps, resulting in the generation of a *SIGILL* signal during execution [143]. However, in both cases, the software interrupts are processed by the operating system in kernel space, then the user-space signal handler is called, and the control returns to kernel space. After that, execution resumes in user space to skip the trap, execute the overwritten instruction(s) out-of-line, and continue after the trap. As discussed, the execution of a trap or illegal instruction causes several transitions between user space and kernel space. This incurs a significant overhead which may be 600 to 900 times higher than that of the branch-based approach on x86 platforms [30] [206].

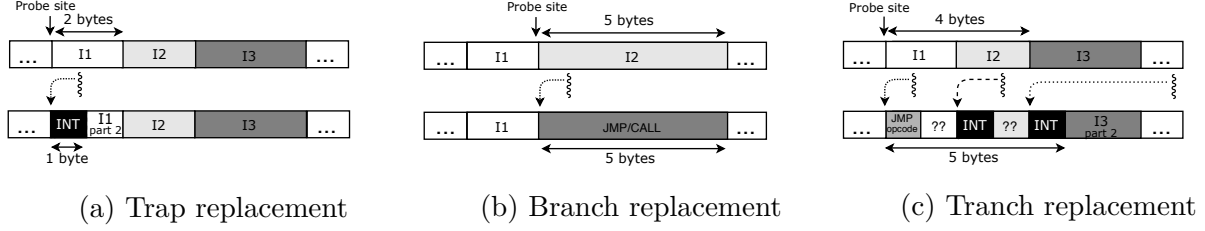


Figure 4.3 Original instruction(s) replacement by a probing instruction in Trap-based, Branch-based, and Tranch-based approaches

The branch-based approach is a collection of techniques for dynamically instrumenting binaries in user space without the overhead of exploiting traps. These techniques typically rely on a relative 5-byte *jmp* or *call* instruction (used in GDB fast tracepoint). Therefore, this approach is unable to instrument an instruction if its size is less than 5 bytes, which is the case for 75% of x86 instructions [207] [30]. The branch in Fig. 4.3b could be a relative 5-byte *jmp* or *call*, while *0xe9* and *0xe8* are the opcodes for these instructions. Both instructions include a 4-byte signed offset to specify the jump or call target relative to the next instruction. This requires the branch target (trampoline) to be within a range of a maximum of 2 GB away from the probe site. Because of these constraints on the target instruction size and the trampoline address range, the insertion of a branch is much more difficult than that of a 1-byte trap/illegal instruction.

When the target instruction is shorter than 5 bytes, the tranch-based approach (used in LiteInst and Dyntrace) can also potentially fulfill instrumentation in user space [208] [143] [209]. It overwrites one or more neighbor instructions (the entire *I2* and the head byte of *I3*) with a 5-byte branch, as depicted in Fig. 4.3c. After overwriting, if one or more overlapped neighbor instructions are the target of other branch instructions, the branching threads would find an invalid sequence of bytes there. As a solution, the tranch-based approach selects the relative offset of the 5-byte *jmp/call* to contain a trap/illegal instruction at the branch destination(s) (head byte of neighbor instructions). Compared to the trap-based approach, the tranch-based one is much more efficient when it is very unlikely that a branch would target such a trap embedded in the relative offset repeatedly. For an instruction that straddled a hot path like the beginning of a loop, the embedded trap significantly contributes to a downfall in performance within the tranch-based approach. To address this issue, employing a trap-based approach is recommended. Specifically, a trap-based probe can be inserted at the first byte of the target instruction immediately preceding the loop.

As explained, the tranch-based approach suffers from limitations in multiple scenarios, such

```

000000000005c3e0 <pevent_event_info>:
5c3e0: 55                push %rbp
5c3e1: 48 89 e5          mov %rsp, %rbp
5c3e4: 48 83 ec 10       sub 0x10, %rsp
...

```

(a) Typical function prologue as produced by the GCC and llvm

```

000000000005c3e0 <pevent_event_info>:
5c3e0: e8                call 0x5c3e0
5c3e1: cc ?? ??          INT3
5c3e4: cc 83 ec 10       INT3
...

```

(b) Tranch-based instrumentation on function prologue results in binary rewriting starting at the address *0x5c3e0*

Figure 4.4 Function prologue layout before and after the instrumentation

as function and loop entry instrumentation. Using the *0xcc* byte code of *int3* at the head byte of an overlapping instruction placed on the last byte of a *jmp*'s offset is also challenging. Generally, a memory type in x86 architecture is represented as little-endian. The most significant byte (MSB) in a little-endian system is stored at the largest address. Fig. 4.4a and 4.5a represent the typical x86 function prologue and for-loop entry layouts. During patching, the head bytes of the second and third instructions are overwritten with *0xcc* bytecode (Fig. 4.4b and Fig. 4.5b). The rest of the offset bytes are represented by double question marks ("??") in these figures. They would be chosen such that they make a valid relative displacement to a mappable memory (trampoline's address). The higher significance the bytes have, the more determinant they are in accessing distant addresses in the vicinity of the patching area. By setting the most significant byte of the offset with a negative value of *0xcc* (for *0x5c3e4* in Fig. 4.4b and *0x1180* in Fig. 4.5b), the range of the offset to reach a mappable available memory for trampoline placement gets limited to an extremely restricted subset of preceding addresses.

Overall, fixing the MSB by embedding traps or illegal instructions in the relative offset of a *call* or *jmp* reduces the opportunity for reaching the available space in the process memory. Hence, the tranch-based approach significantly impedes the extensive memory exploration required for appropriate trampoline placement.

```

...
1150: 85 c0                test    eax, eax
1152: 7e 12                jle     1192 <main+0x42>
1154: 8d 48 ff             lea     ecx, [rax-0x1]
...

```

(a) Typical for-loop entry as produced by the GCC and llvm

```

...
1150: e8 ??                call    <trampoline>
1152: cc ??                INT3
1154: cc 48 ff             INT3
...

```

(b) Tranch-based instrumentation on a for-loop entry results in binary rewriting starting at the address *0x1150*

Figure 4.5 Loop entry layout before and after the instrumentation

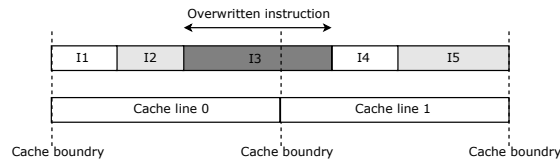


Figure 4.6 Cache straddling

4.2.2 Cross-modification Challenges

Run-time cross-thread code modification in a multi-thread symmetric multiprocessing environment on the Intel x86 platform is complex, poorly documented, and may cause general protection faults (GPF). The instrumentation thread running on processor core *X* could modify some code, while the same code is concurrently executed by another thread running on processor core *Y* [35]. The first concern is cache straddling when the size of the probing instruction is more than 1 byte [143]. There is no problem with atomically inserting trap/illegal instructions since they occupy a single byte. Longer instructions may sit across cache-line boundaries and cannot be written atomically [210] (see Fig. 4.6). Consequently, the thread running on processor core *Y* might find an invalid partially updated instruction on either side of the boundary.

Further problems lie in the coherence of the instruction prefetch cache. If the thread on

processor core Y prefetches the unmodified copy of the code, the micro-operations will be generated accordingly into the trace cache [211]. Since caching and compiler optimizations (like reordering and fusion) may affect the structure of micro-operations, no correlation would be noticed anymore between the original code and the optimized micro-operations. Therefore, the processor no longer keeps track of the corresponding start and end of prefetched instructions in the trace cache. At this point, if the CPU invalidates the trace cache because of an unsynchronized cross-modification, the instruction execution will be terminated with a GPF exception.

According to Intel’s erratum recommendation [35], issuing a *cpuid* instruction on each processor is required to prevent an unsynchronized cross-modification state. It ensures that all modifications made to memory regions, registers, and flags are fully processed before executing subsequent instructions [212]. Although invoking *cpuid* using Intel’s protocols could solve the problem, it remains unsatisfactory. Indeed, the protocol not only demands significant space at the probe site but also introduces a considerable latency through its busy waiting mechanism. This latency is particularly intolerable for NOProbe due to its performance constraints.

4.2.3 Approaches for Safe Code Insertion

Modifying a program at run-time for instrumentation purposes must be achieved without disrupting its normal execution flow. As explained previously, dynamically modifying machine code on modern multicore platforms can lead to many incoherent states. Such modification affects the instruction code, instruction cache, and instruction prefetch cache, potentially resulting in the generation of a GPF. In this context, several techniques were proposed for safe cross-thread code modification:

1. Stop-the-world approach:

By stopping all the process’s threads during dynamic instrumentation, this approach greatly simplifies the problem. It ensures that any concurrent thread executing the modified code section does not enter the patching area. However, stopping all the threads within a process for instrumentation purposes is highly intrusive and often impractical, especially in performance-sensitive cases.

2. Preloading approach:

This approach seeks quiescence by pre-loading an additional library into the target program’s address space using the *LD_PRELOAD* environment variable. It instruments the program before the execution of the target binary. As a limitation, the tracepoints

inserted before the start of the execution cannot be removed once the execution has started. This approach is more straightforward but imposes more constraints than the Stop-the-world approach. This approach does not allow tracing a long-running process (in production) that starts to misbehave.

3. Atomic modification approach:

The patching area of the target process could be atomically modified in this approach. Unlike the other invasive approaches, the probe insertion could be performed during run time without stopping all the running threads in the target program. By writing atomically a probing instruction over the probe site, the control flow transfers to the probe handler. Special care would still be needed not only for the patching of instructions shorter than 5 bytes but also to ensure the safety of cross-thread code modification as discussed in 4.2.2.

4.2.4 Compiler Optimization and Semantic NOPs

Compilers serve the essential function of transforming source codes into executable machine codes. These compilers, such as GCC [213] and LLVM [214], are equipped with several pre-defined optimization levels, each activated by certain optimization flags. These optimization levels aim to enhance specific functionality during run-time [215] [216]. Enabling each flag hinges on specific needs and constraints. Exploiting these flags depends on the software, performance sensitivity, features of the target hardware, etc. In general, optimization is achieved by modifying the number of involved registers, their usage in the code, and their references. For instance, there are several optimization levels available in the GCC compiler to generate customized binaries. The main options are sorted as follows: *-O0*, *-O*, *-O1*, *-O2*, *-O3*. In each level, all optimizations from the preceding levels are combined with additional optimizations. The default optimization flag is *-O0* where most optimizations are disabled. In *-O* and *-O1*, the same optimization applies to a code, resulting in significantly smaller and fast-running code. The *-O2* flag switches on those optimizations that do not require a trade-off between space and speed during compilation. Compared to the previous level, this flag improves the performance of the binary code, while increasing compilation time. The flag '*-O2*' is also the default level of optimization in the release version of software [217]. The resulting optimized code provides a good balance between performance optimization, portability, and reliability of the software's behavior [218]. By adding some additional optimizations (e.g. function inlining) to those specified in *-O2*, the next level (*-O3*) results in larger code sizes while lowering execution time [219].

Padding is one of the most frequently used optimizations that modern compilers can make

Table 4.1 Optimization-level-activated flags at -O2 and -O3 for code alignment in GCC

Flag	Description
<i>-falign-functions</i>	Aligns the start of functions to the next power-of-two.
<i>-falign-labels</i>	Aligns all branch targets to a power-of-two boundary.
<i>-falign-loops</i>	Aligns loops to a power-of-two boundary.
<i>-falign-jumps</i>	Aligns branch targets to a power-of-two boundary, for branch targets that can only be reached by jumping.

Table 4.2 NOPs, 1-byte, and 2-byte pseudo-NOPs in x86 architecture [1]

NOP Instructions		1-byte Pseudo-NOP Instructions		2-byte Pseudo-NOP Instructions							
Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)	Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)	Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)	Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)	Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)	Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)
nop	90	nop	90	rex.W rex.W	48, 48	push ecx, pop ecx	51, 59	PUSHF, POPF	9c, 9d	mov ebx, ebx	89, db
xchg eax, eax	66 90	es	26	xchg %eax, %eax	66 90	dec eax, inc eax	48, 40	pop edx, push edx	5a, 52	xchg ah, ah	86, 04
nop DWORD PTR [eax]	0f 1f 00	cs	2c	invid	0f, 08	rex.W nop	48, 90	push edx, pop edx	52, 5a	xchg bh, bh	86, 0f
nop DWORD PTR [eax+0x0]	0f 1f 40 00	ss	36	ehard	0f, 09	rex.W fwait	48, 9b	pop esi, push esi	5e, 56	xchg ch, ch	86, ed
nop DWORD PTR [eax+eax*1+0x0]	0f 1f 44 00 00	ds	3e	es rex.W	26, 48	rex.W int3	48, ce	push esi, pop esi	56, 5e	xchg dh, dh	86, ff
nop WORD PTR [eax+eax*1+0x0]	66 0f 1f 44 00 00	rex.W	48	es es	26, 26	inc eax, dec eax	40, 48	pop edi, push edi	5f, 57	xchg al, al	86, c0
nop DWORD PTR [eax+0x0]	0f 1f 80 00 00 00	fs	64	es nop	26, 90	dec ebx, inc ebx	4b, 43	push edi, pop edi	57, 5f	xchg bl, bl	86, db
nop DWORD PTR [eax+eax*1+0x0]	0f 1f 84 00 00 00 00	gs	65	cs wait	26, 9b	inc ebx, dec ebx	43, 4b	pop ebp, push ebp	5d, 55	xchg cl, cl	86, c9
nop WORD PTR [eax+eax*1+0x0]	66 0f 1f 84 00 00 00 00	data16	66	cs int3	26, ce	dec ecx, inc ecx	49, 41	push ebp, pop ebp	55, 5d	xchg dl, dl	86, d2
nop WORD PTR [eax+eax*1+0x0]	66 2e 0f 1f 84 00 00 00 00	addr16	67	cs nop	2e, 90	inc ecx, dec ecx	41, 49	pop esp, push esp	5c, 54	xchg ebx, ebx	87, db
		lock	0f	cs nop	2e, 90	dec edx, inc edx	4a, 42	push esp, pop esp	54, 5c	xchg ecx, ecx	87, c9
		wait	90	cs rex.W	2e, 48	inc edx, dec edx	42, 4a	POPA, PUSHA	61, 60	xchg edx, edx	87, d2
				es es	2e, 26	dec esi, inc esi	4e, 46	PUSHA, POPA	60, 61	xchg esi, esi	87, ff
				cs fwait	2e, 9b	inc esi, dec esi	46, 4e	fs nop	64, 90	xchg edi, edi	87, ff
				cs int3	2e, ce	dec edi, inc edi	4f, 4f	cs rex.W	64, 48	xchg esp, esp	87, c1
				ss rex.W	36, 48	inc edi, dec edi	47, 4f	es es	64, 26	xchg ebp, ebp	87, ed
				ss nop	36, 90	dec ebp, inc ebp	4d, 45	cs fwait	64, 9b	mov al, al	88, c0
				ss es	36, 26	inc ebp, dec ebp	45, 4d	cs int3	64, ce	mov bl, bl	88, db
				ss fwait	36, 9b	dec esp, inc esp	4c, 44	gs nop	65, 90	jmp the2	eb, 00
				ss int3	36, ce	inc esp, dec esp	44, 4c	gs rex.W	65, 48	xchg eax, ecx	89, 91
				ds rex.W	3e, 48	pop eax, push eax	58, 50	gs es	65, 26	xchg eax, edx	89, 9d
				ds nop	3e, 90	push eax, pop eax	50, 58	gs fwait	65, 9b	mov bh, bh	88, ff
				ds es	3e, 26	pop ebx, push ebx	5b, 53	gs int3	65, ce	xchg eax, esp	89, 94
				ds fwait	3e, 9b	push ebx, pop ebx	53, 5b	nop	67, 90	xchg eax, ebp	89, 95
				ds int3	3e, ce	pop ecx, push ecx	59, 51	POPF, PUSHF	9d, 9c	mov eax, eax	89, c0
				rep nop (pause)	f3, 90	repnz nop	f2, 90	lock nop	0f, 0f	xchg eax, edi	89, 97
				nop esp	0f, 1f	std, cld	fb, fc	cld, cld	fb, fb		

to generate production code. Such explicit embedded binary optimizations are turned on at *O2* and higher levels. The need for compiler padding comes with heavily optimized modern processors. These processors generally fetch a whole instruction cache line at a time, ranging in size between 16 and 64 bytes. In this context, it is important to align frequently executed code and basic blocks (e.g., the start of a loop or a function) on cache line boundaries. If an instruction straddles a cache line, the CPU would fetch two cache lines. It starts with the one containing the start of the instruction and then fetches the cache line containing the end of the instruction. Hence, code alignment techniques enhance prefetching and cache effectiveness, leading to improved performance. Compilers exploit this mechanism by inserting padding to align instructions on the next cache line boundary at the cost of increasing the binary size. Padding is typically added to four critical regions: functions start, loops, labels, and jumps. Table 4.1 lists the corresponding optimization-level-activated flags enabled in GCC. Apart from code alignment purposes, data structure alignment [220] [221], loop unrolling and function inlining [222] also require compiler padding at *O2* and *O3* optimization levels.

No-Operation Instructions (NOPs) The padding may consist of a single or multiple NOP instructions to fill the gap. The first column of Table 4.2 lists the various types of NOP-paddings and the corresponding assembly instructions. The NOP-padding byte sequences consist of a set of instructions that perform no operation. These instructions have absolutely no effect on the content of their register operands or flags. They only serve to increment the

```

0000000000015610 <do_replay>:
    ...
15763:  0f 88 d7 00 00 00    js     15840 <do_replay+0x230>
    ...
15833:  89 30 ff ff ff      jns    15769 <do_replay+0x159>
15839:  1f 80 00 00 00 00    nopl   0x0(%rax)
15840:  83 c4 01             add    $0x1,%r12d

```

(a) NOP-padding aligning branch target (at address *0x15840*) to 16 bytes

```

0000000000027a00 <free_parsed_cmdline>:
    ...
27a09:  48 8b 7f f8          mov     -0x8(%rdi),%rdi
27a20:  f3 c3               repz   retq
27a22:  0f 1f 40 00          nopl   0x0(%rax)
27a26:  66 2e 0f 1f 84      nopw   %cs:0x0(%rax,%rax,1)
      00 00 00 00 00
0000000000027a30 <absolute_dirname>:
    ...

```

(b) Two NOP-paddings aligning *absolute_dirname* function to 16 bytes

Figure 4.7 Code alignment using NOPs

program counter (*PC*) to the next instruction. The first entry, encoded as a single-byte *0x90*, is the most commonly used NOP-padding instruction in the x86 instruction set. Although *0x90* is identified as a NOP instruction, the same binary opcode also corresponds to *xchg %rax, %rax* instruction (exchange *rax* with itself) in the x86 architecture. The second entry indicates another *xchg* instruction using a redundant prefix (*0x66*) that does not affect the no-operation behavior. For the 9-byte NOP-padding in this table, the *0x66* takes the first byte as a redundant prefix to override the size of the operand. The set of neighboring two bytes, *0x0F 0x1F*, is the instruction opcode. The fourth and fifth bytes, *0x84* and *0x00*, represent the operands format specification *Mod R/M* and *SIB*, respectively. Finally, the last four bytes, *0x00000000* are used to hold the displacement for the memory operand.

The NOPs in a binary code may or may not be reachable depending on the execution flow. Reachable NOP-paddings could be executed by the program at some point. In Fig. 4.7a, if the *jns* (jump if not signed) condition is not satisfied, the branch will not be taken and the *nopl* instruction will be executed. In Fig. 4.7b, two unreachable NOP-paddings are located at the end of the function, immediately after the return point (*repz retq*). While unreachable NOP-paddings can be modified safely, reachable NOP-paddings should be modified carefully due to potential concurrent execution issues. For some NOPs, like the 9-byte one in Table 4.2, the no-operation behavior will remain unaffected, even if the *SIB* field and the 4-byte displacement are modified. This allows embedding data or code into the NOPs without impacting the execution flow [223]. Authors in [224] employ a similar obfuscation technique to hide code by embedding several bytes within specific instructions. They mainly focus on using 9-byte no-operation instruction in their work.

Pseudo No-Operation Instructions (Pseudo-NOPs) They are defined as a set of instructions that collectively imitate the behavior of the NOP instructions. While they are not explicitly NOPs, they maintain the overall integrity of the execution’s control flow [225] [226]. These pseudo-NOP instructions may alter the control flow, register values, and memory content temporarily but execute counterpart instructions to reverse the initial changes semantically. Therefore, pseudo-NOPs preserve the original values for registers and memories for the future use of subsequent instructions. Although variable-length pseudo-NOP instructions are available, the 1-byte and 2-byte semantic NOP are the minimal form of using them. The second column in Table 4.2 includes 1-byte pseudo-no-operation instructions, such as register extension, segment override, operand-size override, and address-size override prefixes. 2-byte semantic NOPs (the third column of Table 4.2) begin with an invertible instruction which is then followed by its inverse instruction. For example, the double-byte sequence of *push reg*, *pop reg* or *pushf*, *popf* (*0x9c*, *0x9d*) or *pusha*, *popa* (*0x60*, *0x61*) instructions, write values to

the stack and then restore them from it [227] [228]. Another set of invertible double-byte instructions is created by incrementing the value of a register and decrementing it immediately afterward (*inc reg, dec reg*). The 2-byte instructions like *mov reg, reg* and *xchg reg, reg* act as pseudo-NOPs by replacing the content of a register with itself. However, the latter option executes less efficiently. Other 2-byte pseudo-NOP instructions are listed in Table 4.2.

Pseudo-NOPs find applications in extensive domains, especially in code obfuscation [225] and malware detection [229], where precise control over code execution, timing, or architectural behavior is required. For example, malware strategically inserts semantic NOPs into critical areas within binaries as part of code obfuscation tactics aimed at evading detection [230]. Inserting a pseudo-NOP instruction between two bytes simply advances the *PC*, and creates intentional space within the code.

4.2.5 Out-of-Line Execution (OLX) Buffer

The original instructions, overwritten by the probing instruction, must be executed eventually to maintain the integrity of the execution's control flow. For position-independent instructions, DBI techniques save a copy of the original code in an OLX buffer. Then, after the instructions in the OLX buffer have been executed, a *jmp* or a trap at the end of the buffer returns the control to the first valid instruction after the patching area. As explained, instructions that do not depend on the *PC* can be copied to the OLX buffer without modification. However, position-dependent instructions should be treated differently.

In the 32-bit x86 platform (x86_32), position-dependent instructions use *EIP* implicitly, without specifying it as an operand. For instance, a jump condition code (*jcc*) instruction *jae 6* would jump to *PC+6* if the condition has been met. With *RIP*-relative addressing in the 64-bit x86 platform (x86_64), instructions can explicitly use [*RIP* + *displacement*] as an address. For example, instruction *mov DWORD PTR [rip+0x1], 0x2* will store the immediate value *0x2* in a double word beginning one byte after the end of the *mov* instruction. The effective address of the memory operand in these instructions is established based on the current value of the instruction pointer (*IP*). Therefore, the copied position-dependent code in the OLX buffer would not be executed correctly without relocation. It rectifies their relative addresses, ensuring they are executed with the same effect as the original instructions. Moreover, if an instruction is relocated far from its intended target in memory, we might not be able to access it using the same addressing mode. Those instructions need to be manipulated to be able to reach their target address. Employing an alternative addressing mode that offers a larger branch offset could resolve this limitation.

4.3 Related Work

Several strategies are discussed in the literature to achieve dynamic binary instrumentation. Some approaches use compiler support, leveraging the compiler’s code transformation capabilities to inject instrumentation hooks at strategic locations within the binary code. Alternatively, other methods apply instrumentation directly to compiled binaries without compiler assistance. The latter category often requires post-compilation techniques to perform DBI. In this section, several instrumentation tools exploiting compiler-assisted and -agnostic techniques are briefly discussed.

4.3.1 Instrumentation Tools

The branch-based approach restricts the size of instrumentable instructions, requiring them to be 5 bytes or larger. To relax this constraint and minimize the run-time-instrumentation overhead, compiler-assisted techniques are used through manually-activated flags. These flags are specified to provide instructions on how to compile the source code and what additional features or modifications are required. Manually-activated flags are often defined as command-line arguments when source files are about to be compiled. Table 4.3 lists compiler flags/attributes that add sufficiently large space at intended probe sites at compile time. The space is patched at run-time by inserting a 5-byte branch instruction.

The Linux kernel Ftrace [231] and its user-space version, Uftrace [232] make use of a compiler-assisted technique. It is enabled by using manually-activated *-mnop-mcount* (the GCC flag in Table 4.3) to insert a 5-byte NOP instruction (*nopl 0x0(%rax,%rax,1)*) at the entry of each function [233]. At runtime, Ftrace locks the probe site with a trap instruction before synchronizing the cores via a low overhead inter-processor interrupt. In user space, Uftrace patches the NOPs before the program execution. However, once the probe is injected into the code, Uftrace can not safely remove or even dynamically modify it.

Google Xray leverages another user-space compiler-assisted technique to simplify cross-modification during run-time. By using the manually-activated *-finstrument-functions* flag in LLVM (as listed in Table 4.3), it reserves 11 bytes at function entry by inserting a 2-byte *jmp* at the probe site to jump over the following 9-byte reserved NOP sleds [234]. The 2-byte *jmp* plays the role of a lock and is aligned in memory not to straddle the cache. At run-time, the NOPs would be patched easily as they are jumped over and thus remain unreachable. The probe is finally unlocked by patching a non-straddler 2-byte *jmp*, eliminating concerns about cross-modification issues.

Compiler-assisted techniques demonstrate significant shortcomings when compared to other

Table 4.3 manually-activated flags used for compiler-assisted techniques

Compiler	Flag/Attribute	Description
LLVM	<i>patchable-function</i>	Guarantees that the first instruction of the function is greater than 2 bytes, and will be sufficiently aligned.
	<i>patchable_function_enter</i>	Inserts a NOP sled as the first instruction of the function for Google XRay instrumentation.
	<i>patchable__ret</i>	Replaces the original <i>ret</i> instruction with the NOP sled at runtime for Google XRay instrumentation. This is the approach to go on CPUs that have a single <i>ret</i> instruction, such as x86.
	<i>patchable_function_exit</i>	Prepends the original return instruction with the NOP sled code preserving the original return instruction just after the NOP sled code.
	<i>-fcs-profile-generate</i>	It is used for CPUs which have multiple options for the return instruction, like ARM.
	<i>-forder-file-instrumentation</i>	Generates instrumented code to collect context-sensitive execution information, such as function entry and exit frequency and branch execution counters.
	<i>-fxray-instrument</i>	Generates instrumented code to track the order of function execution, especially during the startup phase of an application.
GCC		Generates XRay instrumentation sleds on function entry and exit
	<i>-pg -mnop-mcount</i>	Inserts a 5-byte NOP instruction at the entry of functions.
GCC & LLVM	<i>-fpatchable-function-entry=N</i>	Generates N NOPs right at the beginning of each function.
	<i>-finstrument-functions</i>	Generates calls to instrument function entry and exit.
	<i>-fprofile-generate</i>	Generates instrumented code to collect runtime information such as function call frequencies and branch execution information

instrumentation methods. They require access to the source code to perform instrumentation. Furthermore, the probing locations are defined during compilation, and instrumentation of arbitrary probe sites is not permitted afterward. Finally, as instrumentation code exists at every probe site during run-time (such as function entries/exits), this approach can bring higher memory and time overhead to the execution, even when tracing is disabled. Chamith et al. measured a total overhead of 6% added by the *-finstrument-functions* compiler flag [143]. This could be reduced to 2% when smaller functions are left uninstrumented, as implemented in Google XRay.

Apart from compiler-assisted techniques, compiler-agnostic techniques have the capability of fully dynamic instrumentation without generating place-holders at compile-time for run-time patching. These techniques do not require recompilation and do not incur any overhead when inactive. However, they still suffer from major shortcomings. In branch-based Kprobe, a 5-byte *jmp* at the probe site may be inserted if no branch is found targeting the patching area within the current function [210]. Uprobe allows for a fully dynamic instrumentation for user-space processes. However, its implementation in the kernel relies on slower traps and requires root privileges. It is faster than the trap-based approach implemented completely in user space as it avoids mode switching between kernel and user space. It is still much slower than the branch-based approach, though. While Dyntrace uses both branch-based and trap-based approaches, Liteinst is limited to exploiting the branch-based approach for dynamic instrumentation [143, 208]. Authors in [143] showed that, on average, the embedded trap was invoked in 14% of the total probe invocations, significantly affecting probe effectiveness. In Dyntrace, the probing instruction is written atomically at the probe site using *cmpxchg* instruction, which is not sufficient to ensure safe cross-modification operation. Liteinst uses an

improved version of the *wordpatch* algorithm for safe cross-modification [235]. The algorithm waits for a specified upper-bound number of cycles during which modified straddled cache line(s) will not be fetched for execution. The number of cycles was chosen empirically by conducting repeated experiments until no GPF exception was raised. Moreover, the upper bound value depends on the processor model and should be individually extracted for each one.

In summary, both existing compiler-assisted and compiler-agnostic approaches for dynamic instrumentation have shortcomings. Although compiler-assisted approaches are efficiently integrated with the build process, they suffer from source code dependency, fixed probing locations, high compilation overhead, and compatibility challenges across different compilers. On the other hand, the existing compiler-agnostic approach, despite offering greater flexibility, provides low probe insertion and execution rates. This method is only applicable for long instructions and requires costly memory access management and synchronization mechanisms. To address these limitations, we developed a new DBI framework called NOProbe, which leverages NOP and pseudo-NOP instructions to enhance flexibility and performance.

4.4 NOProbe Design and Architecture

In this section, we propose a new multi-strategy framework for DBI called NOProbe. Our novel contributions cover a comprehensive range of instruction patterns (<1 byte) and conditions at probe sites, aiming for efficient instrumentation. Without having control flow information, our contributions accomplish transparent instrumentation while the control flow of the program remains intact. It is noted that we are not leveraging compiler-generated space to inject instrumentation code into a program at runtime as the compiler-assisted approach does. NOProbe gets the advantage of the optimization-level-activated flags to propose a new technique for instrumenting instructions as short as 2 bytes and longer. Moreover, NOProbe introduces another novel instrumenting technique for short instructions that is completely compiler-agnostic. It is not dependent on debug information, such as symbol names, type info for symbols, files, and line numbers where the symbols come from; thus, making possible the patching of stripped binaries.

4.4.1 Assumptions

While NOProbe is compatible with a wide range of x86 binaries, we make some basic assumptions about the architecture and algorithms discussed in subsequent sections. The operational performance of NOProbe relies on four assumptions:

No return address manipulation by the callee: To achieve function exit instrumentation, we replace the return address pushed by the caller on the stack with the address of a trampoline. We assume that the callee function does not interfere with the probe by overwriting the return address. Moreover, the callee is not expected to read and find the caller's return address from the stack. Although it is rare, techniques used by virtual machines like on-stack replacement might cause such interference [236]. Stack unwinding in *C++* is another example of a situation where problems could arise. This could easily be solved by hooking the corresponding functions to restore the call stack when an exception is raised.

No branching from outside the instrumented function: Injecting a probe for DBI requires replacing multiple bytes. We thus need to verify whether any branching instruction would target one of the bytes overwritten after the first byte at the probe site. To limit the scope of this verification, only the branching instructions within the instrumented function are examined. It is assumed that no branches originate from another function to the instrumented function, except for its entry point. This assumption generally holds in the *C/C++* standard.

No data embedding in the NOPs operand: It is assumed that the target program does not use NOP-paddings to embed data. Manipulating those bytes to insert probing instructions would cause interference. While programs rarely do this, some anti-disassembly and obfuscation techniques use the paddings to overlap instruction streams, to hide some portions of the code in the program [224].

No instruction overlapping by the executable itself: Instruction overlapping used in code obfuscation techniques involves intertwining multiple instructions to appear convoluted [237] [238]. It makes the disassembly and static analysis more difficult. We assume that the program does not perform code overlapping by branching in the middle of another instruction intentionally.

4.4.2 Multi-strategy Probing

In this section, we elaborate on the new complementary strategies we have devised for instrumentation in NOProbe. The proposed Algorithm 1 represents a complete overview of our instrumentation approach. It is used as a reference for the rest of the text.

As a framework, NOProbe implements the conventional branch-based approach for target instructions that are at least 5 bytes long. It also handles shorter instructions that form a minimum 5-byte basic block with no branches to internal instructions (line 4 in Algorithm

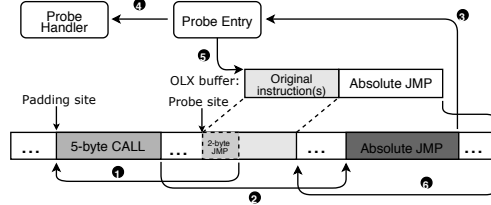


Figure 4.8 NOProbe execution flow with a 2-byte *jmp* as probing instruction

1). The *call* instruction pushes the return address on the stack, which will be used as the probe identifier (line 5 in Algorithm 1). A 5-byte relative *call* is used as probing instruction at the patch site (line 6 in Algorithm 1).

First Strategy

We developed a novel strategy tailored for binaries compiled with optimization-level-activated flags at the most widely-used level of O2 [239] [240] or higher. This strategy is independent of manually-activated flags (listed in Table 4.3) used by compiler-assisted approach (line 8 in Algorithm 1). When the conventional branch-based approach fails, the first assumption is to have a short instruction of 4 bytes, 3 bytes, or 2 bytes at the probe site while a branch lands on the patching area (line 9 in Algorithm 1). As demonstrated in Fig. 4.8, this strategy employs a smaller probing instruction (2-byte relative *jmp*) compared to the conventional branch-based approach [241]. Using a 2-byte basic block helps proportionally reduce the risk of cache straddling. This strategy would make the target probe location ineligible for instrumentation in cases where a single-byte instruction is located at the probe site, with potential branches targeting the subsequent instruction. Having a single byte as offset, the 2-byte *jmp* can only reach target addresses ranging from 128 bytes before to 127 bytes after the probe site. However, this range may not be large enough to reach a trampoline that eventually redirects control to the probe handler. To address this issue, we introduce NOP-paddings as a safe rewritable byte sequence surrounding the probe site. If an unreachable NOP-padding is available within the 2-byte *jmp* range, it can be used to insert an initial trampoline including a relative *jmp* with a 4-byte offset (step 1 in Fig. 4.8). This trampoline then directs control to the secondary trampoline (step 2 in Fig. 4.8), which contains an absolute *jmp* with a 5-byte offset, ensuring access to the probe handler across the entire memory (steps 3 and 4 in Fig. 4.8). By executing the original instruction(s) stored in the OLX buffer, step 5 in Fig. 4.8 is completed. An absolute *jmp* at the end of the OLX buffer redirects the control flow to the immediate instruction followed by the overwritten original instruction(s) at the probe site (step 6 in Fig. 4.8).

Algorithm 1 Instrumentation routine

```

1: procedure INSTRUMENTNOPROBE(addr)
2:   Disassemble the executable in the vicinity of the probe site address
3:   Loop through each disassembled instruction
4:   if ((I0.size >= 5) || ((2 <= I0.size < 5) && (no branch lands on [I0[0] + 1, I0[0] + 4]))
   then
5:     initCallProbe(addr)
6:     patchSiteCall5B(addr)
7:     success  $\leftarrow$  True
8:   else if (code compiled using optimization-level-activated flags (e.g. -O2, -O3)) &&
9:     ((2 <= I0.size < 5) && (no branch lands on I0[0] + 1)) then
10:    nop  $\leftarrow$  findSuitableNop(addr)
11:    if nop  $\neq$  None then
12:      initNopProbe(addr, nop)
13:      patchPaddingSite(nop)  $\triangleright$  with a 5-byte call
14:      patchSiteJmp2B(addr, nop)
15:      success  $\leftarrow$  True
16:    else if (code compiled at lower optimization level than -O2 ||
17:      optimization-level-activated flags applied with out-of-reach NOP-paddings then
18:      follow the Tranch-based technique
19:      if (I0[0] + 4) overlaps with the head byte of a neighbor instruction X (X[0]) then
20:        double-byte-dictionary [i] = {two-byte pseudo-nop instructions};
21:        single-byte-dictionary [j] = {one-byte pseudo-nop instructions};
22:        if (X.size equals 3 or more) then
23:          for each double-byte pseudo-nop instruction in the dictionary do
24:            X[0]  $\leftarrow$  pseudo-nop-dictionary [i]
25:            X[1]  $\leftarrow$  pseudo-nop-dictionary [i + 1]
26:            X[2]  $\leftarrow$  0xcc
27:            if the offset of the call instruction is mappable then
28:              success  $\leftarrow$  True
29:              break;
30:          else
31:            i = i + 2;
32:        else if (X.size equals 2) then
33:          for each single-byte pseudo-nop instruction in the dictionary do
34:            X[0]  $\leftarrow$  pseudo-nop-dictionary [j]
35:            X[1]  $\leftarrow$  0xcc
36:            if the offset of the call instruction is mappable then
37:              success  $\leftarrow$  True
38:              break;
39:          else
40:            j = j + 1;
41:        else if (X.size equals 1) then
42:          X[0]  $\leftarrow$  0xcc
43:          if the offset of the call instruction is mappable then
44:            success  $\leftarrow$  True
45:            break;
46:      else
47:        follow the Trap-based technique

```

The main challenge in this strategy is to discover suitable NOPs. Suitable NOPs are considered as any NOP-paddings in the vicinity of the probe site that provide us with potential placeholders for initial trampoline placement, even if it seems unfitting at first (line 10 in Algorithm 1). For the 2-byte *jmp* insertion at the probe site, a padding site with at least 5 bytes is necessary to fit a relative *call*. Consequently, only NOP-paddings that are 5 bytes or larger are considered candidates. In exceptional cases, the 5-byte *call* at the padding site overwrites two consecutive NOP instructions, forming a sufficiently large NOP-padding, as depicted in Fig. 4.7b. The first block consists of a 4-byte *nopl* and the second block is a 10-byte *nopw*. Therefore, the entire first block (4 bytes) and the first byte of the second block would be patched.

Any unreachable padding of 5 bytes or more is considered suitable for patching. On the other hand, reachable paddings should be approached with extreme caution. Modifying reachable paddings could alter program behavior, as the execution flow may eventually reach that point. For the 8, 9, and 10-byte reachable NOP-padding instructions, the probing instruction could be inserted within the unused memory operand, specifically in the last 5 bytes. Therefore, this placement ensures that it does not impact the operation, as illustrated in Fig. 4.9. The 7-byte reachable padding must be treated differently. As shown in Fig. 4.10, placing a 5-byte *call* on the last 5 bytes of the 7-byte padding will change the third byte of the NOP-padding (*0x80* indicating addressing mode *Mod R/M*). This modification changes the operation format. As a solution, the 7-byte NOP-padding should be patched with a 2-byte bypass relative *jmp* followed by a 5-byte relative *call* to keep the no-operation behavior untouched.

When instrumenting several locations, optimizing the global assignment of NOP-paddings is essential when multiple instances are within the range of each probe site. This optimization aims to maximize the success rate of probe insertions (line 12 in Algorithm 1).

Search and Assignment of Suitable NOPs Searching for suitable NOP-paddings in memory could be as simple as exploring instructions in the 256-byte range around the probe site. During the discovery phase, any suitable NOP-padding that is found is added to a list. Subsequently, the list is then processed by an algorithm that assigns the most suitable padding to every probe site as needed. However, this approach is not always feasible in certain cases. Consider a scenario where function boundaries need to be probed, with two functions *f1* and *f2*, and two NOP-paddings *N1* and *N2* available. *N1* may be within the range of both *f1* and *f2*, while *N2* can only be reached by function *f2*. A non-optimal heuristic algorithm could assign *N1* to function *f2* and leave *f1* unassigned. This underlines the importance of proper assignation of NOP-paddings to probe sites. This combinatorial optimization assignment

```

0:  0f 1f 84 00      nop    DWORD PTR [rax+rax*1+0x0]
    00 00 00 00

```

(a) Original reachable NOP-padding

```

0:  0f 1f 84 e8      nop    DWORD PTR [rax+rbp*8+0x4030201]
    01 02 03 04

```

(b) Integrating a probing instruction into the reachable NOP-padding while maintaining its semantic integrity

Figure 4.9 Embedding a 5-byte *call* in an 8-byte NOP-padding

```

0:  0f 1f 80 00 00 00 00      nop    DWORD PTR [rax+0x0]
7:  48 89 c8                  mov    rax,rcx

```

(a) Original reachable NOP-padding

```

0:  eb 04                  jmp    0x6
2:  e8 01 02 03 04        call   0x4030208
7:  48 89 c8                  mov    rax,rcx

```

(b) A *jmp* bypassing the NOP-padding followed by an inserted *call*

Figure 4.10 Inserting a 5-byte *call* in a 7-byte NOP-padding

problem deals with the allocation of multiple resources to the specified sectors on a 1-to-1 basis. This problem could be broadly defined as follows:

Assume that we have a set of distinct functions available to be instrumented and the list of all paddings reachable from each probe site is known. The algorithm is expected to pair each function with one padding. The definition of the linear assignment problem is formulated as follows:

Given a set of P functions and Q NOP-paddings, we have a weight matrix $\{c_{ij}\}_{P \times Q}$ where

$$c_{ij} = \begin{cases} 1 & \text{if NOP-padding } j \text{ is in the range of function } i; \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm should return a bijection such that the function z is maximized:

$$\text{maximize } z = \sum_{i=1}^P \sum_{j=1}^Q c_{ij} x_{ij}$$

The resultant bijective matrix is a binary matrix defined as $\{x_{ij}\}_{P \times Q}$ where

$$x_{ij} = \begin{cases} 1 & \text{if NOP-padding } j \text{ is assigned to function } i; \\ 0 & \text{otherwise.} \end{cases}$$

Finding $\{x_{ij}\}_{P \times Q}$ that maximizes z is equivalent to pairing as many NOP-paddings as possible with functions. The conventional algorithm employed to address this challenge is known as the Hungarian algorithm [242], which generates an optimal solution. It operates in polynomial time with a worst-case run-time complexity of $\mathcal{O}(n^3)$, where $n = P + Q$. Due to its time complexity, the Hungarian algorithm is preferably used when the number of instrumented functions remains relatively small. The algorithm can be applied to a small subset of functions since each padding is only reachable from a limited number of functions. For instance, it can be used on the instrumented function and its nearby functions to obtain a locally optimal assignment. However, using the Hungarian algorithm to achieve a globally optimal assignment by operating on all functions in a binary would be prohibitively expensive, especially when dealing with a large number of functions. For such cases, a greedy approach (Algorithm 2) has been developed as an alternative algorithm. The greedy algorithm takes the list of all the NOP-paddings within the function's range as input and generates a non-optimal solution with linear time complexity. For function entry instrumentation, any function alignment padding is located directly above the entry point and clearly falls

Algorithm 2 Greedy algorithm for NOP assignment

Input *funcList*: A list of functions near the probe site. ▷
Input *nopList*: A sorted list of potential NOPs. ▷
Output: A suitable *nop* if found else *None* ▷

```

1: procedure FIRSTSUITABLENOP
2:   for all nop ∈ nopList do
3:     if isFunctionPad(funcList, nop) then
4:       return nop
5:     else if isUnreachable(nop) then
6:       return nop
7:     else if canEmbedTrampoline(nop) then
8:       nop ← setEmbedFlag(nop)
9:       return nop
10:  return None

```

within the range. If NOPProbe uses a 2-byte *jmp* at the probe site, the target padding can be accessed within a range of -128 to 127 bytes from the current location. Hence, the same function alignment padding is likely to be out of range of the preceding function entry, unless its size is less than 127 bytes. In this context, the greedy algorithm prioritizes using the function alignment padding over labels, loops, and jumps alignment paddings. This decision is based on the fact that function alignment paddings reliably remain within the range of the current function entry and are seldom reachable from other function entries. The list of NOP paddings is arranged to prioritize those closest to the instrumented function, ensuring they are visited first. Upon discovering the most appropriate NOP-padding through the search and assignment algorithm, a 5-byte *call* is inserted within the padding site. This instruction is responsible for transferring the control flow to the secondary trampoline, as outlined in line 13 of Algorithm 1. Eventually, the probe site would be patched with a 2-byte *jmp* (line 14 in Algorithm 1).

As discussed, function boundaries instrumentation can be achieved by inserting a probe at the entry and another probe at each exit point. When a function exit (usually marked by a *ret* instruction) is positioned at the end of the function’s address space, the probing instruction may overlap with the NOP-padding aligning the next function entry. If this NOP-padding is not sufficiently long, an alternative method is employed. Upon function entry in this method, the probe handler saves the return address and replaces it with the address of the function exit probe handler. This ensures that the function exit probe handler is invoked at the function exit. Subsequently, at the end of the function exit probe handler, the saved return address is restored in the call stack and a normal return to the caller is performed.

If static code analysis cannot confirm the presence of a 2-byte basic block or identify suitable

NOP-padding within range, we must resort to the second strategy.

Second Strategy

The previous strategy relies on the activation of widely used optimization-level flags, like *-falign-loops* to make use of generated NOP-paddings in the performance-sensitive locations all over the code. However, the second proposed strategy is independent of the optimization-level-activated flags listed in Table 4.1. Similarly, this strategy does not rely on the manually-activated flags listed in Table 4.3, which introduce intrinsic compiler dependencies.

If the previous approaches fail, we make four assumptions. First, the conventional branch-based approach cannot instrument a short instruction (<5-byte) at the probe site while threads jump to the middle bytes of the offset. Second, the first proposed strategy cannot locate a 2-byte basic block at the probe site. Third, the first proposed strategy cannot find NOP-paddings generated by optimization-level-activated flags available at compile time (lines 16 in Algorithm 1). Additionally, the first proposed strategy may find NOP-paddings but they are not in the vicinity of the probe site (line 17 in Algorithm 1).

The second strategy develops a new tranch-based approach by overcoming the potential shortcomings of its conventional version. As explained, the conventional tranch-based approach attempts to patch those short instructions at the probe site (line 18 in Algorithm 1) that the branch-based approach cannot handle. Oftentimes, it cannot succeed if the fourth byte of the probing instruction offset overlaps with a neighbor instruction. The fourth byte is the most significant byte of the offset as it significantly extends the addressing range of the probing instruction. Instrumenting typical performance-sensitive examples, such as loop and function entries suffers from this overlapping limitation (Fig. 4.4b and Fig. 4.5b) as mentioned in line 19 in Algorithm 1. Loop entry starts with three instructions in a 2-byte, 2-byte, and 3-byte format (step 1 in Fig. 4.11a). A common arrangement for instruction placement at a function entry comprises 1-byte, 3-byte, and 4-byte instructions (step 1 in Fig. 4.11b). These two cases are covered in line 22 in Algorithm 1. In step 1 of Fig. 4.11c, a random combination of 4-byte and 2-byte instructions is arranged (line 32 in Algorithm 1). Through the second step, the conventional tranch-based approach inserts a branch opcode at the probe site (*ll*) and a trap instruction into the head bytes of all subsequent instructions within a 5-byte range. As a result of step 2 in these figures, the fourth byte of the offset becomes fixed with the opcode of *0xcc*. By fixing the most significant byte of the probing instruction offset with *0xcc*, the conventional tranch-based approach encounters the most restrictive condition in exploring the trampoline address within the memory space.

Our proposed tranch-based approach overcomes this limitation by changing the location of

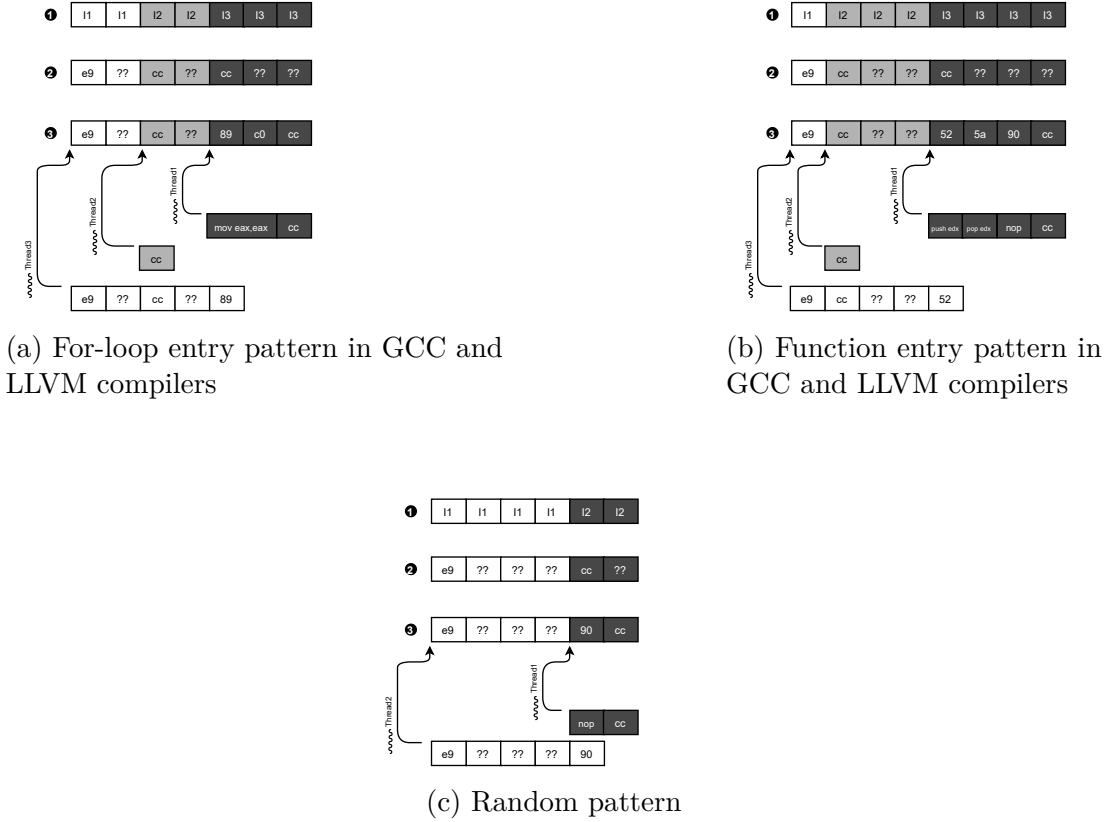


Figure 4.11 Probe insertion at $I1$ using the conventional and the proposed trench-based approaches on three popular instruction patterns

$0xcc$ placed on the fourth byte of the offset within the subsequent instruction layout. This instruction is represented as $I3$ and $I2$ in Fig. 4.11a and 4.11c, respectively. By sliding the $0xcc$ to the far right byte of these instructions, we make space for inserting one or more pseudo-NOP instructions at their first bytes (step 3). If the overlapping neighbor instruction has three or more bytes, the fourth byte of the offset accommodates the first byte of a 2-byte pseudo-NOP (line 24 in Algorithm 1) listed in the third column of Table 4.2. The second and the third bytes of the neighbor instruction will be replaced with the second byte of the pseudo-NOP and the relocated trap ($0xcc$), respectively (lines 25 and 26 in Algorithm 1). On the other hand, when the overlapping neighbor instruction covers two bytes (line 32 in Algorithm 1), this strategy replaces the first byte with a 1-byte pseudo-NOP instruction (listed in the second column of Table 4.2). Simultaneously, the relocated trap sits on the second byte of the neighbor instruction (lines 34 and 35 in Algorithm 1).

Integrating a pseudo-NOP instruction may not always result in a successful address mapping on the first attempt. In such cases, the next entries in the second column of Table 4.2

will be attempted repeatedly until success is achieved (lines 23 and 33 in Algorithm 1). This accommodation provides a higher chance of discovering the trampoline’s address by exploring a broad range of values from *0x0f* (the first bytes of *indv*) to *0xfd* (the first bytes of *{std, cld}*) in Table 4.2. In contrast, the conventional tranch-based approach only considers *0xcc* as a candidate for this position.

Fig. 4.11 shows comparative instrumentation for the conventional and the proposed tranch-based approaches across three commonly encountered instruction patterns. Following step 3 of Fig. 4.11b, the probe insertion for instruction number 1 (*I1*) has been completed. A thread landing on the first byte of *I1* executes the 5-byte probing instruction. In another case, thread number 2 jumps to the first byte of the *I2* and executes a trap (*0xcc*) instruction there. Eventually, thread number 3 may reach the first byte of the *I3* and execute a pseudo-Nop (such as *mov eax, eax* in this case) followed by a *nop* instruction. The control flow subsequently reaches the far-right byte to execute the trap instruction that was initially relocated. In this scenario, the arrangement of bytes in the pseudo-NOP instruction that overwrites *I3* directly affects the displacement calculation for the *call* instruction. Therefore, the binary rewriting algorithm prioritizes fixing the pseudo-NOP before exploring the potential displacement for the *call* instruction placed at the probe site.

As illustrated by line 47 in Algorithm 1, we apply the conventional trap-based approach when neither of the previous strategies is applicable. As explained in Section 4.2.1, this approach uses *int3* with the *0xcc* byte-code as a less efficient but always applicable probe.

4.4.3 Static Analysis

Before inserting a probe into a function, a static analysis is conducted to determine whether to use a 5-byte *call*, 2-byte *jmp*, or *int3* (trap). If the original instruction at the probe site is five bytes long or longer, it can easily be patched using the 5-byte *call*. Otherwise, the function is disassembled and the target of every branch is examined. If no potential branch landing on subsequent overlapped instruction(s) (with the 4 bytes of *call* offset) is identified, it can still be patched with the 5-byte *call*. However, branches may land on the patching area without specifically targeting its second byte. In this case, if suitable NOP-paddings are available nearby, we propose adopting the first strategy as a new branch-based approach using a 2-byte *jmp* instruction. Otherwise, there may be instances where landing branches target the patching area, including its fifth byte, without any suitable NOP-paddings discovered nearby. In this restrictive scenario, the fifth byte overlaps with the opcode of the next instruction (See Fig. 4.3c). For this case, we propose adopting the second strategy as a new tranch-based approach. Consequently, a 5-byte *call* is inserted while replacing the branch-targeting

bytes of the offset with a trap instruction, except for the fourth byte. For 2-byte overlapped instructions, a 1-byte pseudo-NOP instruction listed in Table 4.2 sits on the fourth byte of the offset following the relocated trap (refer to Fig 4.11c). However, if the overlapped instruction is three bytes long or more, a series of pseudo-NOP instructions will be inserted starting from the fourth byte of the offset which follows the relocated trap (see Fig. 4.11a and 4.11b). As a last resort, the probe site can always be instrumented with a single-byte trap instruction.

Through static analysis, we also identify the unreachable function paddings inserted between functions, particularly for the first strategy. To achieve this, we extract the function entry address and size for each function from the Executable and Linkable Format (ELF) files. This allows us to discover the gaps between functions. Moreover, jump paddings inserted using *-falign-jumps* may also become unreachable, especially if they follow an unconditional branch with no incoming branches. During analysis, when an indirect branch (with a register as an operand) is found in a function, we conservatively assume that it could branch anywhere in the function. Thus the search for possible incoming branches will always be affirmative for such functions.

4.4.4 Safe Patching Setup (lock-redirect-load-arm)

Rewriting binaries at runtime requires a safe patching process in multithreaded applications. Ensuring the integrity of instrumentation involves the safe and simultaneous execution of both the original overwritten instruction(s) and the probing instruction. Such modification creates a consistent scheme in which threads are enabled to execute the original overwritten instruction(s) without incoherency. Also, another mechanism is needed to provide a solid scheme for threads to execute the probe transparently. We explain both mechanisms in detail in the following sections.

Simultaneous Execution Probing In Fig. 4.12, we intend to patch a 2-byte instruction (*I1*) with a 5-byte probing instruction. When NOProbe overwrites multiple original instructions at the probe site, a thread might become trapped in the middle of the replaced instructions, as depicted in step 1 of Fig. 4.12. So, the concurrent thread could experience undefined behavior if patching occurs without special care. Therefore, ahead of writing a multi-byte probing instruction to a patching area that contains two or more instructions, it is necessary to apply the lock and redirect mechanisms:

1. Ensure that no new thread will enter the patching area (lock).

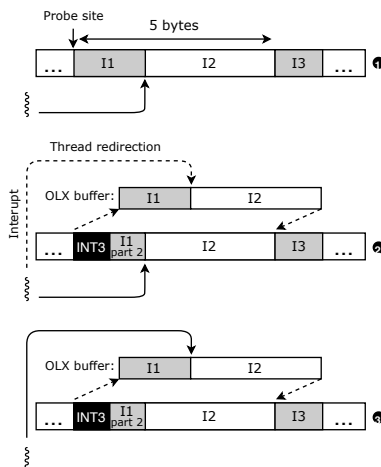


Figure 4.12 Lock the probe site and redirect threads to the OLX buffer

2. Move the threads already present out of the patching area (redirect).

The original instructions that are overwritten by a probing instruction are first copied to the OLX buffer. Then, a trap instruction (*int3*) is inserted into the probe site, acting as a temporary trap-based probe (step 2 in Fig. 4.12). Patching or removing a 1-byte trap instruction is a safe atomic operation and does not result in an incoherent state on Intel x86 platforms. The main role of the trap instruction is to lock and secure the patching area, ensuring that all the incoming threads are redirected to the OLX buffer. This approach helps us avoid the invasive stop-the-world action, which would otherwise stop all threads simultaneously.

The second concern focuses on the threads currently executing, interrupted, or preempted within the patching area. If these threads resume execution after the area has been patched, they might execute invalid instructions. Hence, they should not be permitted to remain there during the patching process. Using signals, the other threads can be interrupted from user space to execute a signal handler. The instrumentation thread sends an asynchronous real-time POSIX signal to all the other available threads in the process. The signal handler verifies if the PC of the receiving thread falls within the patching area. If it does, the thread changes its interrupt return address. This modification ensures that the receiving thread is redirected from the patching area to the corresponding original instruction in the OLX buffer (refer to step 3 in Fig. 4.12). Then, the signal handler clears a flag for the receiving thread, informing the instrumentation thread that its task has been completed. After all the receiving threads clear their flags, the instrumentation thread resumes execution. At this time, no thread is left in the patching area, and no new thread can enter it. Finally, the

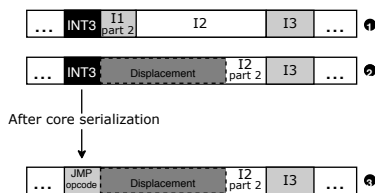


Figure 4.13 Patching the probe site with a 5-byte probing instruction while ensuring serialization

probing instruction can be safely inserted using the load and arm mechanism explained in the following section.

Cross-modifying Probing An atomic store mechanism can safely patch a single-byte probing instruction without causing incoherence for the processor cores. For longer probing instructions that straddle the cache, they should consider the Intel erratum and issue a serialization instruction, such as *cpuid*, *iret*, or *rsm* on other processor cores. This ensures that any modifications made to flags, registers, or memory by previous instructions are resolved before the fetch and execution of the next instruction in the pipeline. Issuing a serialization instruction must take place before other threads execute the patching area. At step 1 in Fig. 4.13, the incoming threads to the patching area are blocked with a trap and rerouted to the OLX buffer. The thread(s) residing within the patching area are also redirected to the proper place in the OLX buffer. In the second step, the offset of the *call* probing instruction can be loaded. Subsequently, the *cpuid* serialization instruction is issued, either by sending a real-time POSIX signal or by utilizing the *membarrier* syscall². The *membarrier* syscall, using the command *MEMBARRIER_CMD_PRIVATE_EXPEDITED_SYNC_CORE*, triggers an *smp_mb()* memory fence, via an inter-processor interrupt (IPI) to all processors where the threads of the process are currently running. When returning from the IPI, the interrupt return (*iret*) instruction performs the required serialization.

Once the pipeline, memory, and cache lines of each processor core involved have been serialized, we proceed to the third step. In this step, the 1-byte trap instruction at the head of the probe site can be atomically and safely replaced with a 1-byte *call/jmp* opcode (arm). This activates the *call* probing instruction.

²*Membarrier* has been added to the Linux kernel since version Linux 4.3. For this reason, signals are used for older kernels.

Table 4.4 Evaluation Machine Specifications

Processor	4x Intel Xeon E7-8867 v3 @2.50GHz
No. of Cores	64 with Hyper-Threading disabled
Operating System	Debian GNU/Linux 9.0 and Linux 4.19.0-6-amd64
Compiler & Libraries	GCC/G++ 8.3.0 and GLIBC 2.28-10
Memory	256GB

4.5 Results

In this section, we empirically and statistically analyze the distribution of NOPs in program binaries. We also conduct three micro-benchmarks to assess the performance of instrumentation mechanisms in NOProbe. Table 4.4 displays the specification of the environment in which our experiments were conducted. The performance overhead of NOProbe includes both probe injection and execution costs. To maintain the accuracy of our benchmark, we chose not to use the POSIX timer API due to its high overhead in multi-threaded environments, mainly caused by sequential locks. Instead, we employed the time stamp counter (TSC) along with a memory fence instruction [243], which incurs a negligible overhead compared to the total benchmarking time.

We chose the Uftrace tracing tool as an efficient user-space prototyping vehicle for implementing the new algorithms in NOProbe. Unlike the current version of Uftrace, the legacy version preloads (*LD_PRELOAD*) an instrumentation library named *libmcount* before forking the target program. This instrumentation library takes care of the whole pipeline, from reading the symbol table (required for the first strategy) to creating shared buffers, hooking into functions, instrumenting the target program, and initiating the target program execution while tracing. The *LD_PRELOAD* technique is a simple and effective approach, but it does not enable the legacy Uftrace to attach the instrumenting library to processes during run time. At the time of our experiment, the version of Uftrace we used did not support dynamic process attachment. We manually enhanced the legacy Uftrace, enabling it to attach to existing processes and insert instrumentation at runtime. However, the current experimental version of Uftrace includes this feature.

On-the-fly instrumentation of the tracee requires *libmcount* to be injected into its address space. To this end, the *ptrace* system call is used for a safe and race-free injection. Once the modified uftrace is attached to a running tracee, it pauses the program using the *ptrace* system call. Aiming to minimize the pause duration, a lightweight library named *libloader* is injected. Using *ptrace*, this library resumes the tracee and loads a set of other required libraries. After injection, *libloader* creates a dormant thread that waits on a Unix socket

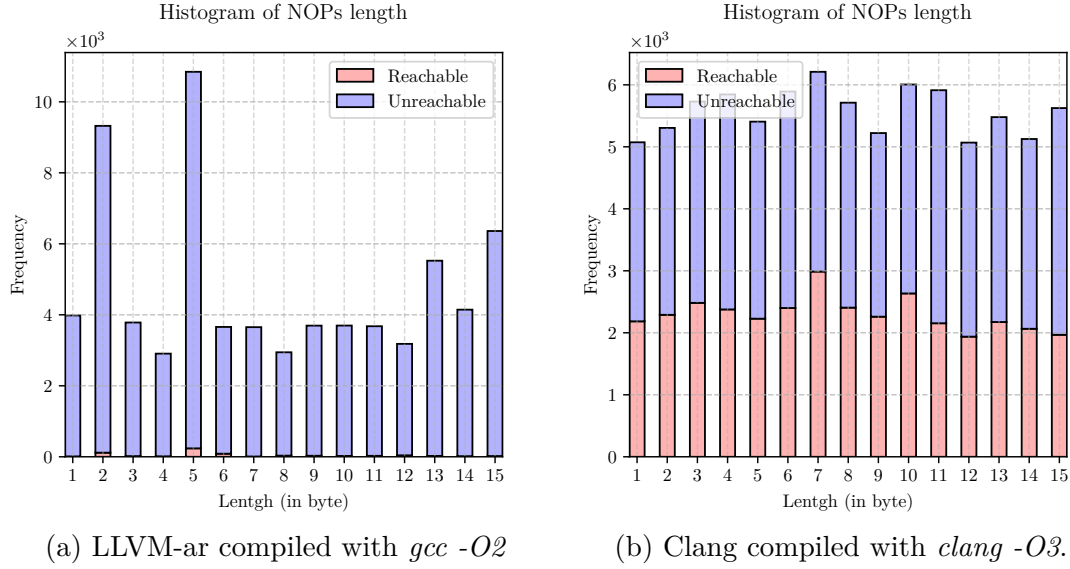


Figure 4.14 The distribution of reachable and unreachable NOPs at different lengths in LLVM-ar and Clang

to receive commands. The tracer program can then send commands to this thread to load (*dlopen*) the *libmcount* library for instrumentation and to initiate tracing (*start_tracing()*).

4.5.1 NOPs Accessibility

NOPProbe exploits NOP-paddings within a binary to serve as a readily accessible trampoline holder. We analyzed the distribution of NOPs in two popular software packages, LLVM-ar and Clang when compiled with GCC and Clang. Fig. 4.14 illustrates the uniform probability distribution of NOP lengths and the proportion of reachable and unreachable paddings. It demonstrates that the paddings in LLVM-ar and Clang can be identified as unreachable in 99% and 58.67% of cases, respectively. This variability can be attributed to compiler design, optimization level, coding style, etc. NOPProbe can utilize any unreachable padding within the range of the probe site and any reachable padding larger than 6 bytes, as discussed in 4.4.2. In Fig. 4.14a, approximately 51k out of 71k NOPs are 5 bytes long or more. This implies that approximately 71.83% of NOPs in LLVM-ar and 73.74% of NOPs in Clang could potentially be used by NOPProbe. Such an empirical experiment can also be supported statistically.

To estimate the expected average length of NOP-paddings in a binary aligned on 16 bytes, we calculate the gap size (in bytes) from the last instruction in that block to the alignment boundary. Assuming a uniform probability distribution for the gap size, we can determine the

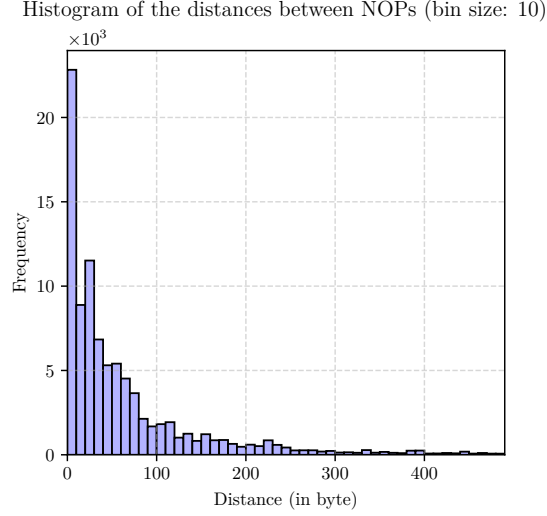


Figure 4.15 The inter-NOPs distance distribution in LLVM-ar

probability of having 5-byte or longer NOP-paddings capable of accommodating the 5-byte *call* instruction as follows:

$$\begin{aligned}
 P(NOP_{len} \geq 5) &= P(NOP_{len} = 5) + P(NOP_{len} = 6) + \dots + P(NOP_{len} = 15) \\
 &= 1/15 + 1/15 + \dots + 1/15 = 11/15 = 0.73
 \end{aligned}
 \tag{4.1}$$

Fig. 4.15 presents the distribution of inter-NOPs distances for LLVM-ar. Notably, the density of [0-128] bytes inter-space is significantly higher than the others. Specifically, 82.59% and 71.92% of NOPs in LLVM-ar and Clang are within 128 bytes of each other. Considering that a 2-byte *jmp* with a 1-byte displacement can reach up to 128 bytes away, there is a high probability of accessing one NOP within easy reach of the probe site.

4.5.2 Probe Effectiveness

In this section, we evaluate the instrumentation success rate achieved by well-known tools. This rate is defined by the number of successfully inserted fast probes using 5-byte *call* instruction (without using a trap-based approach) divided by the total number of target probe locations. We chose GDB and Dyninst for the branch-based approach, Dyntrace for the tranch-based approach, and NOProbe which employs both branch- and tranch-based approaches.

In this experiment, we attempted to insert a 5-byte *call* probing instruction at every function entry point in five large widely-used applications– Chrome, LLVM, Git, Vim, and Nginx. To

prevent potential interference between probe structures in memory, we inserted the probes one by one, leaving them installed until the experiment’s completion. It is worth noting that we instrumented the functions within the main executable, leaving shared library functions untouched. Table 4.5 provides detailed insights into the instrumentation effectiveness reflected by the number of successful insertions divided by the total number of instrumentable locations.

Compilers typically insert a prologue into most functions to set up registers and stack for the function execution. As explained in Section 4.4.2, the function prologue often begins with a short (<5) instruction. Therefore, the probability of finding a 5-byte instruction at the function entry point is quite low. This fact significantly impacts GDB’s success rate as it requires a minimum of 5-byte instruction in the prologue to employ a branch-based approach.

The operation of GDB is based on the injection of a shared library, called in-process agent (IPA), into the target process. The IPA initializes trampolines, known as jump pads, and establishes a tracing infrastructure within the target process. However, a challenge arises in x86_64 architectures where the *.text* section of a binary typically resides at lower addresses compared to shared dynamic libraries. Consequently, a 5-byte *call* instruction placed at a probe site within the *.text* section may not reach the trampoline within the IPA. To address this issue, the experiment manages the compiler to load both the *.text* section and the IPA section in close proximity during link time.

Dyntrace suffers from a similar limitation when the *.text* section is loaded into the lowest mappable memory area. In its branch-based approach, if a trap instruction (*0xcc*) is placed in the fourth byte of a *call*’s displacement at the probe site, the offset becomes negative. As a result, Dyntrace is forced to map the probe structure before the *.text* section, which is not typically feasible by default. This scenario is common in many functions within Git and LLVM-ar, resulting in poor instrumentation effectiveness, as illustrated in Table 4.5.

On the other hand, DynInst demonstrates an outstanding probe instrumentation success rate. This is expected since DynInst completely rewrites and relocates functions. For function boundary instrumentation, DynInst does not deploy the trap-based approach; instead, it can relocate the entire function to a new location. Although this approach delivers a high success rate and flexibility, it comes with substantial costs in terms of attachment overhead and memory usage.

By conducting the probing operation in user space, NOProbe effectively minimizes the overhead typically associated with context switching. Unlike the branch-based approach seen in GDB and Dyntrace, NOProbe allows for user-space instrumentation of any instruction longer than 1 byte with the help of suitable NOPs. Additionally, NOProbe overcomes the

Table 4.5 Number of successful fast probing across different benchmarks

Benchmark	GDB			NOProbe			Dyntrace			Dyninst		
	Success	Total	Rate	Success	Total	Rate	Success	Total	Rate	Success	Total	Rate
chrome	28982	326205	8.88%	310497	326205	95.18%	-	-	-	320216	326205	98.16%
llvm-ar	523	19940	2.62%	19104	19940	95.80%	9318	19940	46.73%	19940	19940	100%
git	1	9901	0.01%	9815	9901	99.13%	4291	9900	43.34%	9901	9901	100%
vim	0	6396	0%	6279	6396	98.17%	5506	6396	86.08%	6396	6396	100%
nginx	137	1148	11.93%	1109	1148	96.60%	1054	1148	91.81%	1148	1148	100%

trampoline placement limitation found in the trench-based approach of Dyntrace by utilizing pseudo-NOPs. Through the introduction of two new strategies, the NOProbe instrumentation framework effectively addresses the gaps in instrumenting short instructions in user space, resulting in higher performance compared to GDB and Dyntrace.

4.5.3 Probe Insertion Performance

In this section, we evaluate the probe insertion costs across Dyninst, Uprobe, and NOProbe, while varying the number of threads in the tracee and the size of its functions.

Multithreading Overhead In this evaluation, we measure the overhead caused by instrumentation when conducted by a single thread. The instrumentation thread inserts probes in 1000 function entries and exits, maintaining a constant number of probes and function size while varying the number of threads. The cost per instrumentation point is calculated by dividing the total time taken to instrument all functions by the number of instrumented functions. To minimize the jitter, the test is repeated 100 times and the results are averaged.

Our experiment shows that Dyninst attaches very slowly to the target, taking nearly six seconds. The reason lies in using *ptrace* for attaching to the tracee. In this scenario, all the threads within the target process will be paused and only resumed after the instrumentation is completed. Moreover, Dyninst utilizes all available core resources in the system (64 cores in this case) to analyze and instrument the target. While measuring probe insertion overhead, we excluded the attachment cost for Dyninst. However, it is important to mention that we selected immediate instrumentation mode over batch mode for Dyninst. This decision could slightly impact its performance due to the inherent caching ability of batch mode, thereby reducing inter-process communication (IPC) cost [244].

Fig. 4.16 illustrates the probe insertion cost for Uprobe, NOProbe, and Dyninst. In NOProbe, the instrumentation thread sends an interrupt signal to all threads at the time of patching. It then waits for their acknowledgment before proceeding with the patching. This approach can lead to delays, especially if a thread is blocked while waiting for a resource.

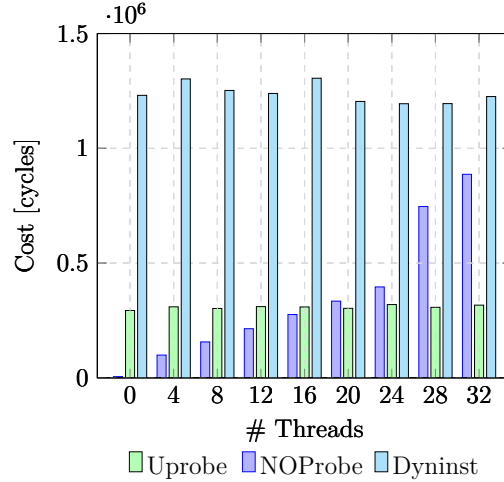


Figure 4.16 Probe insertion cost versus the number of running threads

Queued POSIX signals are handled once a thread returns from kernel space. In cases where signal-receiving threads are waiting in kernel space due to locks (used by *read/write* syscalls to protect kernel resources), the signal-handling process may experience further delays, particularly when dealing with a larger number of threads.

Although the Dyninst instrumentation cost is high, it scales well with the number of threads. This efficient scaling is because *ptrace* can stop all the running threads before patching. The NOProbe outperforms other tools when the thread count remains below twenty. However, beyond this threshold, the Uprobe instrumentation cost maintains stable performance, as Uprobe operates within the kernel and is not blocked waiting for threads returning to user space.

In this experiment, we employed *membarrier* for serialization in kernels version 4.3 or newer, while using real-time signals for older kernels. *Membarrier* significantly surpasses signal-based synchronization by minimizing context switching overhead and employing a direct synchronization mechanism.

Function Size Overhead This test evaluates how NOProbe scales with the size of instrumented functions where probes are inserted. To this end, we measure the average time taken to insert a probe into every function across six large applications. Then, we calculate the average size of all instrumented functions within each application. For NOProbe, we separate the instrumentation cost associated with static analysis from that associated with the *lock-redirect-load-arm* patching cost (using *membarrier*). Fig. 4.17 illustrates the average time taken by NOProbe to instrument six binaries. They are sorted and characterized by their av-

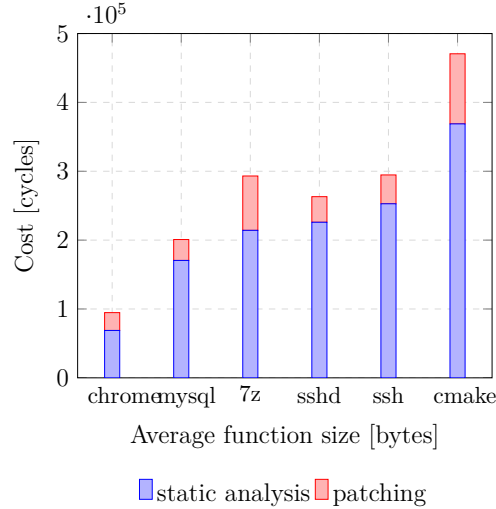


Figure 4.17 Probe insertion cost versus the average function size of multiple binaries

average function sizes: Chrome (65 bytes), Mysql (234 bytes), 7z (251 bytes), Sshd (331 bytes), Ssh (357 bytes), and Cmake (466 bytes). NOProbe is designed to identify NOP-paddings placed by the compiler in the locality of the patching area, essential for implementing the first strategy. Depending on the scarcity of these placeholders and the layout of the neighbor instructions, NOProbe may make use of the second strategy as well. For either strategy, a static analysis is conducted on every instruction within a function to discover branches leading to the patching areas. Subsequently, the most fitting strategy is selected to initiate patching at the probe site. Therefore, as the size of a function increases, the static analysis routine must explore a wider range of code for potential branch discovery. The longer this discovery phase lasts, the higher the overhead imposed on the probe insertion process for either strategy.

4.5.4 Probe Execution Performance

In this section, we evaluated the probe execution overhead across multiple instrumentation tools. The overhead is calculated as the additional CPU cycles incurred beyond the normal execution without probe insertion. To evaluate this, we conducted a benchmark by varying the number of concurrent threads executing the probe and measured how the overhead scales with the thread count. Specifically, we probed the entry and exit points of a sample function in this benchmark. Because GDB lacks native support for instrumenting function entry/exit points, we forced the compiler to insert a 5-byte instruction at the function entry. This approach enables us to accomplish only function entry instrumentation with GDB. The probe

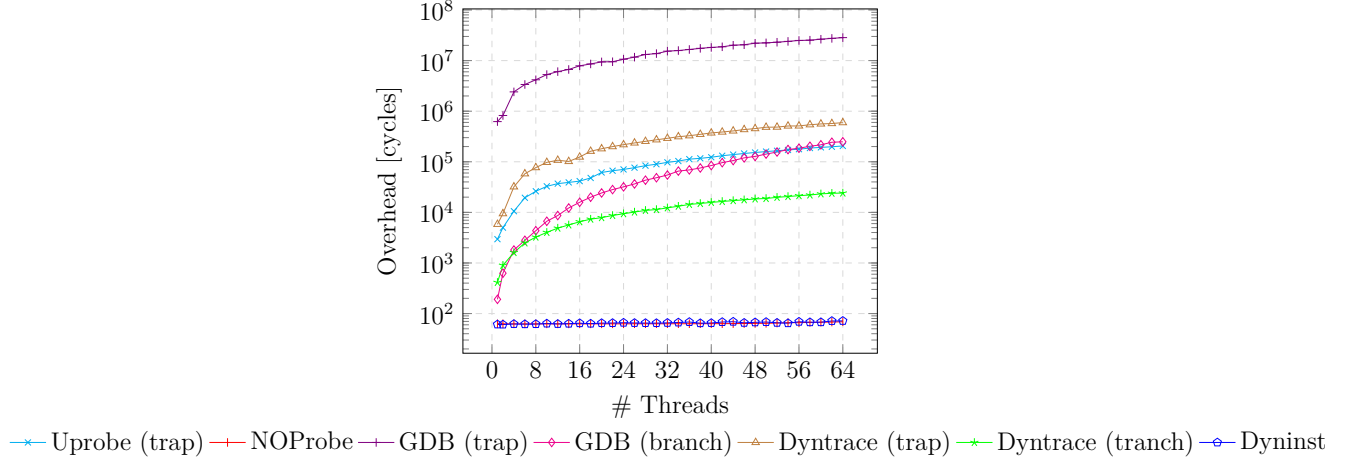


Figure 4.18 Probe execution overhead for Uprobe (trap-based), NOProbe, GDB (trap-based and branch-based), Dyntrac (trap-based and branch-based), and Dyninst

handler used for Dyninst, NOProbe, and Dyntrac is a user-space function performing no operations. This approach was intentional, as we aimed to measure solely the cost of having a probe, excluding the cost of tracing or other operations. Default behavior differs for GDB and Uprobe: GDB does not allow users to override its probe handler, whereas in Uprobe, replacing the probe handler from user space is not feasible. To address this, we recompiled GDB and statically disabled the call to its probe handler. On the other hand, we kept Uprobe unchanged, thus including the additional overhead of saving an event in a per-CPU ring buffer in the results.

We initially executed the probed function with varying numbers of threads, repeating the experiment 1,000,000 times for each configuration. The total execution time was divided by the product of 1,000,000 repetitions and the thread count to calculate the average execution time per thread. Finally, the overhead was determined by subtracting the execution time of the probe-less function from that of the probed one. Fig. 4.18 depicts the execution overhead when multiple threads execute a probe simultaneously. As expected, the trap-based approach utilized by GDB, Uprobe, and Dyntrac performs poorly compared to the others. All three trap-based tools scale similarly with the number of threads. The increased overhead can be attributed to the synchronization mechanisms employed by each tool to ensure coherent execution. Moreover, transferring a trap signal to user space involves kernel operations like acquiring spin-locks and updating atomic reference counters to protect the signal handlers' structure. It is worth mentioning that GDB's performance is particularly poor due to its GDB-server using *ptrace* to receive and handle trap signals while the target is stopped. The target is resumed only after the probe operation is completed.

The branch-based probing approach implemented by Dyntrace and GDB outperforms the trap-based implementations in Dyntrace, GDB, and Uprobe. However, GDB fails to scale well as it acquires an exclusive lock in the trampoline (jump pad), to protect the tracing buffer from concurrent *write*-s [245]. Dyninst probes do not need any lock to protect their structure, because the patching procedure is not performed on the fly. *Dyninst* stops all threads using a *ptrace* syscall and transfers them out of critical regions before patching the code.

In NOProbe, a Red-Black tree as a self-balancing binary search tree maintains balance during insertions and deletions of relocated and replaced instruction(s), with a guaranteed time-bound operation. While *write* operations to the lock-less Red-Black tree require costly atomic (lock) instructions, *read* operations do not. Consequently, NOProbe possesses the lowest overhead for probe execution. The results for NOProbe and Dyninst are overlapped in Fig. 4.18. The NOProbe execution overhead scales very well with the number of threads. However, the *read*-s may miss an entire sub-tree of a node that is being atomically modified. An alternative could be a lighter-weight synchronization mechanism like user-space Read-Copy-Update (RCU), which offers low overhead and solid scalability. In RCU, the writer side waits until all existing reader-side critical sections are finished before making any modifications to the tree. Although the reader side avoids any waiting, a minimal overhead may be added, depending on the library implementation [246].

4.6 Conclusion

Energy tuning for multithread applications is crucial in performance analysis for resource-constrained devices. Discovering energy hot spots in those complex applications requires collecting resource utilization or system call requests. Tracing tools provide such granular information while bringing insights into the application energy behavior by inserting instrumentation probes. Efficient probes use user-space binary rewriting techniques to minimize the overhead of insertion and execution of instrumentation code.

Existing user-space instrumentation approaches suffer from intricate limitations in discovering unmapped memory space for efficient trampoline placement. This paper proposes NOProbe, a DBI framework that implements two novel strategies to accommodate trampolines efficiently and enhance user-space instrumentation. By leveraging NOP and pseudo-NOP instructions, we have significantly improved the success rate of probe insertion. Our experiments demonstrate that by avoiding a trap-based approach, our probe execution overhead is remarkably reduced compared to existing techniques in both user and kernel spaces. In terms of probe insertion rate, NOProbe outperforms its best competitor for probe execu-

tion overhead, Dyninst. Compared to the single-byte trap-based Uprobe, NOProbe exhibits superior performance, especially with up to 20 threads. Additionally, we proposed two algorithms for managing NOP-paddings, one emphasizing performance and the other focusing on instrumentation success rates. Finally, our improved algorithm based on the *membarrier* system call and the *lock-redirect-load-arm* mechanism ensures a safe code patching technique. Through our experiments, NOProbe achieved probe effectiveness rates ranging from 97% to 99%, far surpassing conventional approaches in existing tools like GDB, which achieved rates between 0% and 11%.

In the future, we aim to improve the instrumentation success rate of NOProbe by integrating additional high-performance instrumentation strategies. Additionally, we will focus on developing methods for safe, scalable, and efficient probe removal from user space.

CHAPTER 5 ARTICLE 2: "DEEP LEARNING-DRIVEN ANOMALY DETECTION FOR GREEN IOT EDGE NETWORKS"

Authors Ahmad Shahnejat Bushehri, Ashkan Amirnia, Adel Belkhiri, Samira Keivanpour, Felipe Gohring de Magalhães, Gabriela Nicolescu

Submitted on February 24, 2023 and published on November 22, 2023 in IEEE Transactions on Green Communications and Networking, Volume 8 Issue 1, pages: 498-513, DOI: 10.1109/TGCN.2023.3335342

Abstract The widespread use of sensor devices in IoT networks imposes a significant burden on energy consumption at the network's edge. To address energy concerns, a prompt anomaly detection strategy is required on demand for troubleshooting resource-constrained IoT devices. It enables devices to adapt their configuration according to the dynamic signal quality and transmission settings. However, obtaining accurate energy data from IoT nodes without external devices is unfeasible.

This paper proposes a framework for energy anomaly detection of IoT nodes using data transmission analysis. We use a public dataset that contains peer-to-peer IoT communication energy and link quality data. Our framework first utilizes linear regression to analyze and identify the dominant features of data communication for IoT transceivers. Later, a deep neural network modifies the gradient flow to focus on the dominant features. This modification improves the detection accuracy of anomalies by minimizing the associated reconstruction error. Finally, the energy stabilization feedback provides nodes with insight to change their transmission configuration for future communication.

The experimental results show that the proposed approach outperforms other unsupervised models in anomalous energy detection. It also proves that redesigning the conventional loss function by enhancing the impact of our dominant features can dramatically improve the reliability of the anomaly detection method.

5.1 Introduction

An Internet of Things (IoT) scenario on edge is defined through a set of wireless sensor networks connecting a collection of widely distributed monitoring sensor devices (SD) [2]. This scheme would collect and send the desired data to the edge servers for further processing and decision-making [5]. Such sensor devices feature a severe constraint on the hardware

resource, such as stripped RAM size, low-rank CPU/GPU capability, and limited energy harvesting [7]. However, they are well-adopted in autonomous systems for steady data collection, stable transmission [8], continuous diagnosis [9], and real-time analysis [247]. Energy consumption is counted as one of the most challenging constraints for SD service provisioning. Energy over-consumption occurs in cases like amplifying the transmission power for fading shadowing avoidance. Similarly, when SDs are located in remote areas and distant from gateways, their vulnerability to increased path loss becomes more pronounced, particularly in terms of energy efficiency. Hence, energy optimization techniques are required to increase the battery lifetime of SDs and maintain high-efficiency levels [248].

Most software-layer energy optimization models perform at the levels of systems, architecture, or circuits [18] [19]. The outcome of these energy minimization approaches is categorized into multiple groups for embedded computing devices [20]. The first is called Dynamic Power Management (DPM) [249]. DPM temporarily halts any idle element of the system to avoid wasting energy, like processing cores. In the second group, Dynamic Voltage-Frequency Scaling (DVFS) permits CPUs to switch between different frequency-voltage levels to scale down energy consumption [22]. To save a significant amount of energy expended on generic tasks on the third group, CPUs are customized to meet the requirements of the tasks on a multi-processor system-on-chip [23]. The same customization happens for the fourth group but for the cache-based memory access [24]. By mapping tasks to different processors, the fifth group performs load balancing across all the cores which results in the utilization enhancement of processing elements on a multiprocessor system-on-chip [250].

The mismanagement of energy consumption can result in over-heat generation. The non-uniform heat generation leads to spatial temperature gradients over the whole chip. The more the load on a system caused by concurrent processing, the higher temporal heat generation/dissipation occurs. This may impose the temporal thermal gradients on a small area on the chip. The spatial temperature gradients and temporal thermal gradients besides the thermal cycles are three main factors that degrade the performance and the efficiency of a system during its lifetime. Ignoring overheating in edge nodes may also cause performance, durability, and reliability loss. For example, raising the temperature by ten to twenty degrees in metallic materials would shorten the useful life of the components by sixteen times [17].

However, applying the discussed solutions on a peer-to-peer network is not highly effective in reducing energy consumption for SDs and alleviating the consequences. These approaches do not consider transmission energy consumption as the primary cause of battery depletion for SDs in a wireless sensor network [251]. Although sensor devices are typically designed as a single processor board, most discussed approaches are devised for multiprocessor system-on-

chips. For example, over-heat generation is much more prone to happen in multiprocessors than single processor SDs as the former suffers from non-homogeneous heat dissipation in its processing elements.

SDs can be classified into normal, sink, and gateway nodes. One of the main advantages of these resource-constrained devices located in hostile environments [252] is forwarding the collected data to the gateway nodes [253] for further analysis and decision-making (DM) [254]. The nodes with a higher rate of forwarding DM data are more susceptible to energy failure. By collecting energy and other transmission data (link quality parameters and configuration settings), an AI-based model can detect anomalies in energy behavior. As supervised learning solutions require data labeling for energy over-consumption detection [255], they are not practically feasible for continuous learning in IoT domains. Consequently, there is a need to investigate if other energy-saving approaches, including unsupervised models and routing protocols, may prolong the survival time of the whole network reliably [256].

The principal contributions of our study can be summarized as follows.

- Propose a new energy-aware framework for the detection of energy consumption anomalies that might affect IoT nodes' lifetime in a wireless sensor network. This framework collects peer-to-peer communication energy and link quality data and uses a deep-learning approach to detect the abnormal energy behavior of the SDs;
- Propose a learning solution for anomaly detection based on an unsupervised algorithm. We design a new loss function for the conventional variational autoencoder by analyzing the feature importance of the data using a linear regression model;
- Evaluate the proposed approach by using a public dataset. Our results show that our approach achieves high-performance metrics and less intrusive execution time compared to other alternatives;
- Implement the proposed approach with other baseline clustering algorithms and unsupervised deep-learning models, including multi-layer VAEs, to make a comparison between their effectiveness and that of our proposed solution.

This paper is organized as follows. In the following section, we introduce energy-saving approaches at the network level. The problem statement and research question are presented in Section 5.3. Data and theoretical analysis are provided in Section 5.4. The methodology and the details of the energy-aware framework are explained in Section 5.5. The results of the experiments are described in Section 5.6. Finally, conclusions are drawn in Section 5.7.

5.2 Background and Related Works

Minimizing energy footprint can be performed at the node and cloud levels. At the node level, depending on the communication scenario, such as the location of nodes and the signal quality, data transmission is the primary cause of energy consumption of an IoT node in wireless sensor networks [257]. At the cloud level, data centers as powerful processing units are the decisive elements in the cloud computing domain. They host several power supply blocks, control systems, data communication, storage, and cooling systems, in addition to security components. Evaluating energy consumption in such complex systems is challenging as it requires a deep investigation of various connected elements [76]. We present the above-mentioned energy-saving approaches at the node and cloud levels in the following sections.

5.2.1 Node-based Solutions

For IoT sensor nodes, resource exhaustion occurs in transmit (TX) and receive (RX) states [77]. Several solutions are offered to remedy the high energy consumption rate and increase the life expectancy for sensor nodes in an IoT scenario. Without working on reliable energy-saving strategies for IoT nodes, a strain will be put on the energy harvesting sources to provide sufficient supply for operational activities. There are four primary categories for network-based energy-saving approaches on IoT nodes: sleep/wake-up protocols, radio optimization, energy-efficient routing, and data reduction [78]. As we focus on dynamic link quality and transmission settings in this work, the first two approaches are investigated here to represent the role of protocols in the transmission energy consumption of a node.

Radio Optimization

Radio optimization is a set of approaches applied to a radio module as the most energy-hungry element in a sensor network to optimize the required energy for data communication. This type of radio optimization manages the energy consumption regime by manipulating the radio parameters. To this end, several optimization techniques for power transmission, modulation, and antenna direction are suggested. Moreover, a cooperative communication strategy and cognitive radio approach improve the remaining energy in wireless sensor communication [128]. The cognitive radio, a prominent strategy against energy over-consumption in retransmissions (caused by packet loss), enables a transceiver to distinguish and avoid the occupied channels and access the vacant channels for data communication. Based on channel states, IoT nodes can pick a proper channel in the wireless spectrum and accordingly regulate the TX/RX parameters [129]. By comparing the cognitive-radio-enabled sensor

networks with the classic ones, researchers in [130] indicated a 13% improvement in the network's lifespan. In cooperative communication strategies, a virtual multi-antenna array is achieved by using multiple single-antenna nodes and sharing antennas with them, which results in a remarkable enhancement of the received signal strength (RSS) [82]. Evaluating these strategies shows that a considerable amount of energy could be saved for contributors in a wireless sensor network. This regime designates a set of nodes as relays and uses them to distribute the communication load by reducing the distance between IoT nodes and the gateway. Hence, the required energy for data collection through the gateway would be cut, and the lifetime of the sensor network would be extended.

Selecting proper antennas affects the energy performance of sensor networks significantly. For example, replacing an omnidirectional with a directional antenna limits the unwanted interference caused by retransmissions [131]. The enhanced energy comes from the fact that if a transmission is done toward a specific direction, the neighbor communications could also be accomplished without interfering with others [132]. Although the advantages of removing omnidirectional antennas have been demonstrated, there are practical difficulties in direction adjustment and identifying proper parameters for directional antennas. One of the solutions for the radio modulation parameter selection is the modulation optimization strategy that contributes to energy minimization [85] [86]. By carefully setting the modulation parameters in a wireless sensor network, a balanced compromise between the transmission power and the inter-node distance could be devised to regulate an appropriate circuit/transmission power consumption. In radio communications, the transmission power is managed using the Dynamic/Transmit Power Control (DPC/TPC) mechanism [87]. While maintaining the necessary link quality parameters, such as the Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI), DPC reduces the power of a radio transmitter. By avoiding interference, it results in network life span extension as well [88]. However, applying radio optimization techniques such as adaptive power adjustment at a high rate makes communications vulnerable to packet loss.

Sleep/Wake-Up Protocols

A practical solution for prolonging the life span of event-driven wireless sensor networks is to change the behavioral pattern of IoT nodes for data communication with the edge. The idea behind behavioral tailoring in event-driven networks is to intermittently let the IoT nodes switch to sleep/awake mode. While an event is scheduled, the nodes would be in the awake state (radio on) just before data communication is initiated, and they would be switched back to sleep mode (radio off) as soon as the data stops flowing (no-event avail-

ability). Offering such a mechanism effectively contributes to energy footprint optimization in a resource-constrained edge network. For establishing a connection in this scenario, the sender and receiver should acquire a priori knowledge about the sleep-wake schedule of the neighbors (synchronization step). Given that information, the endpoint switches to the active mode right before the pre-determined moment and is subsequently involved in the data communication. To conduct the synchronization step, a topology control protocol could be used while adjusting the communication duty cycle between individual IoT nodes. The duty cycling technique offers an energy efficiency scheme by enabling nodes to switch between sleep and awake modes at regular intervals [89]. As the amount of consumed energy in awake mode doubles, compared to that of sleep mode [90], tuning the duty cycle expands the sleep period while limiting idle listening time [91]. Duty cycling turns the radio on by detecting the preamble of a low-level carrier. Optimizing the energy consumption could be carried on through the topology control techniques (connectivity- and location-oriented) [133] [93], as they minimize the number of essential IoT nodes to meet the network connectivity and coverage requirements. The connectivity-oriented protocols for wireless sensor networks toggle IoT nodes between ON and OFF states dynamically to verify inclusive coverage and guarantee connectivity among nodes. On the other hand, according to the physical location of the IoT nodes, location-oriented protocols decide which nodes are qualified to be put in ON mode and when this transition takes place in a wireless sensor network to ensure network connectivity and coverage. While this approach reduces energy consumption considerably, it suffers from a weakness. Real-time data collection cannot be performed using this technique as it requires a high-duty cycle of an IoT node and, subsequently, results in high energy consumption.

5.2.2 Cloud-based Solutions

Most proposed approaches addressing high energy consumption are grouped into idle server shutdown and workload consolidation. However, machine-learning-based solutions have recently seen an upward trend in energy optimization. M. Demirci in [94] presented the produced heat, virtual machine placement, task scheduling, and power distribution as the main contributors to power consumption, while the environmental control overhead and processing overload control are considered side effects. However, resource prediction without a priori knowledge of allocated tasks to the data centers usually comes with a large error margin. Thus, an efficient predictive model can significantly enhance energy consumption by providing an optimized allocation approach. DeJaVu [135] is presented as a cloud-based resource management system. The authors made a classification for load consolidation through a supervised machine-learning approach. As a result, the system started to be trained by an-

analyzing the allocation process for varying workloads. After this step, DejaVu has already learned to classify any unseen workload in two ways. It checks if the load is similar to any of those previously seen or not. If so, the exact allocation applies. Otherwise, it is treated as a new sample to be learned. As another load forecasting strategy, a neural-network-oriented predictor is employed in [96] to forecast how large the cloud will be loaded in the future. According to that, the unnecessary running machines would be switched off to prolong the total uptime of the system. Such green scheduling solutions can keep the same productivity with fewer servers while consuming less than 60% of energy compared to the standard scenario. Other works [258] anticipate running tasks' future energy footprint and service level agreement metrics like response time using supervised machine learning. The insightful feedback provided by such estimation helps the maintenance team devise energy-efficient task scheduling for high-demanding data centers. Applying unsupervised machine learning (ML) [134] could estimate the required resources by virtual machines. By evaluating the resource estimation, this approach reduces the consumed energy in data centers by turning unnecessary machines into sleep mode. A centralized approach in [99] is used for resource allocation using Q-learning in Heterogeneous cloud radio access networks. Despite a considerable energy budget saving by this framework, their analysis reports that the interference mitigation is met and the quality of service is acceptably preserved among users. Though the data analytics approaches discussed so far are related only to data centers, leveraging machine learning decision-making to resource-constrained edge devices is also a high-demanding trend. Due to the limited hardware/software capabilities of edge devices for IoT applications, mapping large-scale ML algorithms is not practical. However, an energy-efficient in-memory computing kernel is proposed for linear classification [100]. They use a split-data-aware mechanism to adjust voltage, process, and temperature changes. This results in an admissible balance between energy efficiency and accuracy features. A summary of the discussed energy-saving mechanisms is presented in Table 5.1.

Most introduced energy over-consumption detection approaches use external devices such as smart plugs [119] [120] [121] or are designed for smart buildings rather than computing devices [259] [260] [261]. Some other strategies follow general-purpose models that do not meet the constraints for edge nodes or use dynamic energy and thermal models with high overhead [262] [263] [127]. Edge-based energy modelings are mostly application-specific and may not necessarily target the overall energy consumption of the edge node. In some others, the energy requirements are known in advance based on contextual information such as location, time, and device capabilities. In others, the energy-controlling regime will mainly be deployed unacceptably late. Through our work, we target the limitations of the existing energy anomaly detection models to devise a non-intrusive and affordable mechanism to reduce

Table 5.1 Relevant energy saving mechanisms in IoT domain

	Approaches	Methodology	Strategy	Articles
Energy Saving Mechanisms	Node_based	Radio Optimization	Cognitive Radio	[128], [129], [130]
			Cooperative Communication	[128], [82]
			Directional Antenna	[131], [132]
			Modulation Optimization	[85], [86]
		Sleep/Wake-Up Protocols	Transmission Power Control	[87], [88]
			Duty Cycling	[89], [91]
			Topology Control	[133], [93]
	Cloud_based	Node Sleeping	Unsupervised Learning	[134]
			Supervised Learning	[135], [136], [137], [138], [139]

the energy footprint of an edge network.

5.3 Problem Statement

A pattern in collected data belongs to an anomaly category if it does not conform to the defined expected behavior compared to other experiments [264]. In practice, a range of anomalies may affect the energy efficiency of a wireless sensor network or even expose the IoT nodes to complete failure. Early anomaly detection strategies for IoT nodes are enabled by either internal or external data collection. Internal data collection consists of continuous extraction of granular tracing data [265] [266] [267] [268] out of IoT nodes along with gauged energy consumption so that abnormal energy patterns can be identified and avoided in future scenarios. On the other hand, external data collection consists of monitoring the data communication among edge elements.

Detected anomalies can be categorized into two groups: point and contextual anomalies [269]. The former discusses a rare individual experiment that diverges from the energy of the majority of other observations. An example of this type of generated anomaly is a temporary sensor glitch that was raised from a bug in running software. On the other hand, contextual anomalies deal with a set of improper observations that occur in a particular time window, considering that their values do not reveal their nature of abnormal behavior at first glance. A possible verification process is to detect anomalous patterns by comparing and correlating the patterns of data in time-related periods (day hours vs. night hours, etc.) for a long run [270].

There are also cases in which a node works in an underconsumption state of its energy. Although we believe that this scenario could also be an anomaly, we decided to focus only on the over-consumption of energy to delimit the scope of our research in this paper. We believe that IoT nodes and sensor devices are designed to spend most of their lifetime in standby or sleep mode for an unknown period without intending to start any communication with other elements. Hence, they often do not spend energy on data transmission. Unlike obtaining a clear threshold for over-consumption, there is no valid minimum threshold for

underconsumption in WSN as explained. Accordingly, if an IoT node consumes energy much less than others, there is no clear threshold-based strategy to detect an anomaly just through transmission data analysis. Capturing the anomalies in energy underconsumption in this case, may require more information regarding the internal execution trace of the system such as system call, resource utilization, and timing analysis to unveil the potential failure/malfunction of software or hardware components. To this end, our concern in this paper is to avoid over-consumption and prolong the lifetime of a node as there are cases in which inaccurate node configuration may reduce its operational lifetime by a hundred times [271].

5.4 Data Description and Mathematical Analysis

In this work, we use a public dataset¹ which contains peer-to-peer IoT communication energy and link quality data. The data is collected and used by [272] [273] [274] to provide autonomous configuration of communication systems, sustainable modular solutions, and heterogeneous communication schemes for smart IoT nodes in various environments. The data collection occurs in a context where several IoT nodes are distributed around the experimental environment. IoT nodes with variant communication protocols for data transmission surround a unique gateway. Since the nodes are spread randomly around the gateway, the distance to the gateway varies in line of sight per node. Using variable transmission power and protocol for each node mapped in diverse geospatial locations results in various energy consumption at the transmitter's side and different RSS indicators at the gateway. The dataset is composed of 18448 observations, while each entry includes nine distinctive features as presented below:

- X position: X coordinate relative to the location of the gateway (0,0)
- Y position: Y coordinate relative to the location of the gateway (0,0)
- Scenario: Characteristics of the environment in which the transmission is carried out (i.e., indoors and outdoors)
- Distance: The distance in meters within which the gateway can be reached in line of sight
- Obstacles: The number of interfering obstacles placed between an IoT node and the gateway in line of sight

¹<https://www.kaggle.com/datasets/andregloria/p2p-iot-communication-energy-and-link-quality>.

- Protocol: The transmission protocol used by each IoT node
- Power: The transmission power used by an IoT node
- Energy: The energy used by an IoT node to carry out the transmission
- RSSI: The received signal strength indicator at the gateway

Although in this framework all the connections are established with only one gateway, it could be extended to an actual scenario with multiple gateways. In that case, for each transmission initiated between a certain node and the counterpart gateway, our proposed framework can apply diagnostic troubleshooting actions separately.

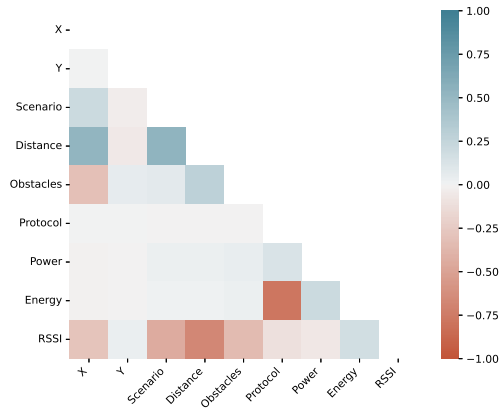


Figure 5.1 Covariance matrix of features

5.4.1 Feature Correlation

To display a clear picture of co-linearity between different features of observations, a correlation matrix heatmap is depicted in Fig. 5.1. Each cell in this matrix contains the correlation coefficient. Blue reflects positive, and red shows a negative correlation. The darker the color, the larger the correlation coefficient magnitude. Targeting *Energy*, the two most correlated features are *Protocol* and *Power*. In addition to the practical correlation investigation done by the covariance matrix, a study on theoretical correlation dependency is also required to find how transmission-oriented features, including *Protocol*, *Power*, and *RSSI*, affect the *Energy* consumption of IoT nodes.

5.4.2 Mathematical Analysis

In this part, a mathematical energy analysis for our model is provided to justify investigating the possible role of transmission parameters in energy consumption. Here, three terms are introduced during a machine cycle to represent the amount of time spent on each machine state. First, the amount of time for an IoT node in the awake state with the radio on is defined as $\text{Time}_{\text{active}}$. Besides that, the period just before initializing data communication is called $\text{Time}_{\text{sleep}}$ in which the radio turns off. The overall machine cycle ($\text{Time}_{\text{Total}}$) is hinged on the $\text{Time}_{\text{active}}$ in conjunction with $\text{Time}_{\text{sleep}}$ in a wireless sensor network [275].

$$\text{Time}_{\text{Total}} = \text{Time}_{\text{active}} + \text{Time}_{\text{sleep}} \quad (5.1)$$

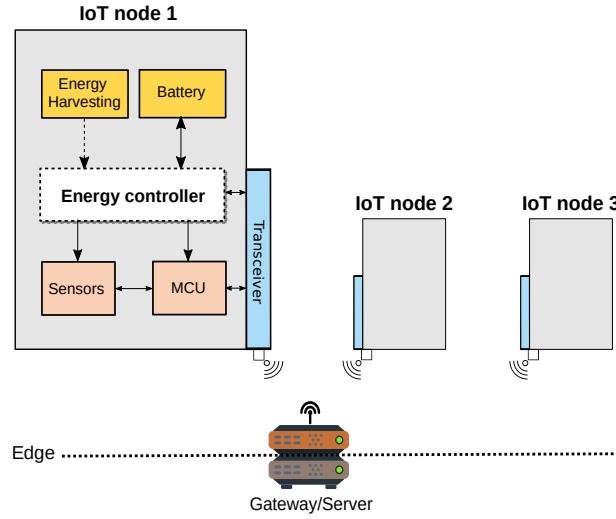


Figure 5.2 IoT nodes communicating with the gateway at the edge

Fig. 5.2 depicts a wireless sensor network on the edge side. For a machine cycle, the overall energy consumption for data communication in a wireless sensor network consists of $\text{Energy}_{\text{active}}$ and $\text{Energy}_{\text{sleep}}$. The first part of $\text{Energy}_{\text{Total}}$ is estimated based on the used energy for data transmission, reception, and task processing on the micro-controller unit (MCU) in the period of $\text{Time}_{\text{active}}$. The second part is estimated by the amount of energy consumed by all operating units like the sensors, MCU, and transceivers while the IoT node is in sleep mode during $\text{Time}_{\text{sleep}}$. The parameters in Equations 5.1 and 5.2 are summarized in Table 5.2.

Table 5.2 Parameters in energy consumption modelling

Parameter	Description
E_{Total}	The overall energy consumption for communicating to the sensor through the transmitter during one machine cycle
E_{active}	The energy utilization for the period of the active mode
E_{sleep}	The energy consumption during the sleeping mode by modules such as micro-controllers, transceivers, and sensors
$E_{\text{sensor_ON}}$	The energy consumed by the sensor during the active mode
$E_{\text{micro_ON}}$	The energy consumed by the micro-controller while processing the data
P_{sleep}	The power utilization for the period of sleeping mode
$P_{\text{sensor_ON}}$	The power consumed by the sensor during the active mode
P_{micro}	The power consumed by the micro-controller during the active mode
P_{RX}	The power consumption during receiving mode
P_{TX}	The power consumption during transmission mode
$T_{\text{sensor_ON}}$	The period in which a sensor is in active mode
$\text{Time}_{\text{Total}}$	The total time comprises of $\text{Time}_{\text{active}}$ and $\text{Time}_{\text{sleep}}$
$\text{Time}_{\text{active}}$	The amount of time for an IoT node in the awake state with the radio on
$\text{Time}_{\text{sleep}}$	The period just before data communication initiation in which the radio is off

$$\begin{aligned}
 \text{Energy}_{\text{Total}} = & \underbrace{E_{\text{active}}}_{\substack{E_{\text{sensor_ON}} + E_{\text{micro_ON}} \\ \underbrace{P_{\text{sensor_ON}} \times T_{\text{sensor_ON}}} + \underbrace{[(P_{\text{micro}} + P_{\text{RX}}) \times T_{\text{RX}}] + [(P_{\text{micro}} + P_{\text{TX}}) \times T_{\text{TX}}]}}} + \underbrace{E_{\text{sleep}}}_{P_{\text{sleep}} \times \text{Time}_{\text{sleep}}} \quad (5.2)
 \end{aligned}$$

In general, the signal broadcasted by the source is affected by multiple environmental factors associated with the geospatial information of the transmission's location. For example, RSSI indicates the estimated level of a radio signal power as it is delivered to the receiver endpoint. Since the T-R pathway might comprise multiple obstacles in between, the path loss behavior roughly follows the log-normal shadowing rather than deterministic modelings such as the ground reflection (Two-Ray) or log-distance approaches [276]. Besides, the signal power behavior does not conform with the log-distance path loss model along the T-R range. Consequently, considering the random variations in path loss for IoT nodes in any T-R range, the log-normal shadowing in Equation 5.3 estimates the RSSI while avoiding the symmetric path loss values [277].

$$\text{RSSI} = P_{\text{TX}} - \underbrace{PL_d}_{PL_{d_0} + 10\eta \log_{10}\left(\frac{d}{d_0}\right) + X_\sigma}, \quad (5.3)$$

Where, PL_d stands for a reference path loss in the range of random d meters from the transmitter, η is the path loss rate at which the RSSI drops with the distance d . X_σ in this equation defines the $X \sim \mathcal{N}(\mu, \sigma^2)$ with zero mean ($\mu = 0$) and standard deviation of σ .

The effect of transmission power on the communication range and the energy consumption of IoT nodes are investigated in [278] and adjusted based on geospatial features and the IoT node attributes. However, the correlation with the communication link is not linear, as multiple nodes transmitting at full power can generate interference in the network [279]. This shows that it is possible to reduce the energy consumption of the device and improve the network reliability by adjusting the transmission power of each end node. Equation 5.2 shows the relation between the total energy consumption of an IoT node in a different set of operating modes and the associated power consumption. On the other hand, there is an association between the power consumed for establishing a transmission on the transmitter side and the received signal strength on the receiver part based on Equation 5.3. In this equation, the η parameter is sensitive to the propagation environment and estimated by a larger value for a scenario with a higher number of obstacles between a transmitter and a receiver. The RSSI and the power of the received signal contain other crucial communication information, such as the transmission range between IoT nodes in wireless sensor networks [280].

Consequently, all features in the dataset but *Protocol* are theoretically correlated with the energy consumption of an IoT node. This article aims to validate how effective other transmission parameters, such as *Protocol*, are against the energy modeling of IoT nodes in wireless sensor networks.

As illustrated in Fig. 5.1 and excluding the *Energy* feature, there is only a relatively high negative correlation between RSSI-Distance and a high positive correlation between Distance-X and Distance-Scenario among all possible cases. There is also a minimum correlation between *Protocol* and all other features. While enhancing the decision-making process, such a low correlation avoids providing similar results and increases the execution time [275].

5.5 Methodology

In this section, we explain our proposed machine learning-based approach for energy anomaly detection at the edge. First, we define what an anomaly is in our scenario and how a sample can disrupt a predictable pattern in the data. Next, we discuss two deep unsupervised models for energy anomaly detection and propose a deep learning-based model as a novel specialized approach to this problem. In this section, we propose an energy-aware framework for a point-to-point (P2P) scenario to design an energy-saving framework at the edge of an energy-hungry wireless sensor network (Fig. 5.3). In this scheme, transmission settings from the IoT node's transceiver (see the blue dotted-arrow in Fig. 5.3) are collected together with the link quality parameters from the edge gateway (see the orange dotted-arrow in Fig. 5.3). The solution is based on a data analytic approach capable of detecting the over-energy-consumed

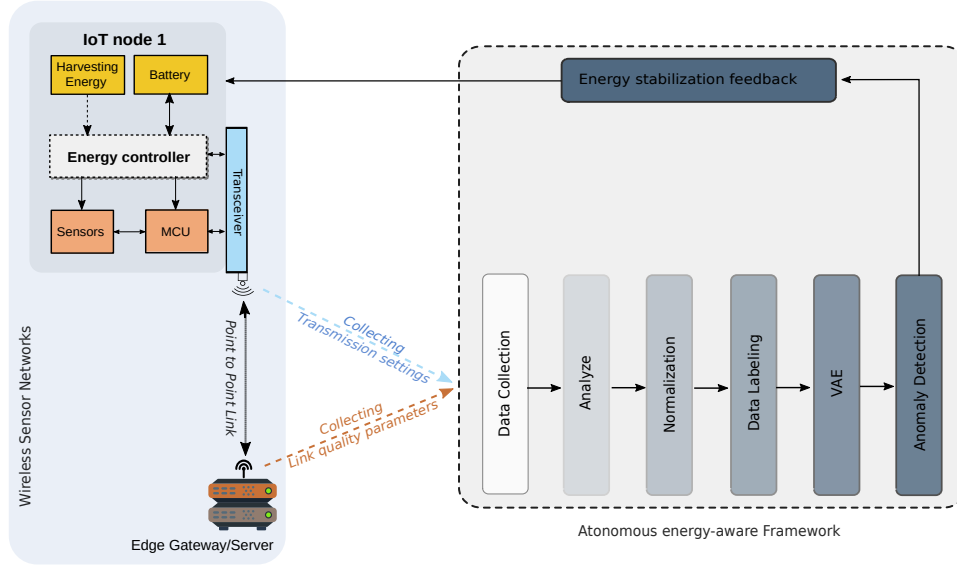


Figure 5.3 The proposed energy-aware Framework

IoT nodes during their transmission to the edge servers using various communication configurations. The decision-making process analyzes the observations based on the geographical location, environmental conditions, communication parameters, and link quality data. To evaluate the performance of each connection using different protocols, we study a public P2P dataset containing IoT communication energy and link quality data. Its contributors developed a script and implemented it on two ESP32 system-on-chip micro-controllers to emulate two wireless sensor endpoints. The dataset includes the X-Y coordinates of nodes obtained through GPS connection relative to the coordinate of the edge gateway as set to (0,0). To collect the RSSI value, multiple transmissions are investigated under various states, such as the position of nodes, the possibility of having indoor/outdoor scenarios, different node-gateway distances, and the number of intervening obstacles in between. By recording the communication parameters, such as transmission power consumption, and the RSSI of the signal, an array of data is created for every established link to the edge gateway. Additionally, all of the above-mentioned features are tested for the following communication protocols in data transmission: ESPNow, BLE, Radio Frequency 434MHz, LoRa, and ZigBee.

5.5.1 Data Labeling

The dataset used in our study comprises the communication configuration and energy consumption for P2P IoT nodes in wireless sensor networks. As the essential part of excessive energy detection lies in distinguishing normal and abnormal consumption, binary labeling fits well in this scenario. Table 5.3 describes the transmission power and the corresponding

Table 5.3 Protocol datasheet

Protocol	Transmission Power (dBm)	Energy Consumption (mJ)
BLE	7	150
RF	20	150
LoRa	23	120
ESP-NOW	1	88
ZigBee	8	40

transmission energy consumption for different communication protocols, according to the datasheets. The power consumption entries in this table are the highest official values for each certain protocol in the dataset. Thus, we decided to distinguish normal and abnormal data by defining the maximum energy value as a threshold for data labeling. In other words, the anomaly threshold setting is considered as any energy use of more than 150 mJ in data transmission. Any resultant energy entry provided in the dataset is designated as an anomaly if it surpasses the maximum energy consumption of all protocols (150 mJ) according to the datasheets (Table 5.3). To this end, the actual energy consumption in the dataset for different protocols is measured and provided using multiple values for all the variables, including transmission power. To detect energy anomalies in this array, we train an autoencoder exclusively on the normal energy data and later expect it to reconstruct the whole data while minimizing the reconstruction error. These observations will be classified as anomalies if the error exceeds a certain threshold defined by the maximum energy consumption of all the protocols under the experiment (Table 5.3). It is assumed that anomalies generate much higher reconstruction errors than normal data.

5.5.2 Data Normalization

The features of the utilized dataset have different distributions. Data normalization is a preprocessing step applied to datasets to improve the performance and reliability of machine learning models. It involves scaling features to a standard range or distribution, often without altering their relationships. Accordingly, we normalize each feature value to a range between zero and one. Equation 5.4 illustrates the normalizing formula used in this paper. In this case, $X_{\min}$ and $X_{\max}$ correspondingly represent the minimum and maximum feature values, and X_{scaled} shows the normalized version of a feature's value.

$$X_{\text{scaled}} = (X - X_{\min}) / (X_{\max} - X_{\min}) \quad (5.4)$$

5.5.3 Feature Importance

In training a machine learning model, each feature has a different importance that can be represented by a score. A higher score leads to a more significant impact on the model, while some have minor influences and can be ignored or removed from a dataset. In this part, to obtain each score, a basic discrete choice model is used. It is a linear regression model that is fitted to the dataset. A coefficient's absolute value (score) of a feature indicates the impact of that feature change on the energy consumption of nodes. The mentioned model is shown in Equation 5.5, where variables X_1 to X_8 represent features and $\hat{Y}$ represents the final output.

$$\hat{Y} = \sum_{i=1}^8 C_i X_i \quad (5.5)$$

Algorithm 3 illustrates the entire process of measuring the contributions of individual input features to the performance of our learning model.

Algorithm 3 Feature scoring

- 1: **Initialize the Model:** Create a linear regression model with randomly initialized coefficients:

$$\hat{Y} = F(X_1, X_2, \dots, X_8),$$

where $\hat{Y}$ represents the predicted values.

- 2: **Fit Initial Model:** Fit all data samples (X_i, Y_i) to the model to obtain the initial predicted values:

$$\hat{Y}^{(0)} = F(X_1, X_2, \dots, X_8).$$

- 3: **Calculate Initial Residuals:** Compute the residual error vector: $\epsilon^{(0)} = Y - \hat{Y}^{(0)}$, where $\epsilon^{(0)}$ represents the vector of residuals.
- 4: **Set Convergence Threshold:** Define a convergence threshold δ , representing the pre-defined tolerance.

- 5: **while** $\|Y - \hat{Y}^{(k)}\| > \delta$ **do**

- 6: **Update Model Coefficients:** Refit the model to update coefficients:

$$\hat{Y}^{(k)} = F(X_1, X_2, \dots, X_8).$$

- 7: **Recalculate Residuals:** Compute the updated residual error vector:

$$\epsilon^{(k)} = Y - \hat{Y}^{(k)}.$$

- 8: **Increment k :** $k \leftarrow k + 1$

- 9: **Rank Features:** Sort the absolute values of the model coefficients in descending order to determine the contributions of individual features.
-

Fig. 5.4 shows the absolute value for the coefficients of the features. As apparent, *Protocol* has

the highest coefficient absolute value compared to other features and has the most significant impact on the final output. On the other hand, several features such as X , Y , $Distance$, $Power$, and $RSSI$ have a lower contribution to the performance of the outcome.

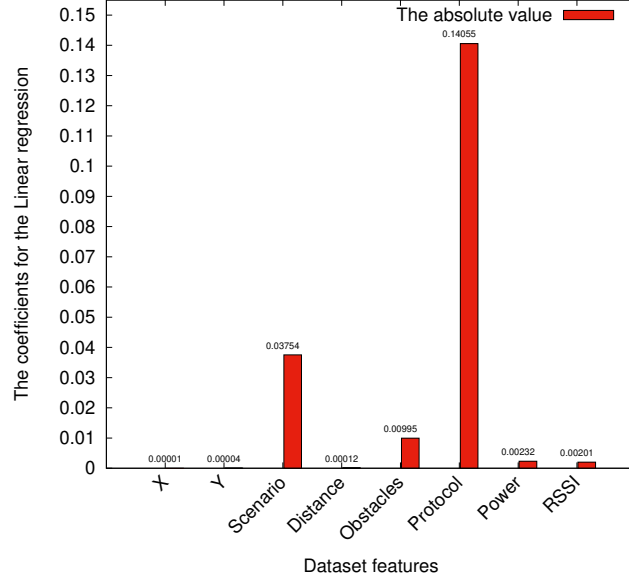


Figure 5.4 The significance of the features' impact on the energy consumption of nodes

5.5.4 Loss Function Design

As discussed in Section 5.5.3, a linear regression model allowed us to sort all features based on their impacts on the final output. The *Protocol* is found to be the most significant feature among others. In this part, we want to redesign the loss function of the conventional VAE by integrating *Protocol* into its definition. The goal is to guide the direction of gradient flow to focus more on the error between the *Protocol* feature in the original and reconstructed versions of data and adjust the model parameters accordingly. As a means of accomplishing it, the loss function is redesigned by adding a new term that represents the *Protocol* value error on the input and output of the model. Equation 5.6 demonstrates the new loss function ($\mathcal{L}_{\text{New}}$), where the new term contains the absolute value's square of the difference between the *Protocol* feature value in the original and reconstructed versions of data. $\mathcal{L}_{\text{recon}}$ is the standard reconstruction loss (e.g., MSE or binary cross-entropy) while the $\mathcal{L}_{\text{KL}}$ is the Kullback-Leibler divergence that regularizes the learned latent space. λ is a weighting coefficient that controls the impact of the *Protocol* term in the overall loss. In this formula, β is a hyperparameter that balances the $\mathcal{L}_{\text{New}}$ and $\mathcal{L}_{\text{recon}}$. It is common to set the value for β equal to 1 in a standard VAE. Moreover, we set λ to 1 to emphasize that the relevant *Protocol* term will have the

same weight as the other components of the loss function.

$$\mathcal{L}_{\text{New}} = \mathcal{L}_{\text{recon}} + \beta \cdot \mathcal{L}_{\text{KL}} + \lambda \cdot \left(|\hat{X}_{\text{protocol}} - X_{\text{protocol}}| \right)^2 \quad (5.6)$$

5.5.5 Deep Unsupervised Learning Model for Anomaly Detection

The proposed framework is divided into training and testing phases. First, we split the dataset into two training and testing parts. While all training samples are composed of normal observations, testing samples include both normal and abnormal data. The principal assumption is to feed normal observations into the model during training to motivate it to learn the normal data pattern. For this purpose, we apply an encoder on training samples to convert data to a latent feature space in the bottleneck layer. Then, a decoder network is applied to the latent data to reconstruct the original input. We then utilize the mean squared logarithmic error (MSLE) to define the error between the original input data and the reconstructed version. Equation 5.7 illustrates the MSLE loss function, where X and $\hat{X}$ represent the original and reconstructed input data, respectively. In this equation, N demonstrates the number of input data samples.

$$\text{MSLE}(X, \hat{X}) = \frac{1}{N} \sum_{i=1}^N \left(\log(X_i + 1) - \log(\hat{X}_i + 1) \right)^2 \quad (5.7)$$

Next, we define a threshold value based on the MSLE loss function to differentiate between normal and abnormal input samples. We obtain the threshold value during the training phase when we feed only normal samples to the model; therefore, the threshold indicates the maximum margin between the normal original and reconstructed input data. Equation 5.8 depicts the threshold formula, which consists of two terms. The first and second terms represent the mean and standard variation of the MSLE loss function, respectively.

$$\text{Threshold} = \underbrace{\frac{1}{N} \sum_{i=1}^N \text{MSLE}(X, \hat{X})}_{\text{Mean}} + \underbrace{\sqrt{\frac{\sum_{i=1}^N \left(l_i - \frac{1}{N} \sum_{j=1}^N \text{MSLE}(X, \hat{X}) \right)^2}{N}}}_{\text{Standard deviation}} \quad (5.8)$$

l_i in Equation 5.8 represents the MSLE loss value for the i -th sample, computed as:

$$l_i = \left(\log(X_i + 1) - \log(\hat{X}_i + 1) \right)^2$$

Next, we compute the MSLE loss between the original and reconstructed test samples in the testing phase. If the MSLE loss value of each test sample is higher than the threshold, the

Algorithm 4 Anomaly Detection

Input: Mixed data

Output: Normal and abnormal data

Training Phase:

- 1: Feed the Encoder with normal data: $\mu, \sigma^2 = \text{Encoder}(X_{\text{train}})$
- 2: Set the latent variable Z as $Z = \mu + \epsilon \cdot e^{\sigma^2/2}$, where $\epsilon \sim \mathcal{N}(0, 1)$
- 3: Feed the Decoder with Z : $\hat{X}_{\text{train}} = \text{Decoder}(Z)$
- 4: Compute $\text{MSLE}(X_{\text{train}}, \hat{X}_{\text{train}})$ based on Equation 5.7
- 5: Compute a threshold value based on Equation 5.8
- 6: Compute ∇L by using Equation 6:
- 7: $\theta, \phi \leftarrow \text{Adam}(\nabla L, \theta, \phi, \alpha)$

Testing Phase:

- 8: Feed the Encoder with sample i : $\mu_i, \sigma_i^2 = \text{Encoder}(X_{\text{test}_i})$
 - 9: Set a latent variable Z_{test_i} as $Z_{\text{test}_i} = \mu_i + \epsilon \cdot e^{\sigma_i^2/2}$
 - 10: Feed the Decoder with Z_{test_i} : $\hat{X}_{\text{test}_i} = \text{Decoder}(Z_{\text{test}_i})$
 - 11: Compute $\text{MSLE}(X_{\text{test}_i}, \hat{X}_{\text{test}_i})$ based on Equation 5.7
 - 12: **if** $\text{MSLE}(X_{\text{test}_i}, \hat{X}_{\text{test}_i}) > \text{threshold}$ **then**
 - 13: X_{test_i} is an anomaly.
 - 14: **else**
 - 15: X_{test_i} is normal.
-

observation is abnormal, and vice versa. Algorithm 4 shows the summary of the mentioned entire process systematically. ∇L is the gradient of a loss function that measures the rate of change of the loss function L concerning the model parameters θ . It is a vector that indicates the direction of the most rapid increase of the loss function. θ and ϕ are parameters (weights) of the model to be explored for optimal values that make loss function minimized. The hyperparameter α is responsible for changing model parameters after each model weight update, according to the estimated error. Technically, an appropriate selection of α has a significant impact on the convergence of the training process.

5.6 Experiment

In this paper, we seek to verify the performance of the energy over-consumption detection framework through the transmission settings and link quality parameters. The data is generated in a point-to-point data transmission between an IoT node and an edge gateway in a wireless sensor network. It is important to note that the design of our proposed framework is transmission-oriented. Therefore, we can simply scale up the concept to many transmissions between a node and multiple gateways.

5.6.1 Experiment Design

In this section, we describe the design of the experiment and discuss the result of the deep neural network models for detecting anomalies in the energy consumption of IoT devices. In practice, online anomaly detection frameworks could be designed as “continuously active” [281] and “active on-demand” [282]. Our proposed framework is designed to work on demand. In other words, we use this framework for diagnosis purposes. If the edge network requires energy troubleshooting, the framework starts to collect the communication arguments only for a bounded time. Data collection in the proposed framework follows the least intrusive technique which is called dynamic binary instrumentation. DBI (Dynamic Binary Instrumentation) [283] is known as a run-time executable modification technique that enables a tracing mechanism to collect detailed software/hardware granular data with the least perturbation [19] [284] [285] [286]. Tracing imposes an extremely low overhead and therefore it is unlikely to impact the energy consumption of nodes significantly [287] [288]. Since dynamic binary instrumentation is a run-time mechanism for collecting desired data, it does not impose any long-term overhead on the devices under experiment and the overhead of data collection could be safely lifted on the nodes immediately after the diagnosis ended. Since the troubleshooting procedure is designed to be performed in the limited time of diagnosis in our scenario, it is undeniable that the cost of our short-term analysis compensates for the cumulative over-consumption in the long run for IoT anomalous nodes.

According to equations 5.2 and 5.3, the energy consumption of an IoT node has an intrinsic relationship with all of the parameters such as T (Transmission time), P (Transmission power), d (Transmission distance), and RSSI. To evaluate the possibility of correlating energy consumption and other transmission parameters, a public dataset is used as a testbed for the experiment. Our proposed framework is expected to integrate the imbalanced attention for features and provide better performance metrics than traditional machine learning models and VAEs.

For the dataset, the influential features associated with the energy consumption of IoT devices are extracted by analyzing the features of experiments through a covariance matrix and linear regression. We verify a strong *Protocol-Energy* correlation as a first step. By Finding that *Protocol* is the most correlated parameter with energy in the dataset, we accordingly design a new loss function for our neural network (variational autoencoder) by adding the error between the *Protocol* value in the original and reconstructed output to guide the direction of gradient flow. According to the binary labeling strategy for anomaly detection scenarios, multiple baseline clustering algorithms have been applied to evaluate the efficiency of those models. The given scores for these models are calculated based on their clustering

estimations' accuracy. By investigating how performance metrics vary for most baseline clustering algorithms, attention is drawn to the key role of data labeling in anomaly detection. In our case, the data labeling strategy discussed in Section 5.5.1 is set based on the official datasheets of the available protocols under the experiment. Such an approach not only defines a realistic threshold for model training but also makes the final result comprehensible.

An activation function is the main part of the neural network design as it describes the transformation of the weighted sum of an input to an output from the node(s) in a network layer. We provide comparisons to demonstrate how configuration parameters are associated with the model's performance in neural networks. This section uses popular activation functions to conduct the comparison experiment. Sigmoid function as a non-linear activation function transforms the values to a range between 0 and 1. Secondly, as a symmetric activation function, the Tanh function carries out the transformation while the output range is constrained between -1 and 1. Regarding the last activation function, the ReLu, as a computationally efficient function, activates the neurons for which the output of linear transformation gets above zero and deactivates them otherwise.

In addition to the provided performance metrics, we evaluated the indirect complexity of the proposed algorithm by measuring the model execution time. It contains the training and testing time of the model that relatively represents how feasible the model implementation is considering other constraints imposed in certain scenarios.

5.6.2 Result and Discussion

By modifying and training a 2-layer VAE with normal energy data and testing it on a mix of normal and abnormal energy consumption observations, the performance metrics and execution time are compared to an unmodified 2-layer VAE in Fig. 5.5. This figure depicts the performance of a 2-layer VAE against the proposed 2-layer modified VAE model measured by accuracy, precision score, F1 score, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), training time, and testing time using three active functions. The performance metrics and execution time (training/testing time) computation are designed to get the advantage of multiple K from 1 to 5 for a K -fold setting. With Tanh as an activation function, the proposed 2-layer modified variational autoencoder performs best (Fig. 5.5a-5.5e). While the performance of 2-layer VAEs for ReLu and Sigmoid degrades compared to the proposed modified version, the anomaly detection capability of 2-layer VAE for Tanh improves in its modified form. Meanwhile, none of the proposed 2-layer modified VAEs could gain any better performance than that of the 2-layer VAE in accuracy, precision score, F1 score, MAE, RMSE levels, and execution time. Based on Fig. 5.5f and 5.5g, the

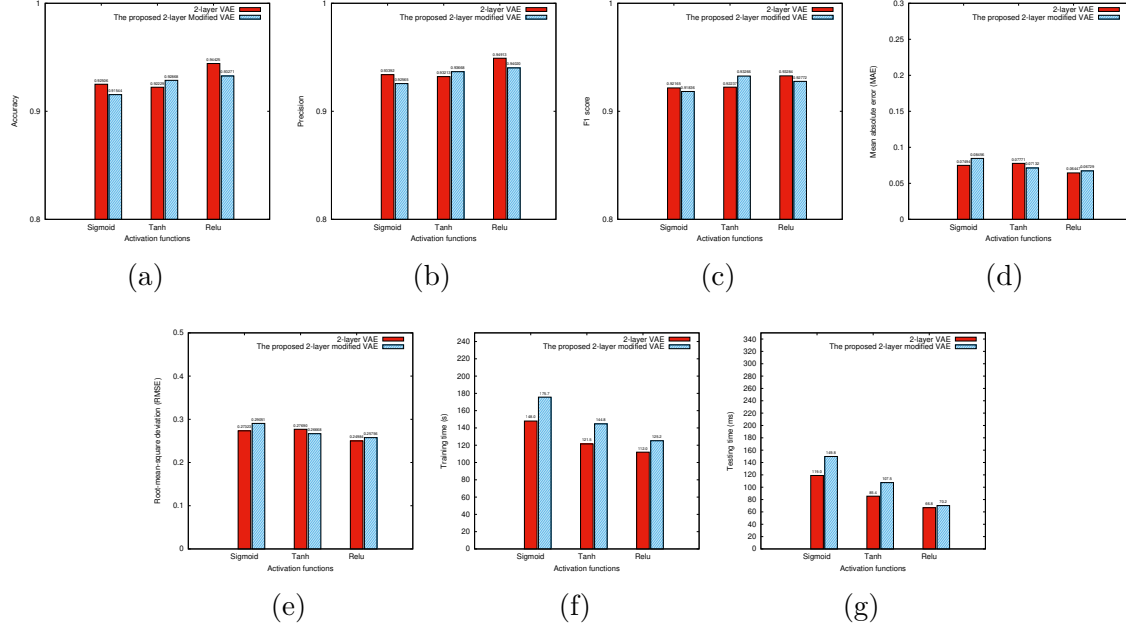


Figure 5.5 A comparison between performance metrics and execution time for 2-layer VAE and the proposed 2-layer modified VAE using different activation functions: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time

execution time for each activation function increases in the modified form. It makes sense as the newly designed loss function has an additive term that comes with additional time for processing and computation (see Equation 5.6). These figures not only confirm that none of the performance metrics for the proposed modified 2-layer VAE improved but also indicate an 11 and 5 percent increase in training and testing time (ReLU) compared to that of the 2-layer VAE (ReLU). Besides, the ReLU activation function takes the least training and testing time of all. Therefore, considering all the activation functions, the 2-layer VAE using the ReLU activation function outperforms the other configurations.

To better evaluate the model performance, a different number of hidden layers is inserted as the depth of our neural network. Fig. 5.6 represents performance metrics, execution time, and configuration for a 3-layer VAE and the proposed 3-layer modified VAE to investigate the effect of the number of hidden layers on the efficiency of our model. By comparing the performance among the three activation functions for different K parameters, the proposed 3-layer modified VAE using the ReLU activation function accomplished the best. It achieved a higher accuracy, precision score, and F1 score while obtaining lower MAE and RMSE values at the cost of 12 and 2 percent increase for the training and testing time, respectively. Fig. 5.6f and 5.6g illustrate how temporally efficient the ReLU activation function is compared to

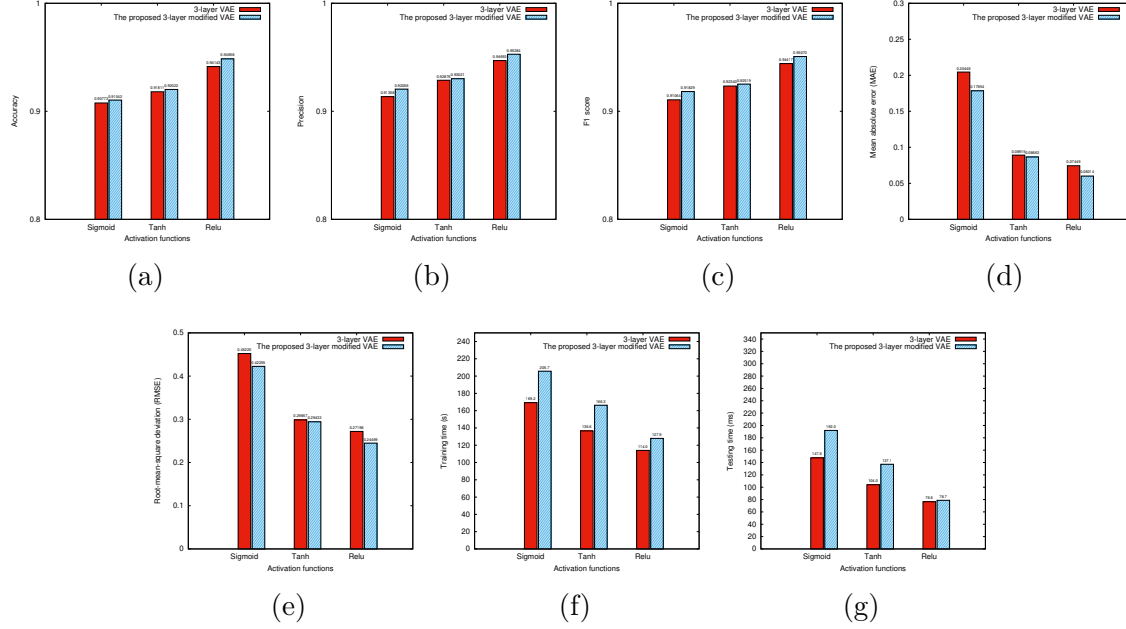


Figure 5.6 A comparison between performance metrics and execution time for 3-layer VAE and the proposed 3-layer modified VAE using different activation functions: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time

other counterparts both in the modified and unmodified 3-layer VAE. One of the advantages of ReLU in obtaining the best execution time lies in sparsity, as a majority of the neurons are inactivated in a layer for a given input. The fewer number of neurons staying active, the lower the computational load and time would be. By keeping all the neurons active in the other activation functions, Sigmoid and Tanh are computationally expensive. In the case of Sigmoid, the execution time is larger than that of others. This activation function contains an exponential term in its definition. This term requires a series expansion and a complex numerical approximation that makes its computation costly, in particular for large values of input. Sigmoid also involves a division operation which adds extra overhead to the computational cost. Moreover, the symmetry in the Sigmoid function requires the computation of both negative and positive values of the input, thus doubling the computational cost compared to non-symmetric activation functions.

At this point, comparing the performance metrics and execution time of a 2-layer VAE using ReLU against the proposed 3-layer modified VAE using ReLU activation function results in a comprehensive analysis of our investigation. Considering the three activation functions, Fig. 5.7 shows that the highest accuracy, precision, and F1 score in parallel with the lowest MAE and RMSE belong to the proposed 3-layer modified VAE. The highest performance metrics

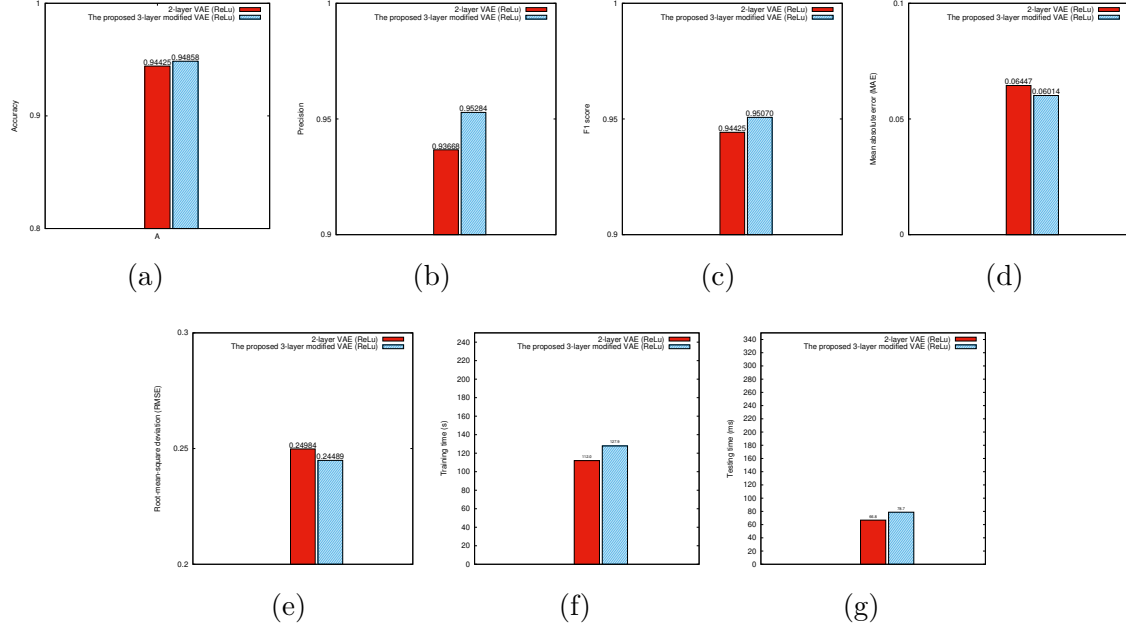


Figure 5.7 A comparison between performance metrics for 2-layer VAE and the proposed 3-layer modified VAE using ReLu activation function: (a) Accuracy; (b) Precision score; (c) F1 score; (d) MAE; (e) RMSE; (f) Training time; and (g) Testing time

level is obtained with three hidden layers for the ReLu activation function at the cost of 14 and 17 percent increase for the training and testing time, respectively. As already discussed, our diagnostic framework performs in an "active on-demand" mode for a short duration in practice. Therefore, such overhead for a troubleshooting framework can be tolerated by the network compared to the driven benefits of conserving excessive amounts of energy consumption in a lifetime of anomalous nodes.

Although 1 or 2 percentages of improvement are negligible for performance metrics with values lower than 90% level, it becomes remarkably challenging to make further improvements as a model becomes more accurate than 90%. In other words, as the 2-layer VAE has already obtained high performance (above 90%), even small gains for the proposed 3-layer modified VAE can be significantly important to achieve because we are operating in a region where errors are inherently hard to reduce.

A comprehensive overview of the performance metrics of the several baseline clustering models and the proposed 3-layer modified VAE is presented in Table 5.4. According to the comparison, the performance of our proposed 3-layer modified VAE is improved dramatically not only for accuracy but also for precision score and F1 score against the baseline models. To evaluate the efficiency of the normalized 3-layer modified VAE model, we concentrate on the

Table 5.4 Comparison between the baseline clustering algorithms and the proposed modified VAE

Models	Accuracy	Precision Score	F1 Score
Kmeans	0.706	0.857	0.774
Normalized Kmeans	0.640	0.829	0.722
Mean Shift	0.594	0.858	0.702
Normalized Mean Shift	0.646	0.847	0.733
Spectral Clustering	0.552	0.872	0.676
Normalized Spectral Clustering	0.569	0.845	0.680
Birch	0.621	0.857	0.720
Normalized Birch	0.640	0.829	0.722
DBSCAN	0.150	0.860	0.260
Normalized DBSCAN	0.920	0.840	0.880
Mixture Model	0.713	0.857	0.778
Normalized Mixture Model	0.644	0.848	0.732
BGMM	0.477	0.857	0.613
Normalized BGMM	0.640	0.829	0.722
Agglomerative Clustering	0.621	0.857	0.720
Normalized Agglomerative Clustering	0.640	0.829	0.722
Normalized 3-layer Modified VAE using ReLU activation function	0.948	0.952	0.950

performance metrics in Table 5.4 and Fig. 5.7. The output of Sigmoid and Tanh activation functions is saturated for large positive and negative inputs. Such poor performance results in the vanishing gradient problem. This is a common problem in creating neural networks [289]. It originates from flattening the derivative of the activation function. Therefore, there will be almost no new updates on parameters in the process of model training which leads to poor convergence. Similar to the pruning mechanisms, the Rectified Linear Unit (ReLU) can invalidate the statistically nonessential features by putting them to zero. In other words, the linear structure of ReLU (non-saturating feature) boosts model training efficiency as the output of the partial derivative of the loss function will be mapped to a binary result (0 or 1) and consequently stops the gradient from vanishing.

We find out that identifying the dominant features and redesigning the loss function accordingly, gives the network a deeper understanding of distinguishing the normal and abnormal data patterns. Our investigation manifests that choosing the proposed 3-layer modified VAE using the ReLU activation function improves the robustness of the model according to the obtained values for accuracy, precision score, F1 score, MAE, RMSE, training time, and testing time.

Eventually, our investigation proves that unfit adjustment of link parameters in data transmission, such as the *Protocol* plays a key indicator of high energy consumption between an IoT device and the edge elements. Based on the protocol type, intuitive assumptions could

be devised for the reason behind the over-consumption of resource-constrained devices. Security protocols, those which require significant cryptographic processing with costly cipher parameters, increase the energy consumed by the battery-powered nodes. For communication protocols, those that are susceptible to collisions (large packet loss) made by simultaneous flooding of tag responses are energy-exhaustive mechanisms. For routing protocols, those determining routes statically and relying on periodic advertisements generated by routers consume significantly more energy than traditional ones. It should be declared that the on-line frameworks equipped with energy-aware protocols should also take care of a limited time window for service provisioning without compromising QoS to avoid operation interruption.

According to the provided statistics, we can verify that the delivered anomaly detection framework could be appointed as a validated approach for energy over-consumption analysis. Further, the energy stabilization feedback enables us to alter the transmission settings which determine the alternative with the least energy footprint, under any given operating environment. It may toggle the node configuration towards a greener setting. In such settings, routing protocols, security protocols, and communication protocols will meet energy-awareness standards, while network responsiveness ensures the necessary QoS levels and reliability.

5.7 Conclusion

Artificial-intelligence-integrated Internet of Things (AIoT) is an emerging discipline that integrates AI analysis and approaches into the Internet of Things field of interest. By nature, applications running in the AIoT domain are computation-intensive. They require a high level of real-time processing to achieve decisions made by ML (Machine Learning), DL (Deep learning), and data analysis operations. The high number of constraints imposed on an AIoT scenario, such as limited energy harvesting potential and demanding energy consumption, makes achieving a better quality of service (QoS) challenging for applications. Since ignoring the energy anomalies can easily result in battery depletion and network failure eventually, our work focuses on the early detection of the hotspots communicating with the edge gateway according to their link quality and communication configurations. Accordingly, this paper proposes a deep learning approach and a binary labeling strategy for identifying energy-intensive operations. Such a self-configuring scheme uses a 3-layer modified VAE to detect anomalies for peer-to-peer connections using link quality and communication features. Additionally, several unsupervised clustering algorithms are presented, validated, and compared with the performance metrics of our model, including the Mixture Model, Bayesian GMM, Agglomerative Clustering, Birch, and Modified Variational Autoencoder. By analyzing these performance metrics, the best efficiency in anomaly detection for the energy consumption of

IoT nodes in wireless sensor networks is achieved. As shown earlier, the proposed 3-layer modified VAE model reached an outstanding performance of 94.8% accuracy, 95.2% precision score, 95.0% F1 score, 0.06014 MAE, and 0.24489 RMSE for anomalous energy detection.

As a primary conclusion, we find out that an improper configuration of IoT nodes (e.g., communication protocol selection for data transmission) may considerably increase the energy consumption of nodes. Such settings subsequently stop the necessary services from executing satisfactorily and fulfilling expectations. Establishing energy-sustainable networks on edge requires a real-time monitoring system to discover the energy-hungry nodes. Most existing AI-based solutions propose offline frameworks using supervised or intrusive approaches. The supervised solutions on low-power IoT nodes are mostly invalid due to the infeasible energy labeling in real-time. High-intrusive approaches modify the native software or hardware to monitor the energy behavior of different components of the nodes. While energy sampling methods can capture the energy peaks in predetermined periods, they cannot target contextual anomalies. The major reason for such deep suffering lies in insufficient precision for complex sensing activities/computation. Moreover, providing thousands of sensors with external energy monitoring hardware is not practical, especially if flexibility, price, and physical size come into the scene. Hence, for early energy anomaly analysis and transmission parameters adjustment, we employ a deep learning approach to detect abnormal energy data indirectly without supervision.

For future work, using time-series data for transmission parameters and link quality data not only helps us identify patterns but also creates the opportunity to detect contextual anomalies and predict future patterns. Using feature interaction learning, we can also capture predictive information (interactions between features) to improve the modified loss function. Moreover, besides having the external data regarding data communication, collecting the internal data such as CPU/GPU/RAM utilization and NIC I/O rate gives us more data to train the model. We hypothesize that such a model not only detects the anomalous nodes in an underconsumption state but can minimize the number of false positive anomalies as it is contextually bound to the internal behavior of the system rather than just a constant threshold setting.

CHAPTER 6 ARTICLE 3: "ANOMALOUS ENERGY DETECTION FOR RESOURCE-CONSTRAINED EMBEDDED SYSTEMS USING TRACING DATA ANALYSIS"

Authors Ahmad Shahnejat Bushehri, Samira Keivanpour, Muhammad Azam, Gabriela Nicolescu

Submitted on October 10, 2021 and accepted on October 28, 2021 in IEEE International Conference on Electrical, Computer and Energy Technologies (ICECET), DOI: 10.1109/ICECET52533.2021.9698794

Abstract Since the multi-sensor embedded systems are spread all over urban and suburban areas, there should be a real-time performance analysis framework to monitor the energy consumption of sensors and services. To this end, applying artificial intelligence to model the behavior can improve operational efficiency and over-consumption detection. Most of the energy over-consumption detection approaches make use of external devices such as smart plugs or designed for smart buildings rather than computing devices. This paper proposes a new architecture in which we extract real-time execution traces after identifying the suspicious running applications using the dynamic binary instrumentation technique. The output of this step is fed to a data analytic framework to detect any behavioral anomalies such as over-threshold energy consumption. Finally, depending on the decision made by the anomaly detection step, some sensors might be switched to standby mode to meet the essential expectations for other sensors or services. Since we have a resource-constrained embedded system and a real-time decision-making constraint on sensor control, the amount of computation that could be done in real-time varies. Accordingly, the desired performance may dictate how, when, and where the processing operations might occur (on the embedded system or a remote server).

This work is mainly focused on anomaly detection operations using tracing data analysis. The experimental results show that the over-consumption detection model achieves a high level of accuracy metrics for classifying abnormal consumption behavior.

6.1 Introduction

As resource-constrained embedded systems get more complex, they are not susceptible to resolving unexpected performance anomalies any longer [155]. Since there are several reasons

for system performance dropping such as service resource overloading, it would be a complicated task to inspect and detect the system execution flaws manually. One of the solutions to tackle this limitation could be a behavioral system analysis to inspect the performance of the running processes on a machine automatically [27]. By monitoring the running service responses to the system actions, the behavioral system analysis approaches can diagnose the potential anomalies. One of these approaches is the analysis of tracing events using dynamic binary instrumentation.

In this paper, we propose a framework with a focus on an anomaly detection model to identify over-consumption system states by using the number of various system calls generated in each time window. In this work, we validate the performance of the proposed framework by applying anomaly detection on three separate public datasets¹ provided by authors in [290]. As the detailed energy information of a running device can reveal its internal status at each instant of time, it can also provide a rough prediction of the energy state of the device shortly and help authorities to indicate the best course of action in monitoring the performance of the system.

To the best of our knowledge, although the tracing data analysis has been considered in the performance evaluation of systems, using system calls for anomaly detection in power consumption has not yet been addressed in embedded systems.

The contributions of this paper are as follows:

- Propose a new framework for anomaly detection in power consumption of energy-restricted embedded systems.
- Propose a learning solution through classification for anomaly detection using the supervised tree-based and Logistic regression algorithms
- The proposed approach is evaluated by using three public datasets. The results show that the proposed approach achieves a high level of accuracy metrics.
- The proposed approach is also implemented with some other classification algorithms to observe the effectiveness of our proposed solution

The rest of the paper is organized as follows. In Section 6.2 dynamic binary instrumentation (DBI) and tracing are introduced. Our proposed methodology for the classification framework and data labeling approach is presented in Section 6.3. Section 6.4 provides the experimental framework, results of experiments, and discussion. Eventually, Section 6.5 is dedicated to conclusions and future work.

¹<https://github.com/greentrace/green-dataset>

6.2 Literature Review

6.2.1 Dynamic Instrumentation

DBI (Dynamic Binary Instrumentation) is known as a run-time executable modification technique that enables tracing to inspect the behavior of an executable. In dynamic instrumentation, a probe has been injected into the user-defined location(s) at run-time to transfer the control into the desired location [144] and toggling among inserted probes to audit the behavior of the program at those analytic points [145].

Run-time binary modification frameworks including DTrace [291] play an effective role in debugging purposes. They are used in a variety of other areas such as cache simulators [292], and JIT compilers [293]. From embedded firmware instrumentation [149] and on-demand scalable testing frameworks [150] [24] to trap-and-emulate hardware VMM of x86 [151], and malicious software detection [152], DBI techniques have been used in practice extensively.

6.2.2 Tracing

Event tracing, a facility between a process and the Linux kernel, enables the user to record the kernel activities or application-defined events on a log file. Since embedded systems are mostly implemented on multitasking RTOS (Real-Time Operating System), their performance also depends on response time. The timing must not exceed a certain threshold. To meet the deadline, developers use tracing. Tracing as an operative technique discovers the state of execution of the target process and analyzes the performance of elaborate components. A trace of a program includes a sequence of events written by probes. The tracing events could be served from a log file or even in real-time to debug a micro-service or to troubleshoot the performance issues that may occur in an application [153]. As system calls are part of tracing events, through this paper, we propose to use them, to spot abnormal behavior of energy consumption in embedded systems such as edge devices. However, coping with data collection/management becomes uncontrollable [164]. The time-delay embedding approach is not able to scale well as tens of thousands of system calls per second are generated just by a single process [165] [155]. As an answer to this problem, other studies, such as [166] [155] divide the chain of the system calls into shorter pieces and then apply feature extraction over a fixed time window. A dynamic length sliding window method is used by Eskin et al. [169] that is independent of the context of the system call streams. Detecting anomalies in Liu's works [154] [155] is done by extracting the histogram of different system call types. Employing a bag of system calls, the authors in [156] detect intrusion and describe the misuse

with the standard machine learning techniques. Canzanese et al. [157] utilize the number of short streams of system calls in a fixed time window to detect malicious processes. For process monitoring, the authors in [158] apply a deep LSTM-based architecture on a stream of system call count vectors. Selecting the best features for the classification of malware system call streams in [159] is done by constructing a neural network based on the convolutional and recurrent network. Tandon et al. [160] [161] [162] use multiple values of multiple system call arguments to improve their rule-based method. The contribution of authors in [163] is considering the system call name, the CPU cycles count, and the distribution of characters in the values of the arguments for anomaly detection in time- and event-driven processes using kernel traces. In [164], two neural networks (LSTM) are applied to textual attributes and response time in logs and traces to detect anomalies in cloud services.

The emergence of edge computing in the Internet of Things (IoT), introduced edge devices (as embedded systems) that are located closer to the sensors/users. These embedded systems are involved in the decision-making process and can reduce communication latency, data migration, and bandwidth consumption [11]. To make a stable environment-friendly and low-power design for these real-time resource-constrained embedded systems, energy consumption should be cut significantly to enable the services to last longer. There are several energy-aware empirical, reasoning, and reconfiguration approaches that study various issues at different code, design, and run-time levels. Recent comprehensive empirical studies performed at the multi-stage software life cycle [39] [40] showed that it is almost impossible to minimize the energy consumption of software products for developers due to the lack of software energy optimization knowledge. Accordingly, they do not attempt to improve its efficiency, since most of them do not have a clear estimation of how large the amount of consumed energy would be for developed applications. Moreover, there are still several incorrect interpretations held even by skilled professionals regarding the principles of energy consumption mitigation [40]. Recent research reveals that the limitations of tools in the green computing area at both the code level and software architecture level may prevent us from reaching the optimization point significantly [41]. Pinpointing where the energy hotspots are, is a factor that has been used by several experimental strategies to discover the pieces of code affecting the overall energy footprint of a program [42] [43] [44]. The goal is to make the most effective use of energy spent by applications by looking over the code. They mostly aim at identifying the energy profile for various energy-consuming concerns [45]. The authors in [46] and [47] displayed that a slight manipulation in the energy model at the design level would influence the energy consumption for the more efficient one. Having multiple design solutions and comparing the energy consumed by each of them is considered a challenging part while seeking compensation for other quality factors such as efficiency and productivity. There are also

other applicable techniques addressing the energy-oriented design patterns [42]. The studies in [48] [49] show how model-based energy efficiency analysis affects various consumption characteristics of software architecture. Organizing these models helps us realize the major connection between all those design patterns and the consumed energy in practice. Apart from the code level, method level [50], and architectural level power analysis, other approaches keep track of the energy consumption during run-time [51] or setting up profiles of power usage at each instant of time dynamically [52].

6.3 Methodology

In this work, we consider a performance analysis approach by recording the stream of system calls to monitor and improve the energy efficiency of the battery-powered embedded systems with a focus on the anomaly detection phase of the whole process. First, by inserting probes in the running applications through DBI, a real-time execution trace is extracted from a system. Tracing generates fine-grained text-based labels in which the footprint of service execution and the corresponding timing data are available. The detailed data contains the reciprocal system requests exchanged among service layers regarding CPU, GPU, RAM, disc, and other hardware components of the system that are known as events. Then, a new context-aware energy-saving framework would apply data analytics to detect any behavioral anomalies for over-threshold power consumption by sensors. Finally, depending on the decision made by anomaly detection, some sensors might be switched to standby mode to meet the essential expectations for other sensors or services. Since we have a resource-constrained embedded system and a real-time decision-making constraint on sensor control here, the amount of computation that could be done in real-time varies a lot. Accordingly, the desired performance may dictate how, when, and where the processing operations might occur, on the embedded system or a remote server.

An overview of the proposed framework is presented in Fig. 6.1. It illustrates the main steps to conduct an energy-aware model for practical monitoring of electrical power consumption by making use of tracing data analysis. In our architecture, we have three consecutive steps to reach a stable energy analysis of an embedded system: Tracing, Anomaly detection, and stabilization feedback. The core of this work is to present a proof of concept for realizing the second step as the identification of anomalous energy consumption of the battery-powered embedded systems. To this end, experiments are conducted on three sets of publicly available datasets. Each dataset contains the system calls trace logs that are converted to short sequences using a sliding window. Then, datasets are arranged by the frequency of individual system calls and their corresponding average power consumption in each sliding window.

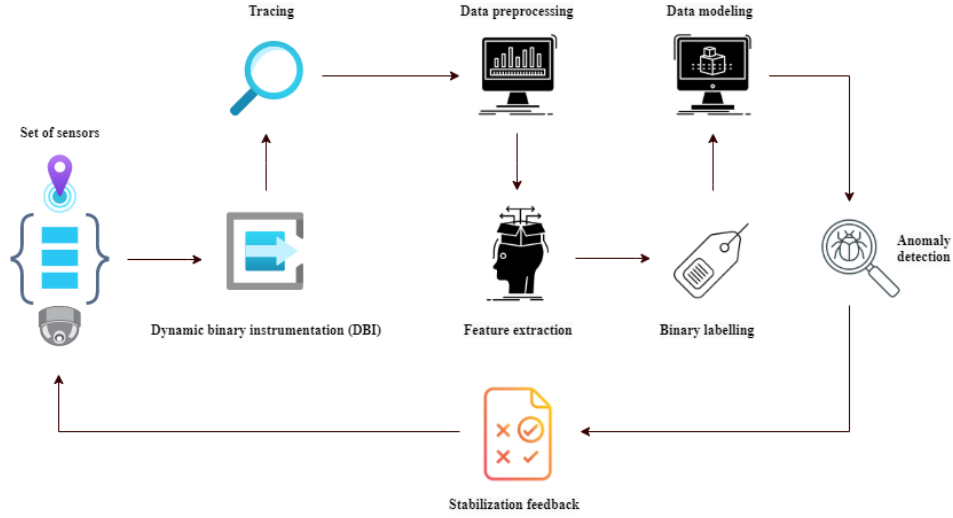


Figure 6.1 The complete architecture for the proposed anomaly detection framework

Finally, a supervised machine learning-based approach is applied to discriminate between normal and anomalous system call/power consumption pairs.

We have a real-time decision-making constraint here. The amount of computations that could be done in real-time varies depending on whether information processing occurs on the embedded system or a remote server. One way of modifying the scale of the computation is to change the level at which we are working (i.e., system call level, function call level, the whole request level, etc.). Knowing that the more information (lower level) comes, the more informed decisions would be made that result in better performance. If we have a real-time decision-making constraint, the desired performance may dictate the data level and the required resources in the end.

6.3.1 Data Labeling

The features of data stored in the Greentrace dataset (created by authors in [290]) are composed of average power consumption in a time window and the number of system call invocations in the same period. Along with the current research for anomaly detection, we are proposing a supervised machine learning approach that requires the annotated data for further actions. Since we are dealing with detecting anomalies in data, a binary labeling approach has been chosen to distinguish between normal and abnormal power consumption. In this binary approach, a label is created based on the labeling strategy presented in Table 6.1 to discover the anomalous and normal behavior of energy consumption. By empirically setting a threshold on the average power consumption, we can categorize the data into normal

Table 6.1 Labeling strategy for three datasets

Classes	Class labeling strategy
Dataset 1	
Class A (Abnormal)	<i>if POWER > 19.96856</i>
Class B (Normal)	<i>if POWER ≤ 19.96856</i>
Dataset 2	
Class A (Abnormal)	<i>if POWER > 20.67112</i>
Class B (Normal)	<i>if POWER ≤ 20.67112</i>
Dataset 3	
Class A (Abnormal)	<i>if POWER > 21.80403</i>
Class B (Normal)	<i>if POWER ≤ 21.80403</i>

and anomalous classes concerning power consumption as shown in Table 6.1. In this way, we can validate the effectiveness of the development of the anomaly detection model in our proposed framework considering the existence of normal and abnormal consumption data.

6.3.2 Proposed Classification Framework

Figure 6.1 demonstrates the complete architecture for the proposed over-consumption detection framework considering tracing data. This architecture acquires a supervised machine-learning algorithm for the classification of power consumption. We have the average power consumption as the first feature followed by the frequency of system call invocations in a time window in which the power consumption has been gauged. The binary class labels are introduced as the presence or absence of anomalous power consumption as stated in the previous section.

We have divided the data into training and testing sets and applied multiple supervised algorithms for model training. The set of applied algorithms on dataset1, dataset2, and dataset3 is listed in Tables 6.4, 6.5, and 6.6 respectively. We evaluate the performance of those algorithms in the testing step by multiple performance metrics such as Accuracy, F1 Score, Precision Score, and Recall Score. Accordingly, the eligible model for predicting the real-time consumption data is the one that benefits from the highest performance metrics in the tables 6.4, 6.5, and 6.6. The result of this step could provide us with an insightful overview of the system behavior and enable the stabilization feedback loop to recognize anomalous trace patterns in advance and react more smartly. The responsibility of the feedback loop is to reconfigure the sensors based on the result given from the anomaly detection step and switch sensors to sleep mode to meet the essential expectations for other sensors. Overall, by

Table 6.2 Structure of datasets

POWER	getsocketname	send	...	getegid32	readlink
21.96170492	4	16	...	2	45
22.09574430	3	20	...	10	44
⋮	⋮	⋮	⋮	⋮	⋮
20.38008439	3	12	...	11	42

taking advantage of the outcome of the anomaly detection phase, the stabilization feedback loop promotes long-term system efficiency. Accordingly, the system consumes sufficiently low energy to improve the system lifetime, especially in the case of fast battery depletion.

6.4 Experiments and Results

6.4.1 Experimental Framework

In this step, we show how the proposed anomaly detection model can be used in our framework for energy consumption data in practice. We make use of three public datasets generated by authors in [290] consisting of tracing data for different versions of applications versus the corresponding average energy consumption under multiple test cases. These datasets are the result of studying the *gedit* (An open-source editor) and the *mpg123* (an open-source player). Different versions of both *gedit* and *mpg123* have been built for generating these three datasets. Besides, data collection has also been done by developing two test cases. Every dataset is provided with the collected data directly from each test case. As it is shown in Table 6.2, datasets contain the average power consumption as the first feature column while the rest of the columns include the number of a specific system call invocation in the period in which the consumption is measured. Rows of datasets represent the detailed data of various versions of each of the two applications. Some of the highly variable features of all three datasets are arranged in Table 6.3.

In our framework, we create a labeling scenario for data to distinguish anomalous and normal behavior. This labeling is based on defining an empirical threshold (as a proof of concept) as shown in Table 6.1. Once the labels are created, the data is applied to the presented framework for modeling and examining the effectiveness of the proposed approach.

6.4.2 Results of Experiments and Discussion

In this section, we provide the result (as a proof of concept) for the classification of tracing data based on the associated power consumption. We also show that machine learning algorithms can discover anomalous power consumption using tracing data analysis. In addition,

Table 6.3 The most highly variable features of all datasets and the associated descriptions

System call	Description
poll	It waits for one of a set of file descriptors to be available for I/O operations.
futex	It facilitates waiting and waking up at a particular address for a program.
listen	It labels a connection-mode socket, as accepting connections.
recvmsg	It receives a message and its header from a connected or unconnected socket.
recv	It receives data on a connected socket and stores it in a buffer.
writv	It always writes a complete area into multiple buffers before proceeding to the next.
close	It closes an open file descriptor such that it does not refer to any file anymore.
mprotect	It modifies permissions on any mapped memory region of a process.
stat64	It gets file status and returns the associated information in a buffer.
read	It attempts to read data from a file descriptor and stores it in a buffer provided by the calling process.
mmap2	It establishes a mapping between an address of process space to another memory.

we present our results in this section.

Research question 1:

How applicable is the categorization of anomalies and normal trace data extracted from an embedded device with a high performance based on the level of power consumption?

Tables 6.4, 6.5, and 6.6 are provided to give a comprehensive overview of the performance of the anomaly detection models by calculating how accurate their estimations are. Based on binary labeling, multiple algorithms have been deployed to assess the efficiency of power consumption classification. In these tables, the classification results for each dataset using binary labeling are given. We aim to verify the effectiveness of the power over-consumption detection by analyzing the tracing data (system calls) generated using the dynamic binary instrumentation process on the running applications. By detecting a pattern for system call frequency in each time window, the corresponding power consumption is estimated by employing the most stable classification algorithm among all. This model is designated through the testing phase by evaluating the classification performance metrics such as Accuracy, F1 Score, Precision Score, and Recall Score. As you can see in these tables, we achieved the accuracy of 95%, 93%, and 92% for those three datasets. In the same order, the F1 score reaches 95%, 94%, and 92%. The next performance metric (precision) gains 96%, 95%, and 93% scores for the respective datasets. The recall score is considered the last performance metric with values of 95%, 93%, and 92%. The numerical outputs for three datasets are presented in Tables 6.4, 6.5, and 6.4. They show performance for most of the presented algorithms in this article. Consequently, using system calls as fine-grained internal indicators of interactions between a process of an application and the kernel, helps us to dig more into

finding its correlation with power over-consumption.

Research question 2:

How applicable is the interpretation of the result of classification algorithms in practice based on the nature of granular tracing data?

We choose two out of three datasets to generate heat maps to examine the correlation among different features of datasets. This correlation helps us represent the importance of the efficient feature selection step for the pre-processing of the modeling stage. As the first intuitive example, the *poll*, and *futex* system calls (listed in Table 6.3) have the highest correlation with power consumption as it is easy to grasp on the heat map shown in Figure 6.2. The *poll* system call lets the process wait for occurring an event and resumes the process execution once the event happens. The *futex* system call allocates a method to stay waiting until a particular condition comes true. These two system calls are strongly correlated with time. Secondly, based on the heatmap illustrated in Figure 6.3, the *recvmsg* and *listen* systems calls show a high correlation with power consumption in the second dataset. The *recvmsg* is responsible for receiving and storing messages on a socket. The *listen* system call keeps listening on a socket to accept incoming connection requests. It accepts all the connections to the point that nothing is left in the queue. Both the *recvmsg* and *listen* systems calls are the players of input/output operation that are explicitly considered as one of the most time-intensive tasks in performance evaluation. In other words, the more time-intensive system calls detected within a time window, the higher power consumption is likely to be detected.

We gain highest accuracy of 87% for SVM, 93% for GNB, 92% for BNB, 95% for DT, 93% for MLP, 93% for LDA, 92% for QDA, 92% for KNN, 95% for Log-Reg, 91% for RF, 93% for EntraTree, 95% for AdaBoost, and 95% for Grad_Tree_Boost.

By reviewing the performance metrics of all the aforementioned models in Tables 6.4, 6.5, and 6.6, it can be easily concluded that the presented anomaly detection technique including the binary classification could be considered as a trusted approach for energy consumption assessment in energy-restricted systems. By looking over the variation of accuracy metrics for algorithms in three datasets, it is realized how crucial the role of data labeling in anomaly detection is. Thus, reaching sufficiently accurate data labels is also expected by precisely defining the actual amount of threshold for model training. However, defining a certain fine-tuned energy consumption threshold for such systems is not an easy task. Taking all the results into account, it is preferred to follow the logistic regression and tree-based algorithms for data classification in our framework.

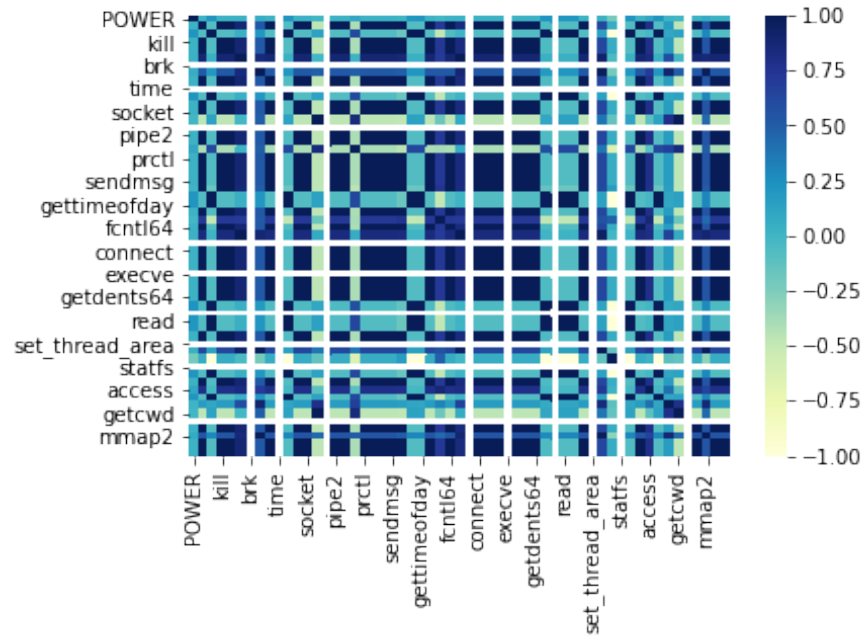


Figure 6.2 Heat map- The correlation between features in the first dataset

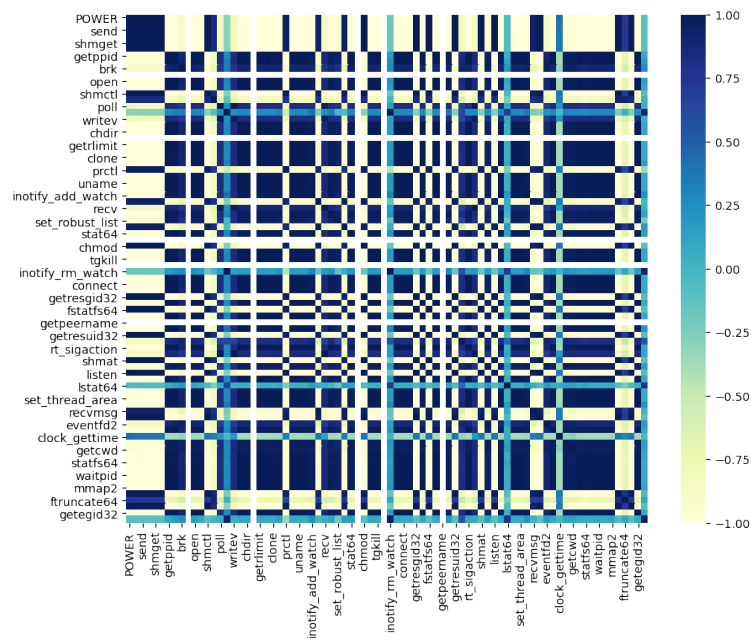


Figure 6.3 Heat map- The correlation between features in the second dataset

Table 6.4 Experimental results for the first dataset

Model	Accuracy	F1 Score	Precision Score	Recall Score
SVM	0.8695	0.8088	0.7561	0.8695
GNB	0.8260	0.8432	0.8734	0.8260
BNB	0.8260	0.8260	0.8260	0.8260
DT	0.9565	0.9590	0.9673	0.9565
MLP	0.9130	0.9216	0.9478	0.9130
LDA	0.6521	0.6864	0.7246	0.6521
QDA	0.8695	0.8088	0.7561	0.8695
KNN	0.8695	0.8088	0.7561	0.8695
Log-Reg	0.9565	0.9590	0.9673	0.9565
RF	0.9130	0.8933	0.9209	0.9130
EtraTree	0.8695	0.8088	0.7561	0.8695
AdaBoost	0.9560	0.9590	0.9673	0.9565
Grad_Tree_Boost	0.9565	0.9590	0.9673	0.9565

Table 6.5 Experimental results for the second dataset

Model	Accuracy	F1 Score	Precision Score	Recall Score
SVM	0.8461	0.8461	0.8461	0.8461
GNB	0.9230	0.9230	0.9340	0.9230
BNB	0.9230	0.9221	0.9326	0.9230
DT	0.9230	0.9230	0.9340	0.9230
MLP	0.5384	0.3769	0.2899	0.5384
LDA	0.9230	0.9230	0.9340	0.9230
QDA	0.9230	0.9230	0.9340	0.9230
KNN	0.9230	0.9230	0.9340	0.9230
Log-Reg	0.9230	0.9230	0.9340	0.9230
RF	0.7692	0.7664	0.7730	0.7692
EtraTree	0.8461	0.8461	0.8461	0.8461
AdaBoost	0.9230	0.9230	0.9340	0.9230
Grad_Tree_Boost	0.9230	0.9230	0.9340	0.9230

Table 6.6 Experimental results for the third dataset

Model	Accuracy	F1 Score	Precision Score	Recall Score
SVM	0.8750	0.8166	0.7656	0.8750
GNB	0.9375	0.9281	0.9416	0.9375
BNB	0.8125	0.7844	0.7583	0.8125
DT	0.9375	0.9281	0.9416	0.9375
MLP	0.9375	0.9281	0.9416	0.9375
LDA	0.9375	0.9425	0.9583	0.9375
QDA	0.8750	0.8750	0.8750	0.8750
KNN	0.8750	0.8166	0.7656	0.8750
Log-Reg	0.9375	0.9281	0.9416	0.9375
RF	0.8750	0.8166	0.7656	0.8750
EtraTree	0.9375	0.9281	0.9416	0.9375
AdaBoost	0.9375	0.9281	0.9416	0.9375
Grad_Tree_Boost	0.9375	0.9281	0.9416	0.9375

6.5 Conclusion and Future Work

For resource-constrained autonomous embedded systems, meeting the QoS metrics such as response time, delay, and electrical energy consumption is necessarily required. There are cases like power over-consumption that ignoring an anomaly, could fail a system to function

and may result in unwanted scenarios such as fast battery depletion. Hence, the main purpose of our work is to control the damage by identifying the anomalies that may eventually cause an outage by exceeding battery depletion and stopping other vital services from working properly.

To this end, we extract system calls from the tracing events that are generated by the dynamic binary instrumentation. Thereafter, we integrate an anomaly detection approach into the systems to keep an eye on the always-running services. We proposed a classification model based on binary labeling to differentiate the normal and abnormal power consumption data within every three datasets. Then we evaluate a highly effective over-threshold consumption detection framework through performance metrics shown in Tables 6.4, 6.5, and 6.6.

For future work, apart from the frequency of system call invocation in an interval, we can use the tracing event name, and its duration as well. Therefore, it is a reasonable assumption that collecting this information gives us more data to train the model. We hypothesize that the model decision would improve this way, and hopefully lead to a better decision. Considering the huge amount of system calls in a generated trace, we have no choice but to apply filters due to real-time data processing constraints and a high degree of data correlation between features. The high level of correlation between system calls (areas in dark blue) in Fig. 6.2 and 6.3 shows that most of the dataset features (system calls) are more linearly dependent on each other and hence have almost the same effect on the power consumption. One solution could be to consider just the system calls located on the critical path of application execution instead of the entire tracing events. The critical path is the longest in the Directed Acyclic Graph (DAG) of the program execution.

Building upon these ideas, expanding the scope to include user-space tracing presents another opportunity for further research. User-space tracing enables the monitoring of application-level events, such as function call traces and user-level execution states. This added granularity would not only enhance the detection of energy anomalies but also support a more detailed performance analysis by correlating application behaviors with energy consumption. Moreover, integrating user-space and kernel-space tracing could provide a comprehensive, cross-layer view of system performance, bridging the gap between application logic and system-level resource utilization.

CHAPTER 7 DISCUSSION

In this chapter, we revisit the research objectives we set earlier and evaluate how well the work presented in Chapters 4-6 has helped us achieve them. Our main goal is to develop solutions for analyzing the energy performance of IoT elements used in edge networks. From this primary aim, we have specific objectives focusing on data collection, energy anomaly detection for SDs using internal tracing, and anomalous energy detection for edge devices using transmission data analysis.

Efficient data collection As the first objective, this thesis presents NOProbe, a user-space DBI framework developed to target significant challenges in tracing and data collection for the sake of energy optimization within multi-threaded systems. The design and implementation of NOProbe offer solutions for the limitations of traditional instrumentation techniques, particularly in the context of popular x86 architecture in production systems.

NOProbe introduces several innovative techniques to enhance DBI, leveraging low-overhead probe insertion and safe instruction patching mechanisms. NOProbe integrates a multi-strategy approach to insert fast probes at nearly any probe site, including in compiler-optimized or stripped binaries. This flexibility enables efficient performance analysis and debugging without imposing existing constraints made by other DBI strategies. The development of a light atomic patching algorithm for basic block overwriting in SMP environments is also a remarkable achievement. This solution ensures that the probe insertion procedure does not interfere with multi-threaded operations by avoiding a stop-the-world approach.

NOProbe implements the proposed strategies within the user-space tracing tool *uftrace* to demonstrate its practical application. NOProbe’s performance and effectiveness both in reducing overhead and maintaining accurate trace collection capabilities are evaluated against existing instrumentation methods. The results demonstrate that NOProbe outperforms other conventional DBI techniques by offering substantial improvements. NOProbe presents a distinct advantage over conventional DBI techniques, which frequently experience performance degradation or program execution interference, especially when handling variable-length instructions in the x86 architecture.

We employ NOProbe’s granular trace data collection for energy consumption evaluation in prospective steps. A model-based approach is then leveraged to provide insights into application energy behavior without relying on intrusive measurement hardware. This non-intrusive method aligns with the needs of non-disruptive service execution on resource-constrained IoT

devices.

While NOProbe represents solid progress in efficient data collection, there is always room for further exploration and improvement. Future work could focus on expanding the support for NOProbe’s applicability to other architectures and exploring additional optimizations to reduce overhead for probe insertion and execution further. Additionally, integrating new strategies for uninstrumentable instruction patterns with NOProbe could provide a more comprehensive solution for system monitoring and energy optimization. As systems continue to grow in complexity, scaling NOProbe to meet the requirements for instrumenting even larger and more dynamic applications will be crucial. Investigating how NOProbe can retain its performance benefits in such scenarios will be the most important aspect of future research of this kind. Future efforts could also explore alternative solutions for embedding *0xcc* bytecode in a branch-based approach in NOProbe to enhance its efficiency in performance monitoring.

Energy Anomaly Detection for Communication-Intensive Edge Elements As the second objective, we have addressed a significant challenge in the operational lifetime of edge networks: energy consumption, QoS in service provisioning, and the early detection of energy anomalies. Our work underscores the importance of proactive energy management to extend the lifespan of IoT devices and secure the reliability of the network. Traditional energy management techniques, such as DPM and DVFS, optimize energy usage at the system, architecture, or circuit levels. While effective in certain contexts, these techniques are not customized to address the unique challenges of IoT devices, particularly the energy footprint of data transmission operations. We have developed a novel framework for energy anomaly detection using a deep learning model in the form of a modified variational autoencoder (VAE). By identifying dominant features through linear regression and incorporating them into the loss function of the VAE, we have improved the performance metrics and reliability of the anomaly detection model.

Unlike factors easily found in theoretical energy consumption formulation such as power, time, and distance, monitoring the impact of link quality data and transmission settings (such as communication protocols) for network elements is vital for energy evaluation. By enabling early detection of energy anomalies, our framework allows IoT devices to adapt their transmission configurations dynamically, thereby prolonging battery life. Misconfigured parameters can lead to excessive energy usage, which our framework can effectively detect and mitigate through a feedback loop. Protocols that are prone to collisions, static routing protocols that rely on periodic advertisement, and those requiring extensive cryptographic processing are some examples of misconfigured settings when targeting the conservation of

the remaining battery level. For instance, protocols that lack effective collision avoidance mechanisms, such as Aloha or basic CSMA (Carrier Sense Multiple Access), can lead to frequent packet collisions. Each collision constrains extra retransmission, which increases energy consumption. Protocols like CSMA/CA (Collision Avoidance) used in Wi-Fi networks reduce collisions but still consume more energy compared to protocols designed for low-power networks, such as Zigbee which endures for months on a single battery. Dynamic routing protocols adapt to frequent network changes for routes that are adjusted based on network conditions. Such protocols like those used in routing protocol for low-power and lossy networks are adjustable to network changes and can reduce energy consumption by modifying path selection. Protocols with high control overheads such as frequent status updates and acknowledgments, are expected to consume more energy. Lightweight protocols designed for low-power devices such as MQTT-SN (Message Queuing Telemetry Transport for Sensor Networks) minimize overhead and consume lower levels of energy. Security protocols employing complex cryptographic operations such as TLS/SSL (Transport Layer Security/Secure Sockets Layer) can drain the battery faster. Lightweight security protocols designed for IoT such as DTLS (Datagram Transport Layer Security) offer a better compromise between security and energy consumption. Protocols with adaptive duty cycle management can save energy based on network traffic and device state. These protocols like IEEE 802.15.4 can put devices into sleep mode during idle periods, thus significantly reducing energy consumption.

Energy anomalies identified in an adequately short period prevent premature battery depletion, securing longer operational periods for IoT devices in remote or inaccessible locations. These proactive configuration adjustments obviate the need for manual intervention and periodical maintenance. In particular, large-scale IoT deployments are taking advantage of configuration adaptation, especially where maintaining an industrially scaled number of devices can be costly and time-consuming. Our adaptive energy management framework allows IoT nodes to fine-tune their performance based on real-time link quality and transmission setting data.

Energy Anomaly Detection for Processing-intensive Edge Elements As the third objective, this study emphasizes that system call traces can reveal the effect of various operations on energy consumption. It can be used to implement context-aware energy management strategies for edge elements engaged in processing-intensive operations. For instance, system calls associated with CPU-bound tasks, such as data encryption/decryption or video encoding, can lead to significant energy consumption due to the intensive computational load. By analyzing these traces, it is feasible to optimize the scheduling and execution of such tasks, potentially reducing their energy footprint. Consequently, the frequency and duration of

specific system calls are often associated with misconfiguration or malfunctioning services which results in detectable energy patterns. For example, frequent use of system calls like *fork()* and *exec()* in multi-threaded applications can indicate incompetence in process management. Subsequently, the system remains active longer than necessary, draining battery life in IoT devices. These operations which involve creating and executing new processes are resource-intensive and can lead to significant energy consumption, particularly if not managed properly.

System call tracing also enables the development of customized energy optimization strategies tailored to specific applications or use cases. This could involve fine-tuning system call usage, adjusting application workflows, or implementing custom power management policies based on the insights gained from granular tracing data. Additionally, by integrating system call data with internal factors of a system such as task priority, edge devices can adaptively toggle energy states to optimize performance and efficiency. For example, system call traces may detect resource over-consumption for a CPU-bound task while running at a low priority. The system in return throttles the CPU frequency or reschedules the task during underloading periods to conserve energy.

To summarize, a user-space DBI framework is developed through this thesis to overcome the limitations of traditional instrumentation methods by making performance analysis more efficient with low-overhead probe insertion and safe patching. We also developed a new framework for detecting energy anomalies in edge devices. It uses a modified Variational AutoEncoder which helps optimize data transmission settings and extend battery life. Additionally, we explored system call tracing to create context-aware energy management strategies for services demanding significant processing power. These contributions address existing gaps in data collection and energy management by offering practical solutions for improving IoT network performance.

7.1 Limitations and Future Research

While our contributions offer significant advantages in terms of efficient data collection and anomaly detection frameworks, they are not without limitations. Understanding these limitations is crucial for guiding future research and development efforts to enhance the frameworks' capabilities. The primary limitations are as follows:

NOProbe can instrument any series of instructions, but it does not offer benefits over conventional branch-, tranch-, and trap-based approaches if both the first and second proposed strategies fail. The first strategy fails if the instruction at the probe site and the subsequent

neighbor instruction are both single-byte and a branch lands on the second instruction. Another scenario for failure occurs when a suitable NOP cannot be found in the vicinity of the probe site. The second proposed strategy loses the advantage over the legacy techniques if there are two single-byte instructions at the fourth and fifth bytes following the probe site. These constraints arise from the complexity of the x86 architecture and the challenges associated with safe code modification. We demonstrated that NOPProbe outperforms Dyninst in terms of performance, even with a large number of threads. However, despite efforts to reduce probe insertion costs compared to Dyninst, NOPProbe cannot maintain its advantage over the trap-based approach implemented in Uprobe when the number of threads exceeds 20. Given that the overhead varies with the number of concurrent threads, future work should focus on reducing performance bottlenecks in the probe insertion algorithm. The existing architecture of NOPProbe is primarily designed for single-node systems. Extending its design to distributed systems and large-scale deployments faces serious challenges in the context of data synchronization, network transmission overhead, and maintaining low-overhead instrumentation. In such environments, the scalability of NOPProbe requires a deep investigation and enhancement to ensure the level of data generation is tolerated without significant degradation in performance. NOPProbe is currently developed for the x86 architecture. Supporting other architectures, such as ARM requires additional development efforts and may introduce architecture-specific challenges. The limited architectural support restricts the applicability of NOPProbe in heterogeneous computing environments, where different types of processors and accelerators are used. NOPProbe also lacks a design method for safe, scalable, and efficient probe removal from user space, maintaining system stability and ensuring minimal application disruption. Future work could also explore the possibility of using modern hardware-assisted features such as Intel Processor Trace (Intel PT) in DBI. This feature is supported on Broadwell processors and other newer generations. Additionally, employing static and dynamic code analysis techniques could optimize the type and placement of probes, reducing performance impact while maximizing coverage.

The effectiveness of deep learning models for anomaly detection is highly dependent on the quality and quantity of the training data. Inadequate or biased data can lead to poor model performance and generalization issues. Therefore, future work should focus on developing methods for improving the quality and representativeness of the training data, including techniques for data augmentation and synthetic data generation. While deep learning models can provide high accuracy, they often require significant computational resources, which may not be feasible for resource-constrained IoT edge devices. Therefore, this computational overhead can itself be energy-intensive, potentially counteracting the benefits of detecting and mitigating energy over-consumption. Designing energy-efficient deep learning algorithms

specifically tailored for resource-constrained IoT edge devices is essential to ensure the feasibility of deployment. A balance between detection performance and resource consumption should be explored by drawing insights from lightweight deep learning architectures, such as MobileNet and SqueezeNet, which are optimized for energy efficiency. Additionally, as the number of IoT devices and the volume of data increase, scaling the deep learning-based anomaly detection framework may become challenging. To this end, ensuring that the system can handle large-scale deployments without performance degradation is crucial. Another area for improvement involves the static threshold settings used for anomaly detection, which may not be optimal for all scenarios. Dynamic or adaptive thresholds customized for specific protocols and transmission setups could improve energy conservation. In future work, anomaly prediction and detection frameworks could be integrated using time series data and feature interaction learning to enhance proactive management in IoT edge networks. By predicting anomalies before they occur, the system could optimize energy consumption and prevent potential disruptions. Incorporating internal resource consumption metrics such as CPU/GPU/RAM utilization and NIC I/O rate could also offer a more comprehensive understanding of energy footprint, leading to even more accurate contextual underconsumption detection.

Integrating system call tracing data with additional contextual information can lead to the development of better context-aware energy management strategies. These strategies investigate the impact of various contextual factors, such as network traffic patterns, activity types, and environmental conditions, on energy consumption. Such adaptive energy management techniques dynamically adjust system behavior based on real-time tracing data and context. To inform the design of energy-efficient systems, energy-intensive operations should be identified and categorized. Consequently, the impact of system call patterns for a wide range of heterogeneous workloads on energy consumption should be studied by collecting and analyzing traces from a diverse set of devices and applications. The real-time constraint can potentially aggravate the decision-making process, as the computational capacity varies depending on whether processing occurs on the edge or in the cloud. The current design may have limitations in handling high-frequency data or large-scale deployments efficiently. Tracing at the system call level is one of the most challenging approaches in anomaly detection. This method often leads to uncontrollable data collection and management issues due to the high rate of trace generation. The time-delay embedding approach struggles to scale as tens of thousands of system calls can be generated per second just by a single process. To manage demands, adjusting the scale of data collection—such as system call level, function call level, or request level—can be practical. Since lower-level data provides better decision-making and performance, a compromise should be made to manage the associated resource consumption

effectively. Therefore, real-time constraints and desired performance levels may ultimately determine the appropriate data granularity and resource allocation.

CHAPTER 8 CONCLUSION

In the Industrial Internet of Things (IIoT), gathering comprehensive operational data is essential to analyze the performance and maintain efficiency and reliability across industrial systems. Accurate data collection supports real-time monitoring and is fundamental for detecting energy anomalies that could signal potential failures. As the complexity of managing energy in remote devices grows, having detailed and timely data becomes increasingly important for taking proactive measures and making informed decisions.

This thesis emphasizes NOProbe’s significant role in this data collection process. NOProbe is introduced as an efficient DBI framework, specifically designed for the x86 architecture. It uses innovative NOP- and pseudo-NOP-based techniques to gather detailed data while minimizing performance impact. This makes NOProbe particularly suited for analyzing the performance of resource-constrained IIoT elements. Traditional instrumentation techniques often come with high overhead due to the complexity of binary code modification and managing variable-length instructions. NOProbe addresses these challenges by using userspace-safe probe insertion methods. These methods avoid unnecessary user-kernel space switches during probe execution, significantly reducing performance overhead. This reduction allows NOProbe to improve real-time monitoring and make a more efficient and effective tool for data collection in complex edge systems.

NOProbe’s detailed and low-overhead data collection integrates well with anomaly detection frameworks. It provides the precise input needed for accurate energy monitoring. Efficiently collecting transmission settings—like signal strength, bandwidth utilization, and link quality metrics—is crucial for identifying early signs of energy anomalies. Properly configuring these settings, including the choice of transmission protocols, has a direct impact on the energy consumption of SDs and edge devices. By continuously monitoring and optimizing these parameters, the system can detect deviations from normal patterns that indicate potential failures or inefficiencies. This allows for timely interventions and adjustments, ensuring that transmission settings are optimized to minimize energy consumption while maintaining performance.

System calls, as captured by NOProbe, also play a key role in understanding the energy dynamics of IIoT elements. By monitoring the frequency, type, and sequence of system calls, the anomaly detection framework can correlate specific operational behaviors with energy consumption patterns. This enables the early detection of anomalies, such as unexpected spikes in energy use that might be linked to inefficient processes or hardware malfunctions.

In conclusion, this thesis presents two complementary anomaly detection frameworks—the first leverages external data collection for communication-intensive edge elements through transmission settings and link quality data. The second focuses on internal data collection for processing-intensive edge elements through system calls. NOProbe’s efficient data collection enhances both frameworks by delivering granular system traces that significantly aid in diagnosing energy performance. This allows for more accurate and timely anomaly detection, ultimately optimizing energy management in IIoT environments.

REFERENCES

- [1] C. B. Parker, “Adversarial machine learning for the protection of legitimate software,” Ph.D. dissertation, University of South Alabama, 2022.
- [2] X. Li, J. Peng, J. Niu, F. Wu, J. Liao, and K.-K. R. Choo, “A robust and energy efficient authentication protocol for industrial internet of things,” *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1606–1615, 2018.
- [3] J. Toledo-Castro, P. Caballero-Gil, N. Rodríguez-Pérez, I. Santos-González, C. Hernández-Goya, R. Aguasca-Colomo, and A. Schumann, “Forest fire prevention, detection, and fighting based on fuzzy logic and wireless sensor networks,” *Complex.*, vol. 2018, jan 2018. [Online]. Available: <https://doi.org/10.1155/2018/1639715>
- [4] R. Pachlor and D. Shrimankar, “Vch-eccr: A centralized routing protocol for wireless sensor networks,” *Journal of Sensors*, vol. 2017, no. 1, p. 8946576, 2017. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2017/8946576>
- [5] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, “Industrial internet of things: Challenges, opportunities, and directions,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.
- [6] G. M. Abdulsahib and O. I. Khalaf, “Accurate and effective data collection with minimum energy path selection in wireless sensor networks using mobile sinks,” *Journal of Information Technology Management*, vol. 13, no. 2, pp. 139–153, 2021. [Online]. Available: https://jitm.ut.ac.ir/article_80359.html
- [7] L. Dash and M. Khuntia, “Energy efficient techniques for 5g mobile networks in wsn: A survey,” in *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, 2020, pp. 1–5.
- [8] E. S. Ali, R. A. Saeed, I. K. Eltahir, and O. O. Khalifa, “A systematic review on energy efficiency in the internet of underwater things (iout): Recent approaches and research gaps,” *Journal of Network and Computer Applications*, vol. 213, p. 103594, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804523000139>
- [9] Y. Sun, J. Wang, and X. Wang, “Fault diagnosis of mechanical equipment in high energy consumption industries in china: A review,” *Mechanical*

- Systems and Signal Processing*, vol. 186, p. 109833, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0888327022009013>
- [10] Y. Padarathi and R. R. P. Kuppusamy, “Iot-based embedded sensor system for real-time health monitoring of composite structures for large-scale industrial operations,” in *Industrial Automation and Robotics*. CRC Press, 2022, pp. 3–32.
 - [11] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, “Edge computing in industrial internet of things: Architecture, advances and challenges,” *IEEE Communications Surveys Tutorials*, 2020.
 - [12] M. A. Khan, “A survey of computation offloading strategies for performance improvement of applications running on mobile devices,” *Journal of Network and Computer Applications*, 2015.
 - [13] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, 2016.
 - [14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, 2016.
 - [15] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, “Performance guaranteed computation offloading for mobile-edge cloud computing,” *IEEE Wireless Communications Letters*, 2017.
 - [16] I. Kohyarnnejadfard, M. Shakeri, and D. Aloise, “System performance anomaly detection using tracing data analysis,” in *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, ser. ICCTA ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 169–173. [Online]. Available: <https://doi.org/10.1145/3323933.3324085>
 - [17] A. Iranfar, M. Kamal, A. Afzali-Kusha, M. Pedram, and D. Atienza, “Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1532–1545, 2018.
 - [18] C. Niemann, C. Ewert, H. Puttnies, M. Rethfeldt, D. Timmermann, and P. Danielis, “Modeling energy consumption for task-offloading decisions on mobile and embedded devices,” in *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*, 2020, pp. 400–404.

- [19] A. S. Bushehri, S. Keivanpour, M. Azam, and G. Niculescu, “Anomalous energy detection for resource-constrained embedded systems using tracing data analysis,” in *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2021, pp. 1–8.
- [20] S. Dey, A. K. Singh, S. Saha, X. Wang, and K. D. McDonald-Maier, “Rewardprofiler: A reward based design space profiler on dvfs enabled mpsoes,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2019, pp. 210–220.
- [21] A. Katal, S. Dahiya, and T. Choudhury, “Energy efficiency in cloud computing data centers: a survey on software technologies,” *Cluster Computing*, vol. 26, no. 3, pp. 1845–1875, 2023.
- [22] P. Bratek, L. Szustak, R. Wyrzykowski, and T. Olas, “Reducing energy consumption using heterogeneous voltage frequency scaling of data-parallel applications for multicore systems,” *Journal of Parallel and Distributed Computing*, vol. 175, pp. 121–133, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0743731523000059>
- [23] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives,” *Journal of Systems Architecture*, vol. 129, p. 102561, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762122001138>
- [24] P. Dibal, E. Onwuka, S. Zubair, E. Nwankwo, S. Okoh, B. A. Salihu, and H. Mustaphab, “Processor power and energy consumption estimation techniques in iot applications: A review,” *Internet of Things*, vol. 21, p. 100655, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660522001366>
- [25] J. Chen, M. Manivannan, M. Abduljabbar, and M. Pericàs, “Erase: Energy efficient task mapping and resource management for work stealing runtimes,” *ACM Trans. Archit. Code Optim.*, vol. 19, no. 2, mar 2022. [Online]. Available: <https://doi.org/10.1145/3510422>
- [26] C. Guo, S. Ci, Y. Zhou, and Y. Yang, “A survey of energy consumption measurement in embedded systems,” *IEEE Access*, vol. 9, pp. 60 516–60 530, 2021.

- [27] J. Álvarez Cid-Fuentes, C. Szabo, and K. Falkner, “Adaptive performance anomaly detection in distributed systems using online svms,” *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 928–941, 2020.
- [28] D. P. Johnson, E. C. Saxe, and B. Smaalders, “Frequency scaling of processing unit based on aggregate thread cpi metric,” Jul. 10 2012, uS Patent 8,219,993.
- [29] R. Hebbbar and A. Milenković, “An experimental evaluation of workload driven dvfs,” in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 95–102. [Online]. Available: <https://doi.org/10.1145/3447545.3451192>
- [30] S. S., “Gdb tracepoints, redux,” *Proceedings of the GCC Developers’ Summit*.
- [31] R. Chouhan, “Efficient and compatible cfi for x86-64 binaries,” Ph.D. dissertation, 2023.
- [32] D. Chisnall, “How to design an isa: The popularity of risc-v has led many to try designing instruction sets.” *Queue*, vol. 21, no. 6, p. 27–46, jan 2024. [Online]. Available: <https://doi.org/10.1145/3639445>
- [33] I. Alshaer, G. Burghoorn, B. Colombier, C. Deleuze, V. Beroulle, and P. Maistri, “Cross-layer analysis of clock glitch fault injection while fetching variable-length instructions,” *Journal of Cryptographic Engineering*, pp. 1–18, 2024.
- [34] I. Alshaer, B. Colombier, C. Deleuze, V. Beroulle, and P. Maistri, “Variable-length instruction set: Feature or bug?” in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 464–471.
- [35] “Intel® 64 and ia-32 architectures software developer’s manual,” vol. 3A: System Programming Guide, Part 1, 2016. [Online]. Available: <https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.pdf>
- [36] P. Mishra, N. Kumar, and W. W. Godfrey, “An evolutionary computing-based energy-efficient solution for iot-enabled software-defined sensor network architecture,” *International Journal of Communication Systems*, vol. 35, no. 8, p. e5111, 2022. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.5111>
- [37] D. E. Ruíz-Guirola, C. A. Rodríguez-López, S. Montejo-Sánchez, R. D. Souza, and M. A. Imran, “Drx-based energy-efficient supervised machine learning algorithm

- for mobile communication networks,” *IET Communications*, vol. 15, no. 7, pp. 1000–1013, 2021. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cmu2.12137>
- [38] O. I. Khalaf, C. A. T. Romero, S. Hassan, and M. T. Iqbal, “Mitigating hotspot issues in heterogeneous wireless sensor networks,” *Journal of Sensors*, vol. 2022, no. 1, p. 7909472, 2022. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/7909472>
- [39] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, “An empirical study of practitioners’ perspectives on green software engineering,” in *Proceedings of the 38th International Conference on Software Engineering*. Association for Computing Machinery, 2016.
- [40] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, “What do programmers know about software energy consumption?” *IEEE Software*, 2016.
- [41] F. Ciccozzi, J. Feljan, J. Carlson, and I. Crnković, “Architecture optimization: Speed or accuracy? both!” *Software Quality Journal*, 2018.
- [42] A. Nouredine and A. Rajan, “Optimising energy consumption of design patterns,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015.
- [43] D. Li, S. Hao, J. Gui, and W. G. Halfond, “An empirical study of the energy consumption of android applications,” in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014.
- [44] I. Manotas, L. Pollock, and J. Clause, “Seeds: A software engineer’s energy-optimization decision support framework,” in *Proceedings of the 36th International Conference on Software Engineering*. Association for Computing Machinery, 2014.
- [45] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, “Energy profiles of java collections classes,” in *Proceedings of the 38th International Conference on Software Engineering*. Association for Computing Machinery, 2016.
- [46] K. Grosskop and J. Visser, “Identification of application-level energy optimizations,” Eidgenössische Technische Hochschule Zürich *ETH Zürich*, 2013.
- [47] E. Jagroep, J. M. E. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet, “Extending software architecture views with an energy consumption perspective - A case study on resource consumption of enterprise software,” *Computing*, 2017.

- [48] C. Stier, A. Koziol, H. Groenda, and R. Reussner, “Model-based energy efficiency analysis of software architectures,” in *Software Architecture*. Springer International Publishing, 2015.
- [49] B. Ouni, H. Ben Rekhiss, and C. Belleudy, “Inter-process communication energy estimation through aadl modeling,” in *2012 International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2012.
- [50] H. M. Alvi, H. Majeed, H. Mujtaba, and M. O. Beg, “Mlee: Method level energy estimation — a machine learning approach,” *Sustainable Computing: Informatics and Systems*, 2021.
- [51] A. Nouredine, R. Rouvoy, and L. Seinturier, “Monitoring energy hotspots in software,” *Automated Software Engg.*, 2015.
- [52] S. Götz, C. Wilke, S. Cech, and U. Assmann, “Runtime variability management for energy-efficient software by contract negotiation,” 2011.
- [53] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems,” in *2008 45th ACM/IEEE Design Automation Conference*, 2008.
- [54] A. Schranzhofer, J.-J. Chen, and L. Thiele, “Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms,” *IEEE Transactions on Industrial Informatics*, 2010.
- [55] W. Quan and A. D. Pimentel, “A scenario-based run-time task mapping algorithm for mpsocs,” in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013.
- [56] A. Weissel and F. Bellosa, “Process cruise control: Event-driven clock scaling for dynamic power management.” Association for Computing Machinery, 2002.
- [57] L. C. Singleton, C. Poellabauer, and K. Schwan, “Monitoring of cache miss rates for accurate dynamic voltage and frequency scaling.” SPIE, 2005.
- [58] W. Quan and A. D. Pimentel, “A hybrid task mapping algorithm for heterogeneous mpsocs,” *ACM Trans. Embed. Comput. Syst.*, 2015.
- [59] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, “Pack amp; cap: Adaptive dvfs and thread packing under power caps,” 2011.

- [60] H. Sasaki, S. Imamura, and K. Inoue, “Coordinated power-performance optimization in manycores,” 2013.
- [61] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, “Scheduling heterogeneous multi-cores through performance impact estimation (pie),” 2012.
- [62] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, “Power-aware performance adaptation of concurrent applications in heterogeneous many-core systems.” Association for Computing Machinery, 2016.
- [63] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, “Workload-aware power optimization strategy for asymmetric multiprocessors,” 2016.
- [64] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, “Sparta: Runtime task allocation for energy efficient heterogeneous manycores,” 2016.
- [65] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, “Dynamic resource management of heterogeneous mobile platforms via imitation learning,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.
- [66] S. Dey, A. K. Singh, D. K. Prasad, and K. D. McDonald-Maier, “Socodecnn: Program source code for visual cnn classification using computer vision methodology,” *IEEE Access*, 2019.
- [67] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras, “Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs,” 2017.
- [68] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, “Learning transfer-based adaptive energy minimization in embedded systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.
- [69] P. K. Holzenspies, J. L. Hurink, J. Kuper, and G. J. Smit, “Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc),” in *2008 Design, Automation and Test in Europe*, 2008.
- [70] V. Petrucci, O. Loques, D. Mossé, R. Melhem, N. A. Gazala, and S. Gobriel, “Energy-efficient thread assignment optimization for heterogeneous multicore systems,” *ACM Trans. Embed. Comput. Syst.*, 2015.

- [71] B. K. Reddy, A. K. Singh, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi, "Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, 2018.
- [72] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett, "Adamd: Adaptive mapping and dvfs for energy-efficient heterogeneous multicores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.
- [73] N. Peters, D. Füß, S. Park, and S. Chakraborty, "Frame-based and thread-based power management for mobile games on hmp platforms," 2016.
- [74] S. Dey, A. K. Singh, X. Wang, and K. D. McDonald-Maier, "Deadpool: Performance deadline based frequency pooling and thermal management agent in dvfs enabled mp-socs," 2019.
- [75] B. A. Begum and S. V. Nandury, "Data aggregation protocols for wsn and iot applications – a comprehensive survey," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 651–681, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1319157823000083>
- [76] C. Jiang, Y. Qiu, H. Gao, T. Fan, K. Li, and J. Wan, "An edge computing platform for intelligent operational monitoring in internet data centers," *IEEE Access*, vol. 7, pp. 133 375–133 387, 2019.
- [77] E. Fazel, H. E. Najafabadi, M. Rezaei, and H. Leung, "Unlocking the power of mist computing through clustering techniques in iot networks," *Internet of Things*, vol. 22, p. 100710, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660523000331>
- [78] O. Kanoun, S. Bradai, S. Khriji, G. Bouattour, D. El Houssaini, M. Ben Ammar, S. Naifar, A. Bouhamed, F. Derbel, and C. Viehweger, "Energy-aware system design for autonomous wireless sensor nodes: A comprehensive review," *Sensors*, vol. 21, no. 2, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/2/548>
- [79] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer Networks*, vol. 67, pp. 104–122, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128614001418>
- [80] P. S. Jayakrishna and T. Sudha, "Energy efficient wireless sensor network assisted spectrum sensing for cognitive radio network," in *2017 IEEE International Conference*

- on *Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, 2017, pp. 1–7.
- [81] G. P. Joshi, S. Y. Nam, and S. W. Kim, “Cognitive radio wireless sensor networks: Applications, challenges and research trends,” *Sensors*, vol. 13, no. 9, pp. 11 196–11 228, 2013. [Online]. Available: <https://www.mdpi.com/1424-8220/13/9/11196>
 - [82] M. A. Alamri, “An efficient cooperative technique for power-constrained multiuser wireless network: “investigation of cooperation strategies in multiuser wireless network with the power constraint”,” *Telecommunication Systems*, vol. 69, no. 3, pp. 263–271, 2018.
 - [83] T. Cevik and A. Gunagwera, “A directional multicasting-based architecture for wireless sensor networks,” *International Journal of Electronics*, vol. 106, no. 10, pp. 1441–1462, 2019. [Online]. Available: <https://doi.org/10.1080/00207217.2019.1600734>
 - [84] C. Lee, J. Oh, W. Na, J. Yoon, W. Noh, and S. Cho, “Directional-antenna-based spatial and energy-efficient semi-distributed spectrum sensing in cognitive internet-of-things networks,” *Journal of Network and Computer Applications*, vol. 217, p. 103687, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804523001066>
 - [85] M. Abo-Zahhad, M. Farrag, and A. Ali, “Optimization of transmitted power and modulation level for minimizing energy consumption in wireless sensor networks,” *Wireless Personal Communications*, vol. 96, pp. 4047–4062, 2017.
 - [86] H. A. B. Salameh, M. Bani Irshaid, A. Al Ajlouni, and M. Aloqaily, “Energy-efficient cross-layer spectrum sharing in cr green iot networks,” *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 3, pp. 1091–1100, 2021.
 - [87] X. Wang and C. Zhai, “Dynamic power control for cell-free industrial internet of things with random data arrivals,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4138–4147, 2022.
 - [88] K.-H. Hsia, C.-W. Hung, H. T. Chang, and Y.-H. Lai, “Transmission power control for wireless sensor network.” *J. Robotics Netw. Artif. Life*, vol. 3, no. 4, pp. 279–282, 2017.
 - [89] X. Liu, M. Zhao, A. Liu, and K. K. L. Wong, “Adjusting forwarder nodes and duty cycle using packet aggregation routing for body sensor networks,” *Information Fusion*, vol. 53, pp. 183–195, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1566253519302106>

- [90] O. Kanoun, *Energy Harvesting for Wireless Sensor Networks: Technology, Components and System Design*. Walter de Gruyter GmbH & Co KG, 2018.
- [91] A. A. Shabaneh, A. M. Ali, C. K. Ng, N. K. Noordin, A. Sali, and M. H. Yaacob, "Review of energy conservation using duty cycling schemes for ieee 802.15. 4 wireless sensor network (wsn)," *Wireless personal communications*, vol. 77, pp. 589–604, 2014.
- [92] Y. Du, J. Xia, J. Gong, and X. Hu, "An energy-efficient and fault-tolerant topology control game algorithm for wireless sensor network," *Electronics*, vol. 8, no. 9, 2019. [Online]. Available: <https://www.mdpi.com/2079-9292/8/9/1009>
- [93] R. Kavra, A. Gupta, and S. Kansal, "Systematic study of topology control methods and routing techniques in wireless sensor networks," *Peer-to-Peer Networking and Applications*, vol. 15, no. 4, pp. 1862–1922, 2022.
- [94] M. Demirci, "A survey of machine learning applications for energy-efficient resource management in cloud computing environments," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, pp. 1185–1190.
- [95] N. Vasić, D. Novaković, S. Miućin, D. Kostić, and R. Bianchini, "Dejavu: accelerating resource allocation in virtualized environments," *SIGPLAN Not.*, vol. 47, no. 4, p. 423–436, mar 2012. [Online]. Available: <https://doi.org/10.1145/2248487.2151021>
- [96] T. V. T. Duy, Y. Sato, and Y. Inoguchi, "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–8.
- [97] J. L. Berral, I. n. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, ser. e-Energy '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 215–224. [Online]. Available: <https://doi.org/10.1145/1791314.1791349>
- [98] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient cloud resource management," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 386–391.
- [99] I. AlQerm and B. Shihada, "Enhanced machine learning scheme for energy efficient resource allocation in 5g heterogeneous cloud radio access networks," in *2017 IEEE*

28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017, pp. 1–7.

- [100] S. Jiang, S. R. Priya, N. Elango, J. Clay, and R. Sridhar, “An energy efficient in-memory computing machine learning classifier scheme,” in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSI-SID)*, 2019, pp. 157–162.
- [101] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, “Machine learning at facebook: Understanding inference at the edge,” 2019.
- [102] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, “{TVM}: An automated end-to-end optimizing compiler for deep learning,” 2018.
- [103] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein *et al.*, “Glow: Graph lowering compiler techniques for neural networks,” *arXiv preprint arXiv:1805.00907*, 2018.
- [104] Y. Bai, L. Chen, M. Abdel-Mottaleb, and J. Xu, “Automated ensemble for deep learning inference on edge computing platforms,” *IEEE Internet of Things Journal*, 2021.
- [105] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, “Energy consumption in mobile phones: A measurement study and implications for network applications.” Association for Computing Machinery, 2009.
- [106] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding, “Cool-tether: Energy efficient on-the-fly wifi hot-spots using mobile phones.” Association for Computing Machinery, 2009.
- [107] A. Kumar, S. Goyal, and M. Varma, “Resource-efficient machine learning in 2 kb ram for the internet of things.” JMLR.org, 2017.
- [108] X. Zhang, A. Ramachandran, C. Zhuge, D. He, W. Zuo, Z. Cheng, K. Rupnow, and D. Chen, “Machine learning on fpgas to face the iot revolution,” 2017.
- [109] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, “Energy conservation in wireless sensor networks: A survey,” *Ad hoc networks*, 2009.
- [110] M. A. Razzaque, C. Bleakley, and S. Dobson, “Compression in wireless sensor networks: A survey and comparative evaluation,” *ACM Trans. Sen. Netw.*, 2013.

- [111] H. Harb, A. Makhoul, and C. A. Jaoude, “En-route data filtering technique for maximizing wireless sensor network lifetime,” 2018.
- [112] H. Li, K. Ota, and M. Dong, “Learning iot in edge: Deep learning for the internet of things with edge computing,” *IEEE Network*, 2018.
- [113] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, “Deepx: A software accelerator for low-power deep learning inference on mobile devices,” 2016.
- [114] J. Azar, A. Makhoul, R. Darazi, J. Demerjian, and R. Couturier, “On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks,” 2018.
- [115] J. Azar, R. Darazi, C. Habib, A. Makhoul, and J. Demerjian, “Using dwt lifting scheme for lossless data compression in wireless body sensor networks,” 2018.
- [116] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, “An energy efficient iot data compression approach for edge machine learning,” *Future Generation Computer Systems*, 2019.
- [117] Q. Zeng, Y. Du, K. Huang, and K. K. Leung, “Energy-efficient radio resource allocation for federated edge learning,” 2020.
- [118] K. Thangaramya, K. Kulothungan, R. Logambigai, M. Selvi, S. Ganapathy, and A. Kannan, “Energy aware cluster and neuro-fuzzy based routing algorithm for wireless sensor networks in iot,” *Computer Networks*, 2019.
- [119] A. Sayed, A. Alsalemi, Y. Himeur, F. Bensaali, and A. Amira, “Endorsing energy efficiency through accurate appliance-level power monitoring, automation and data visualization,” in *Networking, Intelligent Systems and Security: Proceedings of NISS 2021*. Springer, 2021, pp. 603–617.
- [120] A. Alsalemi, Y. Himeur, F. Bensaali, and A. Amira, “An innovative edge-based internet of energy solution for promoting energy saving in buildings,” *Sustainable Cities and Society*, vol. 78, p. 103571, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210670721008362>
- [121] —, “Appliance-level monitoring with micro-moment smart plugs,” in *Innovations in Smart Cities Applications Volume 4: The Proceedings of the 5th International Conference on Smart City Applications*. Springer, 2021, pp. 942–953.

- [122] L. Lei, B. Wu, X. Fang, L. Chen, H. Wu, and W. Liu, "A dynamic anomaly detection method of building energy consumption based on data mining technology," *Energy*, vol. 263, p. 125575, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360544222024616>
- [123] A. Copiaco, Y. Himeur, A. Amira, W. Mansoor, F. Fadli, S. Atalla, and S. S. Sohail, "An innovative deep anomaly detection of building energy consumption using energy time-series images," *Engineering Applications of Artificial Intelligence*, vol. 119, p. 105775, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0952197622007655>
- [124] J. Kong, W. Jiang, Q. Tian, M. Jiang, and T. Liu, "Anomaly detection based on joint spatio-temporal learning for building electricity consumption," *Applied Energy*, vol. 334, p. 120635, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S030626192201892X>
- [125] E. Shamsa, A. Kanduri, P. Liljeberg, and A. M. Rahmani, "Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 743–755, 2022.
- [126] E. Shamsa, A. Kanduri, A. M. Rahmani, and P. Liljeberg, "Energy-performance co-management of mixed-sensitivity workloads on heterogeneous multi-core systems," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 421–427. [Online]. Available: <https://doi.org/10.1145/3394885.3431516>
- [127] H. Ahmadvand, F. Foroutan, and M. Fathy, "Dv-dvfs: merging data variety and dvfs technique to manage the energy consumption of big data processing," *Journal of Big Data*, vol. 8, pp. 1–16, 2021.
- [128] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer networks*, 2014.
- [129] P. Jayakrishna and T. Sudha, "Energy efficient wireless sensor network assisted spectrum sensing for cognitive radio network." IEEE, 2017.
- [130] G. P. Joshi, S. Y. Nam, and S. W. Kim, "Cognitive radio wireless sensor networks: applications, challenges and research trends," *Sensors*, 2013.
- [131] T. Cevik and A. Gunagwera, "A directional multicasting-based architecture for wireless sensor networks," *International Journal of Electronics*, 2019.

- [132] C. Lee, J. Oh, W. Na, J. Yoon, W. Noh, and S. Cho, “Directional-antenna-based spatial and energy-efficient semi-distributed spectrum sensing in cognitive internet-of-things networks,” *Journal of Network and Computer Applications*, 2023.
- [133] Y. Du, J. Xia, J. Gong, and X. Hu, “An energy-efficient and fault-tolerant topology control game algorithm for wireless sensor network,” *Electronics*, 2019.
- [134] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, “Energy-efficient cloud resource management,” in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014.
- [135] N. Vasić, D. Novaković, S. Miućin, D. Kostić, and R. Bianchini, “Dejavu: Accelerating resource allocation in virtualized environments.” Association for Computing Machinery, 2012.
- [136] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, “Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms,” 2017.
- [137] S. Malik, M. Tahir, M. Sardaraz, and A. Alourani, “A resource utilization prediction model for cloud data centers using evolutionary algorithms and machine learning techniques,” *Applied Sciences*, 2022.
- [138] J. Kumar, D. Saxena, A. K. Singh, and A. Mohan, “Biphase adaptive learning-based neural network model for cloud datacenter workload forecasting,” *Soft Computing*, 2020.
- [139] D. Saxena and A. K. Singh, “A proactive autoscaling and energy-efficient vm allocation framework using online multi-resource neural network for cloud data center,” *Neurocomputing*, 2021.
- [140] S. Kreutzer, C. Iwainsky, M. Garcia-Gasulla, V. Lopez, and C. Bischof, “Runtime-adaptable selective performance instrumentation,” in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2023, pp. 423–432.
- [141] A. WONG, P. BUCCI, I. BESCHASTNIKH, and A. FEDOROVA, “Making sense of multi-threaded application performance at scale with nonsequitur,” 2024.
- [142] M. Wang, J. Liang, C. Zhou, Z. Wu, X. Xu, and Y. Jiang, “Odin: on-demand instrumentation with on-the-fly recompilation,” in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design*

- and Implementation*, ser. PLDI 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1010–1024. [Online]. Available: <https://doi.org/10.1145/3519939.3523428>
- [143] B. Chamith, B. J. Svensson, L. Dalessandro, and R. R. Newton, “Instruction punning: Lightweight instrumentation for x86-64,” *SIGPLAN Not.*, vol. 52, no. 6, pp. 320–332, Jun. 2017. [Online]. Available: <http://doi.acm.org/10.1145/3140587.3062344>
 - [144] C. Harper-Cyr, M. Dagenais, and A. Shahnejat Bushehri, “Fast and flexible tracepoints in x86,” *Software: Practice and Experience*, 2019.
 - [145] B. Chamith, B. J. Svensson, L. Dalessandro, and R. R. Newton, “Instruction punning: Lightweight instrumentation for x86-64,” 2017.
 - [146] A. Tiwari, M. Schulz, and L. Carrington, “Predicting optimal power allocation for cpu and dram domains,” 2015.
 - [147] S. Shebs, “Gdb tracepoints, redux,” *Proceedings of the GCC Developers’ Summit. GCC Summit. Montréal, Canada*, 2009.
 - [148] A. R. Bernat and B. P. Miller, “Anywhere, any-time binary instrumentation,” 2011.
 - [149] J. Oh, S. Kim, E. Jeong, and S.-M. Moon, “Os-less dynamic binary instrumentation for embedded firmware,” in *2015 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII)*, 2015.
 - [150] J. Misurda, J. Clause, J. Reed, B. Childers, and M. Soffa, “Demand-driven structural testing with dynamic instrumentation,” in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005.
 - [151] K. Adams and O. Agesen, “A comparison of software and hardware techniques for x86 virtualization.” Association for Computing Machinery, 2006.
 - [152] N. Aaraj, A. Raghunathan, and N. K. Jha, “Dynamic binary instrumentation-based framework for malware defense,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer Berlin Heidelberg, 2008.
 - [153] S. Lima, J. Correia, F. Araujo, and J. Cardoso, “Improving observability in event sourcing systems,” *Journal of Systems and Software*, 2021.
 - [154] A. Liu, C. Martin, T. Hetherington, and S. Matzner, “A comparison of system call feature representations for insider threat detection,” in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005.

- [155] I. Kohyarnejadfard, M. Shakeri, and D. Aloise, "System performance anomaly detection using tracing data analysis." Association for Computing Machinery, 2019.
- [156] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005.
- [157] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015.
- [158] M. Dymshits, B. Myara, and D. Tolpin, "Process monitoring on sequences of system call count vectors," *CoRR*, 2017.
- [159] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences." Springer International Publishing, 2016.
- [160] G. Tandon and P. Chan, "On the learning of system call attributes for host-based anomaly detection," *Int. J. Artif. Intell. Tools*, 2006.
- [161] Q. Fournier, D. Aloise, S. V. Azhari, and F. Tetreault, "On improving deep learning trace analysis with system call arguments," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021.
- [162] G. Tandon and P. Chan, "Learning useful system call attributes for anomaly detection." 2005.
- [163] O. M. Ezeme, Q. Mahmoud, and A. Azim, "A framework for anomaly detection in time-driven and event-driven processes using kernel traces," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [164] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019.
- [165] M. Dymshits, B. Myara, and D. Tolpin, "Process monitoring on sequences of system call count vectors." IEEE, 2017.
- [166] Z. Xu, X. Yu, Y. Feng, J. Hu, Z. Tari, and F. Han, "A multi-module anomaly detection scheme based on system call prediction," in *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, 2013.

- [167] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, “A survey on the evolution of stream processing systems,” *The VLDB Journal*, vol. 33, no. 2, pp. 507–541, 2024.
- [168] S. Bou, T. Amagasa, and H. Kitagawa, “O(1)-time complexity for fixed sliding-window aggregation over out-of-order data streams,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–12, 2024.
- [169] E. Eskin, W. Lee, and S. Stolfo, “Modeling system calls for intrusion detection with dynamic window sizes,” in *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX’01*, 2001.
- [170] L. Liu, H. Zhang, Y. Jing, Z. He, K. Zhang, and X. S. Wang, “Learned optimizer for online approximate query processing in data exploration,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 8, pp. 3977–3991, 2024.
- [171] N. Sheoran, S. Chockchowwat, A. Chheda, S. Wang, R. Verma, and Y. Park, “A step toward deep online aggregation,” *Proc. ACM Manag. Data*, vol. 1, no. 2, jun 2023. [Online]. Available: <https://doi.org/10.1145/3589269>
- [172] B. Bejeck, *Kafka Streams in Action*. Simon and Schuster, 2024.
- [173] B. G. Deepthi, K. S. Rani, P. V. Krishna, and V. Saritha, “An efficient architecture for processing real-time traffic data streams using apache flink,” *Multimedia Tools and Applications*, vol. 83, no. 13, pp. 37 369–37 385, 2024.
- [174] C.-Y. Wu, “Machine learning-based task scheduling for apache storm,” Master’s thesis, The Catholic University of America, 2024.
- [175] L. Erhan, M. Ndubuaku, M. Di Mauro, W. Song, M. Chen, G. Fortino, O. Bagdasar, and A. Liotta, “Smart anomaly detection in sensor systems: A multi-perspective review,” *Information Fusion*, 2021.
- [176] B. Feng, Q. Li, X. Pan, J. Zhang, and D. Guo, “Groupfound: An effective approach to detect suspicious accounts in online social networks,” *International Journal of Distributed Sensor Networks*, 2017.
- [177] K. Verma, B. Singh, and A. Dixit, “A review of supervised and unsupervised machine learning techniques for suspicious behavior recognition in intelligent surveillance system,” *International Journal of Information Technology*, 2019.

- [178] R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, M. A. Amanullah, I. Abaker Targio Hashem, E. Ahmed, and M. Imran, “Clustering-based real-time anomaly detection—a breakthrough in big data technologies,” *Transactions on Emerging Telecommunications Technologies*.
- [179] K. S. A. Kumar and A. M. M. O. Chacko, “Clustering algorithms for intrusion detection: A broad visualization,” in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*. Association for Computing Machinery, 2016.
- [180] M. Ahmed, A. N. Mahmood, and M. R. Islam, “A survey of anomaly detection techniques in financial domain,” *Future Generation Computer Systems*, 2016.
- [181] E. Vanem and A. Brandsæter, “Unsupervised anomaly detection based on clustering methods and sensor data on a marine diesel engine,” *Journal of Marine Engineering Technology*, 2019.
- [182] P. Arjunan, H. D. Khadilkar, T. Ganu, Z. M. Charbiwala, A. Singh, and P. Singh, “Multi-user energy consumption monitoring and anomaly detection with partial context information,” in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. Association for Computing Machinery, 2015.
- [183] B. Rossi, S. Chren, B. Buhnova, and T. Pitner, “Anomaly detection in smart grid data: An experience report,” in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016.
- [184] J. Henriques, F. Caldeira, T. Cruz, and P. Simões, “Combining k-means and xgboost models for anomaly detection using log datasets,” *Electronics*, 2020.
- [185] H. Izakian and W. Pedrycz, “Anomaly detection in time series data using a fuzzy c-means clustering,” in *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, 2013.
- [186] J. Yeckle and B. Tang, “Detection of electricity theft in customer consumption using outlier detection algorithms,” in *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, 2018.
- [187] K. Guo, B. Hu, J. Ma, S. Ren, Z. Tao, and J. Zhang, “Toward anomaly behavior detection as an edge network service using a dual-task interactive guided neural network,” *IEEE Internet of Things Journal*, 2021.

- [188] M. Atemkeng and T. Narh, “Anomaly detection in power generation plants using machine learning and neural networks,” *Applied Artificial Intelligence*, 2019.
- [189] A. Sial, A. Singh, and A. Mahanti, “Detecting anomalous energy consumption using contextual analysis of smart meter data,” *Wireless Networks*, 2021.
- [190] V. Jakkula and D. Cook, “Outlier detection in smart environment structured power datasets,” in *2010 Sixth International Conference on Intelligent Environments*, 2010.
- [191] J. Nagi, K. S. Yap, S. K. Tiong, S. K. Ahmed, and A. M. Mohammad, “Detection of abnormalities and electricity theft using genetic support vector machines,” in *TENCON 2008 - 2008 IEEE Region 10 Conference*, 2008.
- [192] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, “Support vector machine based data classification for detection of electricity theft,” in *2011 IEEE/PES Power Systems Conference and Exposition*, 2011.
- [193] A. Amara korba and N. El Islem karabadji, “Smart grid energy fraud detection using svm,” in *2019 International Conference on Networking and Advanced Systems (IC-NAS)*, 2019.
- [194] L. Zhang, L. Wan, Y. Xiao, S. Li, and C. Zhu, “Anomaly detection method of smart meters data based on gmm-lda clustering feature learning and pso support vector machine,” in *2019 IEEE Sustainable Power and Energy Conference (iSPEC)*, 2019.
- [195] K. Kammerer, B. Hoppenstedt, R. Pryss, S. Stöckler, J. Allgaier, and M. Reichert, “Anomaly detections for manufacturing systems based on sensor data—insights into two challenging real-world production settings,” *Sensors*, 2019.
- [196] C. Cody, V. Ford, and A. Siraj, “Decision tree learning for fraud detection in consumer energy consumption,” in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015.
- [197] M. Reif, M. Goldstein, A. Stahl, and T. M. Breuel, “Anomaly detection by combining decision trees and parametric densities,” in *2008 19th International Conference on Pattern Recognition*, 2008.
- [198] C. Bresch, “Approaches, strategies, and implementations of memory safety defenses in critical and constrained embedded systems,” Ph.D. dissertation, Université Grenoble Alpes [2020-....], 2020.

- [199] H. R. Ayers, “Making small embedded systems secure and dependable,” Ph.D. dissertation, 2023. [Online]. Available: <https://www.proquest.com/dissertations-theses/making-small-embedded-systems-secure-dependable/docview/2877956931/se-2>
- [200] C. Guo, S. Ci, Y. Zhou, and Y. Yang, “A survey of energy consumption measurement in embedded systems,” *IEEE Access*, vol. 9, pp. 60 516–60 530, 2021.
- [201] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob, “Cmp \$ im: A pin-based on-the-fly multi-core cache simulator,” in *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, 2008, pp. 28–36.
- [202] J. Poimboeuf, “Introducing kpatch: dynamic kernel patching,” Red Hat, Tech. Rep., 2014.
- [203] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification, Java SE 7 Edition*, 1st ed. Addison-Wesley Professional, 2013.
- [204] B. Gregg and J. Mauro, *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2011.
- [205] A. Mavinakayanahalli, P. Panchamukhi, J. Keniston, A. Keshavamurthy, and M. Hiramatsu, “Probing the guts of kprobes,” in *Linux Symposium*, vol. 6, 2006, p. 5.
- [206] A. R. Bernat and B. P. Miller, “Anywhere, any-time binary instrumentation,” in *Proceedings of the 10th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools*, ser. PASTE ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 9–16. [Online]. Available: <https://doi.org/10.1145/2024569.2024572>
- [207] K. Peter, “x86 machine code statistics,” 2009. [Online]. Available: https://www.strchr.com/x86_machine_code_statistics
- [208] C. Harper-Cyr, M. R. Dagenais, and A. S. Bushehri, “Fast and flexible tracepoints in x86,” *Software: Practice and Experience*, vol. 49, no. 12, pp. 1712–1727, 2019. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2746>
- [209] O. Dion, “Points de trace dynamiques, portables et extensibles en espace utilisateur avec un surcoût minimal et une couverture maximale,” Master’s thesis, Polytechnique Montréal, April 2023. [Online]. Available: <https://publications.polymtl.ca/53343/>
- [210] M. Hiramatsu and S. Oshima, “Djprobe kernel probing with the smallest overhead,” in *Linux Symposium*, 2007, p. 189.

- [211] “[patch] linux kernel markers,” 2006. [Online]. Available: <https://linux.kernel.narkive.com/FDc9TB0d/patch-linux-kernel-markers>
- [212] , “Studies on compiler controlled cache coherency for multicore processors,” Ph.D. dissertation, Waseda University, 2020.
- [213] “GNU Compiler Collection,” <https://gcc.gnu.org/>.
- [214] “LLVM Compiler Infrastructure,” <https://llvm.org/>.
- [215] J. Chen and C. Suo, “Boosting compiler testing via compiler optimization exploration,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, aug 2022. [Online]. Available: <https://doi.org/10.1145/3508362>
- [216] M. Zhu and D. Hao, “Compiler auto-tuning via critical flag selection,” in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 1000–1011.
- [217] F. M. Lins, “The effects of the compiler optimizations in embedded processors reliability,” 2017.
- [218] D. Pizzolotto and K. Inoue, “Identifying compiler and optimization level in binary code from multiple architectures,” *IEEE Access*, vol. 9, pp. 163 461–163 475, 2021.
- [219] R. Brum, M. de Castro, and C. Faria, “The impact of compilation flags and choosing single-or-double-precision variables in linear systems solvers,” *Trends in Computational and Applied Mathematics*, vol. 24, pp. 319–335, 2023.
- [220] Z. Guo, Z. He, and Y. Zhang, “Mira: A program-behavior-guided far memory system,” in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 692–708. [Online]. Available: <https://doi.org/10.1145/3600006.3613157>
- [221] Z. He, “Towards a transparent and efficient far memory system,” Ph.D. dissertation, UC San Diego, 2023.
- [222] M. Ismail, Y. Lin, D. Han, and D. Gao, “Binalign: Alignment padding based compiler provenance recovery,” in *Information Security and Privacy*, L. Simpson and M. A. Rezazadeh Bae, Eds. Cham: Springer Nature Switzerland, 2023, pp. 609–629.
- [223] C. LeDoux, M. Sharkey, B. Primeaux, and C. Miles, “Instruction embedding for improved obfuscation,” in *Proceedings of the 50th Annual Southeast Regional*

- Conference*, ser. ACM-SE '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 130–135. [Online]. Available: <https://doi.org/10.1145/2184512.2184543>
- [224] C. Jämthagen, P. Lantz, and M. Hell, “A new instruction overlapping technique for anti-disassembly and obfuscation of x86 binaries,” in *2013 Workshop on Anti-malware Testing Research*, Oct 2013, pp. 1–9.
- [225] L. Zhang, P. Liu, Y.-H. Choi, and P. Chen, “Semantics-preserving reinforcement learning attack against graph neural networks for malware detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1390–1402, 2023.
- [226] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, *Deceiving ML-Based Friend-or-Foe Identification for Executables*. Cham: Springer International Publishing, 2023, pp. 217–249. [Online]. Available: https://doi.org/10.1007/978-3-031-16613-6_10
- [227] M. Wong, E. Raff, J. Holt, and R. Netravali, “Marvolo: Programmatic data augmentation for deep malware detection,” in *Machine Learning and Knowledge Discovery in Databases: Research Track*, D. Koutra, C. Plant, M. Gomez Rodriguez, E. Baralis, and F. Bonchi, Eds. Cham: Springer Nature Switzerland, 2023, pp. 270–285.
- [228] J. D. Herath, P. P. Wakodikar, P. Yang, and G. Yan, “Cfgexplainer: Explaining graph neural network-based malware classification from control flow graphs,” in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 172–184.
- [229] H. Benkraouda, J. Qian, H. Q. Tran, and B. Kaplan, “Attacks on visualization-based malware detection: Balancing effectiveness and executability,” in *International Workshop on Deployable Machine Learning for Security Defense*. Springer, 2021, pp. 107–131.
- [230] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, “Malware makeover: Breaking ml-based static analysis by modifying executable bytes,” in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 744–758. [Online]. Available: <https://doi.org/10.1145/3433210.3453086>
- [231] The Linux Foundation. Ftrace: The Linux Kernel Tracer. [Online]. Available: <https://www.kernel.org/doc/html/v5.0/trace/ftrace.html>
- [232] “Uftrace,” <https://uftrace.github.io/slide/#1>.

- [233] B. C. K. Don, *Reducing Overheads of Binary Instrumentation on X86-64: Techniques and Applications*. Indiana University, 2020.
- [234] D. M. Berris, A. Veitch, N. Heintze, E. Anderson, and N. Wang, “Xray: A function call tracing system,” Tech. Rep., 2016, a white paper on XRay, a function call tracing system developed at Google.
- [235] B. Chamith, B. J. Svensson, L. Dalessandro, and R. R. Newton, “Living on the edge: Rapid-toggling probes with cross-modification on x86,” in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 16–26. [Online]. Available: <https://doi.org/10.1145/2908080.2908084>
- [236] S. J. Fink and Feng Qian, “Design, implementation and evaluation of adaptive recompilation with on-stack replacement,” in *International Symposium on Code Generation and Optimization, 2003. CGO 2003.*, March 2003, pp. 241–252.
- [237] R. K. Lebedev, “Using 86 mode switching for program code protection,” , no. 61, pp. 104–120, 2023.
- [238] “desync-cc: A research tool for automatically applying disassembly desynchronization during compilation,” *Science of Computer Programming*, vol. 228, p. 102954, 2023.
- [239] J. Lin, D. Wang, R. Chang, L. Wu, Y. Zhou, and K. Ren, “Enbindiff: Identifying data-only patches for binaries,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 343–359, 2023.
- [240] R. Qiao and R. Sekar, “Function interface analysis: A principled approach for function recognition in cots binaries,” in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 201–212.
- [241] A. Balboul, “Points de trace rapides et efficaces par injection adaptative de sauts en x86,” Master’s thesis, Polytechnique Montréal, May 2020. [Online]. Available: <https://publications.polymtl.ca/5271/>
- [242] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [243] Gabriele Paoloni, Intel Corporation, *How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures*, 2010, no. 324264-001.

- [244] “Paradyn parallel performance tools,” Tech. Rep., 2016, dyninst Programmer’s Guide.
- [245] D. Nadeau, N. Ezzati-Jivan, and M. R. Dagenais, “Efficient large-scale heterogeneous debugging using dynamic tracing,” *Journal of Systems Architecture*, vol. 98, pp. 346 – 360, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1383762118301838>
- [246] M. Gebai and M. R. Dagenais, “Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead,” *ACM Comput. Surv.*, vol. 51, no. 2, pp. 26:1–26:33, Mar. 2018. [Online]. Available: <http://doi.acm.org/10.1145/3158644>
- [247] Y. Padarthy and R. R. P. Kuppusamy, “Iot-based embedded sensor system for real-time health monitoring of composite structures for large-scale industrial operations.” CRC Press, 2023.
- [248] D. Zhai, C. Wang, R. Zhang, H. Cao, and F. R. Yu, “Energy-saving deployment optimization and resource management for uav-assisted wireless sensor networks with noma,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 6, pp. 6609–6623, 2022.
- [249] A. Katal, S. Dahiya, and T. Choudhury, “Energy efficiency in cloud computing data centers: a survey on software technologies,” *Cluster Computing*, 2022.
- [250] J. Chen, M. Manivannan, M. Abduljabbar, and M. Pericàs, “Erase: Energy efficient task mapping and resource management for work stealing runtimes,” 2022.
- [251] P. Mishra, N. Kumar, and W. W. Godfrey, “An evolutionary computing-based energy-efficient solution for iot-enabled software-defined sensor network architecture,” *International Journal of Communication Systems*, 2022.
- [252] J. Toledo-Castro, P. Caballero-Gil, N. Rodríguez-Pérez, I. Santos-González, C. Hernández-Goya, and R. Aguasca-Colomo, “Forest fire prevention, detection, and fighting based on fuzzy logic and wireless sensor networks,” *Complexity*, 2018.
- [253] R. Pachlor and D. Shrimankar, “Vch-eccr: A centralized routing protocol for wireless sensor networks,” *Journal of Sensors*, 2017.
- [254] G. M. Abdulsahib and O. I. Khalaf, “Accurate and effective data collection with minimum energy path selection in wireless sensor networks using mobile sinks,” *Journal of Information Technology Management*, 2021.

- [255] D. E. Ruíz-Guirola, C. A. Rodríguez-López, S. Montejo-Sánchez, R. D. Souza, and M. A. Imran, “Drx-based energy-efficient supervised machine learning algorithm for mobile communication networks,” *IET Communications*, 2021.
- [256] O. Khalaf, C. Romero, S. Hassan, and M. T. Iqbal, “Mitigating hotspot issues in heterogeneous wireless sensor networks,” *Journal of Sensors*, 2022.
- [257] B. A. Begum and S. V. Nandury, “Data aggregation protocols for wsn and iot applications – a comprehensive survey,” *Journal of King Saud University - Computer and Information Sciences*, 2023.
- [258] J. L. Berral, I. n. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavalda, and J. Torres, “Towards energy-aware scheduling in data centers using machine learning.” Association for Computing Machinery, 2010.
- [259] L. Lei, B. Wu, X. Fang, L. Chen, H. Wu, and W. Liu, “A dynamic anomaly detection method of building energy consumption based on data mining technology,” *Energy*, 2023.
- [260] A. Copiaco, Y. Himeur, A. Amira, W. Mansoor, F. Fadli, S. Atalla, and S. S. Sohail, “An innovative deep anomaly detection of building energy consumption using energy time-series images,” *Engineering Applications of Artificial Intelligence*, 2023.
- [261] J. Kong, W. Jiang, Q. Tian, M. Jiang, and T. Liu, “Anomaly detection based on joint spatio-temporal learning for building electricity consumption,” *Applied Energy*, 2023.
- [262] E. Shamsa, A. Kanduri, P. Liljeberg, and A. M. Rahmani, “Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms,” *IEEE Transactions on Computers*, 2021.
- [263] E. Shamsa, A. Kanduri, A. M. Rahmani, and P. Liljeberg, “Energy-performance co-management of mixed-sensitivity workloads on heterogeneous multi-core systems,” 2021.
- [264] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, “Bayesian optimization with machine learning algorithms towards anomaly detection,” 2018.
- [265] A. Belkhiri, A. Shahnejat Bushehri, F. Gohring de Magalhaes, and G. Nicolescu, “Transparent trace annotation for performance debugging in microservice-oriented systems (work in progress paper),” 2023.

- [266] A. Belkhiri, A. S. Bushehri, and M. Dagenais, "Virtual networks link-layer topologies discovery through host-based tracing." IEEE, 2022.
- [267] A. Belkhiri, M. Pepin, M. Bly, and M. Dagenais, "Performance analysis of dpdk-based applications through tracing," *Journal of Parallel and Distributed Computing*, 2023.
- [268] A. Belkhiri and M. Dagenais, "Diagnostic and troubleshooting of openflow-enabled switches using kernel and userspace traces," *International Journal of Communication Systems*, 2021.
- [269] A. Sgueglia, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of iot time series anomaly detection solutions," *Future Generation Computer Systems*, 2022.
- [270] A. Malki, E.-S. Atlam, and I. Gad, "Machine learning approach of detecting anomalies and forecasting time-series of iot devices," *Alexandria Engineering Journal*, 2022.
- [271] M. Bor and U. Roedig, "Lora transmission parameter selection," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2017.
- [272] A. F. X. Glória and P. J. A. Sebastião, "Autonomous configuration of communication systems for iot smart nodes supported by machine learning," *IEEE Access*, 2021.
- [273] A. F. X. d. Glória, "Sustainable modular iot solution for smart cities applications supported by machine learning algorithms," 2021.
- [274] B. C. D. Dias, "Heterogeneous communication scheme for iot smart nodes," Ph.D. dissertation, 2021.
- [275] H. K. Yugank, R. Sharma, and S. H. Gupta, "An approach to analyse energy consumption of an iot system," *International Journal of Information Technology*, 2022.
- [276] "Large scale survey for radio propagation in developing machine learning model for path losses in communication systems," *Scientific African*, 2023.
- [277] W. Njima, M. Chafii, A. Nimr, and G. Fettweis, "Convolutional neural networks based denoising for indoor localization," 2021.
- [278] A. Gloria, C. Dionisio, G. Simões, and P. Sebastião, "Lora transmission power self configuration for low power end devices," 2019.

- [279] R. Macharia, K. Lang'at, and P. Kihato, "Interference management upon collaborative beamforming in a wireless sensor network monitoring system featuring multiple unmanned aerial vehicles," 2021.
- [280] Z. El Khaled, W. Ajib, and H. Mcheick, "Log distance path loss model: Application and improvement for sub 5 ghz rural fixed wireless networks," *IEEE Access*, 2022.
- [281] Y. Maleh, A. Ezzati, Y. Qasmaoui, and M. Mbida, "A global hybrid intrusion detection system for wireless sensor networks," *Procedia Computer Science*, 2015.
- [282] A. Yahyaoui, T. Abdellatif, and R. Attia, "Read: Reliable event and anomaly detection system in wireless sensor networks," 2018.
- [283] C. Harper-Cyr, M. R. Dagenais, and A. S. Bushehri, "Fast and flexible tracepoints in x86," *Software: Practice and Experience*, 2019.
- [284] M. Schaarschmidt, M. Uelschen, and E. Pulvermüller, "Power consumption estimation in model driven software development for embedded systems." 2021.
- [285] —, "Towards power consumption optimization for embedded systems from a model-driven software development perspective." Springer International Publishing, 2022.
- [286] Y. B. Atitallah, J. Mottin, N. Hili, T. Ducroux, and G. Godet-Bar, "A power consumption estimation approach for embedded software design using trace analysis," 2015.
- [287] M. Bourdon, P.-F. Gimenez, E. Alata, M. Kaaniche, V. Migliore, V. Nicomette, and Y. Laarouchi, "Hardware-performance-counters-based anomaly detection in massively deployed smart industrial devices," 2020.
- [288] R. Gassais, N. Ezzati-Jivan, J. M. Fernandez, D. Aloise, and M. R. Dagenais, "Multi-level host-based intrusion detection system for internet of things," *Journal of Cloud Computing*, 2020.
- [289] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, 2022.
- [290] C. Zhang and A. Hindle, "A green miner's dataset: Mining the impact of software change on energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. Association for Computing Machinery, 2014.
- [291] B. Gregg and J. Mauro, *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*. Prentice Hall Press, 2011.

- [292] A. Jaleel, R. Cohn, C. Luk, and B. Jacob, “Cmp \$ im : A pin-based onthe-fly multi-core cache simulator,” 2008.
- [293] A. SANTOS FILHO, R. J. RODRÍGUEZ, and E. L. FEITOSA, “Evasion and countermeasures techniques to detect dynamic binary instrumentation frameworks,” 2021.