Springer Series in Reliability Engineering

Long Wang · Karthik Pattabiraman · Catello Di Martino · Arjun Athreya · Saurabh Bagchi *Editors*

System Dependability and Analytics

Approaching System Dependability from Data, System and Analytics Perspectives

Springer

Springer Series in Reliability Engineering

Series Editor

Hoang Pham, Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ, USA

Today's modern systems have become increasingly complex to design and build, while the demand for reliability and cost effective development continues. Reliability is one of the most important attributes in all these systems, including aerospace applications, real-time control, medical applications, defense systems, human decision-making, and home-security products. Growing international competition has increased the need for all designers, managers, practitioners, scientists and engineers to ensure a level of reliability of their product before release at the lowest cost. The interest in reliability has been growing in recent years and this trend will continue during the next decade and beyond.

The Springer Series in Reliability Engineering publishes books, monographs and edited volumes in important areas of current theoretical research development in reliability and in areas that attempt to bridge the gap between theory and application in areas of interest to practitioners in industry, laboratories, business, and government.

Indexed in Scopus and EI Compendex

Interested authors should contact the series editor, Hoang Pham, Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ 08854, USA. Email: hopham@rci.rutgers.edu, or Anthony Doyle, Executive Editor, Springer, London. Email: anthony.doyle@springer.com.

Long Wang · Karthik Pattabiraman ·
Catello Di Martino · Arjun Athreya ·
Saurabh Bagchi
Editors

System Dependability and Analytics

Approaching System Dependability from Data, System and Analytics Perspectives

Springer

Editors
Long Wang
Tsinghua University
Beijing, China

Catello Di Martino
Nokia Bell Labs
São Paulo, Brazil

Saurabh Bagchi
Purdue University
West Lafayette, IN, USA

Karthik Pattabiraman
University of British Columbia
Vancouver, BC, Canada

Arjun Athreya
Mayo Clinic
Rochester, MN, USA

ISSN 1614-7839 ISSN 2196-999X (electronic)
Springer Series in Reliability Engineering
ISBN 978-3-031-02062-9 ISBN 978-3-031-02063-6 (eBook)
https://doi.org/10.1007/978-3-031-02063-6

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Introduction

The idea of this book was born at the end of 2019, when we celebrated Professor Ravishankar K. Iyer's 70-year-old birthday. Professor Ravishankar K. Iyer is George and Ann Fisher Distinguished Professor in the Department of Electrical and Computer Engineering at University of Illinois at Urbana-Champaign (UIUC), Urbana, Illinois, USA. He has been our Ph.D. or Postdoctoral Advisor, and importantly, a lifelong mentor to us.

Professor Iyer has made seminal contributions to multiple sub-areas within the area of computing system dependability spanning his over 40-year career. And inspiringly, he is continuing to make more path-defining contributions. Therefore, this book took shape as reviewing some of the most important technical achievements in **four dominant themes in dependability**, namely *software dependability*, *large-scale systems and data analytics*, *healthcare and cyber-physical systems*, and *dependability assessment*. Each section is both a look back and a look forward. The look back describes the important milestones, several from the authors of the chapters, as well as detours on the way to the milestones. The look forward defines important open challenges, which are both relevant and technically challenging, needing concerted efforts from the community. Hopefully, this book will serve as a "call to arms" to the community to pick up some of these problems and to solve them.

Fittingly, we have a section with personal reflections from colleagues who have known Prof. Iyer well. The fact that they happen to be towering researchers in their own right adds more weight to these reflections. These reflections offer a view rarely seen in public documents and will, we hope, serve to inspire a fresh generation of researchers in the field of dependability and beyond.

Each section begins with a chapter, written by one of us, introducing the rest of the chapters in that section, and providing a broad perspective on the theme profiled in that section. These introductory chapters can serve as a guidepost for the reader wishing to selectively navigate through the chapters in the book.

Topic of Dependable Computing Systems

Dependability has long been studied in computer science and engineering—our premier conference, **IEEE/IFIP Dependable Systems and Networks**, or **DSN**, had its start in 1970. The importance of this area is understandable since human safety and well-being have long depended on computing and engineered systems. Research on computer system dependability has led to innumerable successes in fields as varied as follows: *aviation and space* (NASA was one of the early organizations that emphasized dependable computing), *supercomputing clusters*, *banking and finance*, *electric power*, *transportation*, and *distributed computing clusters.* As dependability earned more successes, we ventured into the construction of more complex large systems such as cloud platforms, big autonomous IT infrastructures, and the Internet of Things (IoT).

This book is titled *System Dependability and Analytics* to emphasize its focus on system dependability, rather than only of its component pieces, as well as its intersection with data-driven analytics and machine learning. This latter aspect is becoming increasingly important at a rapid pace. The impetus is coming from large amounts of data being generated by our systems, which are being analyzed for understanding dependability weaknesses and for mitigating effects of dependability failures. The field is growing, and we expect many foundational as well as applied advances to come in the next few years. This book is an early attempt to chart that course, though doubtless, there is a good deal of speculation involved in our charting activity.

Staging of Dependability Topics

In the early stage of his research career, Prof. Iyer worked on analysis of dependability data and building of dependability models from the data. Subsequently, he worked on the design of dependability technologies and measurement of system dependability. In the recent decade or so, his research focus has moved onto analytics-driven approaches to dependability, including a prominent focus on dependability in genomics and autonomous transportation. Correspondingly, this book features the four sections that approximately cover these themes. It also makes sense that Prof. Iyer's dependability research started with modeling and measurement and then steered toward application to use cases, as the models and measurement techniques gained maturity. Thus, his career exemplifies the synergistic relationship that should ideally exist between theory and practice. In terms of the target systems for the dependability techniques, Ravi's work spans a long arc. Correspondingly, this book follows such an arc covering dependability of mainframes (early era) to that of supercomputers and software systems, to analytics of healthcare systems, and now to CPS and autonomous systems.

We start off with the theme of software dependability where we look at software that goes in small to large devices. Then, we move to the dependability of large-scale systems and the aspect of data analytics introduced above. Next, we delve into the impact of dependability on healthcare and cyber-physical systems (CPS), two relatively recent but already highly impactful sub-areas. We then come to the topic of how to assess if our dependability design meets its goals or not. We end the book with personal reflections on Ravi from three of his colleagues at the University of Illinois at Urbana-Champaign.

Goals

By reading this book, the reader will obtain an understanding of leading-edge dependability techniques in the diverse areas of software, large-scale systems and data analytics, healthcare and CPS, and dependability assessment techniques. These are grouped into four corresponding sections of the book. The book does not aim for completeness of the coverage of these topics. Rather, it provides influential techniques that have strong theoretical foundations and, in many cases, have proven to be of practical value in real-world systems.

The contributors of this book are active researchers and practitioners in leading universities and research laboratories. They conduct research and build real-world systems, services, and technologies in the areas covered in this book. In the book, they bring forward their deep insights and provide their contemporary views and visions on dependability. Thus, researchers, professional practitioners, and graduate students will all obtain a clear-eyed view of the state of the art of the research and real-world practice of system dependability and analytics.

Biographical Note on Prof. Ravishankar K. Iyer

Professor Ravishankar K. Iyer is ACM Fellow, IEEE Fellow, AAAS Fellow, and served as Interim Vice Chancellor of UIUC for research during 2008–2011. He has received several awards, including the IEEE Emanuel R. Piore Award, and the 2011 ACM Outstanding Contributions award. He has supervised about 40 Ph.D. dissertations over his distinguished career.

Long Wang
Karthik Pattabiraman
Catello Di Martino
Arjun Athreya
Saurabh Bagchi

Contents

Software Dependability

Introduction: Software Dependability 3
Long Wang

Intelligent Software Engineering for Reliable Cloud Operations 7
Michael R. Lyu and Yuxin Su

Data Analytics: Predicting Software Bugs in Industrial Products 39
Robert Hanmer and Veena Mendiratta

From Dependability to Security—A Path in the Trustworthy Computing Research 55
Shuo Chen

Assessment of Security Defense of Native Programs Against Software Faults 69
Keun Soo Yim

Multi-layered Monitoring for Virtual Machines 99
Cuong Pham

Security for Software on Tiny Devices 141
Saurabh Bagchi

Large-Scale Systems and Data Analytics

Introduction: Large-Scale Systems and Data Analytics 163
Saurabh Bagchi

On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks 167
Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi

Providing Compliance in Critical Computing Systems 191
Long Wang

Application-Aware Reliability and Security: The Trusted Illiac Experience 207
Karthik Pattabiraman

Mining Dependability Properties from System Logs: What We Learned in the Last 40 Years 221
Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia

Critical Infrastructure Protection: Where Convergence of Logical and Physical Security Technologies is a Must 239
Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano

Health Care and CPS

Introduction: Cyber Physical Systems and Healthcare Analytics 257
Arjun P. Athreya

On Improving the Reliability of Power Grids for Multiple Power Line Outages and Anomaly Detection 259
Jie Wu, Jinjun Xiong, and Yiyu Shi

Domain-Specific Security Approaches for Cyber-Physical Systems 301
Hui Lin

Uniting Computational Science with Biomedicine: The NSF Center for Computational Biotechnology and Genomic Medicine (CCBGM) 323
Liewei Wang and Richard M. Weinshilboum

Data-Driven Approaches to Selecting Samples for Training Neural Networks 327
Murthy V. Devarakonda

Classifying COVID-19 Variants Based on Genetic Sequences Using Deep Learning Models 347
Sayantani Basu and Roy H. Campbell

Twenty-First Century Cybernetics and Disorders of Brain and Mind 361
Gregory Worrell

Dependability Assessment

Introduction: Dependability Assessment 369
Karthik Pattabiraman

Effect of Epistemic Uncertainty in Markovian Reliability Models 371
Hiroyuki Okamura, Junjun Zheng, Tadashi Dohi, and Kishor S. Trivedi

System Dependability Assessment—Interplay Between Research and Practice 393
Mohamed Kaâniche and Karama Kanoun

Assessing Dependability of Autonomous Vehicles 405
Saurabh Jha

Personal Reflections

Foreword: Computing and Genomics at Illinois 425
Gene E. Robinson

An Academic Life Begins and Continues at University of Illinois at Urbana-Champaign 429
Janak H. Patel

Learning from Prof. Iyer 431
Wen-Mei Hwu

Software Dependability

Introduction: Software Dependability

Long Wang

Abstract This is the introduction of the 6 chapters in this "software dependability" section. Threats to software dependability are getting aggravated as more complex software and systems are being used and hardware devices with thinner MOSFET channel lengths are being used. This section presents 6 state-of-the-art work that demonstrate a few trends in software dependability research: popular use of data-driven AI, blurring limits between software dependability and security, and software dependability and security in emerging computing environments. The audience will get an up-to-date view of the software dependability research, especially its ongoing trends, after reading this section.

Keywords Dependability · Security · Blurring limit

Information technology (IT) is rapidly expanding its application scope and spreading into more critical domains such as electric power management, transportation traffic regulation and public health, in addition to the traditional domains of scientific computing, office business, finance and telecommunication, etc. Large computing platforms such as cloud systems and artificial intelligence (AI) platforms, and large networks such as internet-of-things (IoT) network are emerging as key computing infrastructures that host IT services. As a result, the complexities of software running on these modern computing systems have been increasing by a lot.

The rapid spread of software into broader critical domains and the increasing complexities of software demand high dependability of software. Moreover, hardware devices underlying computing systems are using MOSFET (or similar technologies) devices with very thin channel length (5 nm, or thinner expected in near future), which give rise to a much larger amount of soft errors in computing systems. This issue further aggravates the software dependability problem, and demands more focus be placed on software dependability in modern computing systems. However, the rapid progress of IT technologies also brings new capabilities of improving software dependability.

L. Wang (✉)
Tsinghua University, Beijing, China
e-mail: longwang@tsinghua.edu.cn

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_1

This section presents a select set of state-of-the-art work that demonstrate a few trends in software dependability research now. (i) One recent principal thrust addressing software dependability is through data-driven AI, including machine learning based on deep neural network, data analytics, and various classification techniques. (ii) Another trend is the blurring limits between software dependability and software security. Specifically, a number of technologies originally proposed and traditionally applied for software dependability are recently applied for software security and have demonstrated their significance in addressing security issues. Examples include bit flip injection, fuzzing (exploration of various inputs for tests), formal method, distributed consensus and monitoring technologies. As the limits between software dependability and security get blurring a new gate is open, and a number of technology advancements are being proposed and then employed in practice. (iii) Software dependability and security in emerging computing environments such as cloud systems and IoT environments are also hot topics recently.

The first two articles of this section demonstrate two good examples on how data-driven AI is adopted for addressing software dependability issues. *Intelligent Software Engineering for Reliable Cloud Operations*, authored by Prof. Lyu and Prof. Su, describes an AIOps (Artificial Intelligence for IT Operations) framework that employs AI technologies for anomaly detection in cloud systems. The framework leverages existing monitoring data of a cloud, particularly Key Performance Indicators (KPIs) data such as CPU usages of VMs, packet loss rates, packet error rates, etc., and applies neural network models to do anomaly detection and generate system incidents. Then the framework applies Graph Representative Learning algorithms to cluster and aggregate the incidents for failure diagnosis and root cause analysis. Hanmer and Prof. Mendiratta's *Data Analytics: Predicting Software Bugs in Industrial Products* presents a survey of software bug prediction techniques and a case study that employs source code complexity metrics, such as percent branch statements, block depth, line number of deepest block, statements at block level 0, to do bug prediction. The proposed technique in the case study uses Random Forest for the prediction. The two articles show that AI has demonstrated its super powerful capabilities in identifying patterns in complicated data, and such capabilities greatly help with anomaly detection, failure diagnosis, and error prediction.

The following three articles are examples that show blurring limits between software dependability and software security. Dr. Chen's *From dependability to security—a path in the trustworthy computing research* provides enlightenments on the relationships between dependability and security, between faults and attacks, by virtue of the author's own experience. Dependability and security are discussed in context of a common adversary model. Particularly, "bit flips", "formal methods" and "distributed consensus" are discussed as the main instruments used for both dependability and security (actually most of them, if not all, were proposed and applied first for dependability, and then repurposed for security). *Assessment of Security Defense of Native Programs Against Software Faults* by Dr. Yim studies security defense of C/C++ programs against faults. Faults and attacks, though they are two distinct adversaries of programs, are related in that faults, e.g. bit flips, may cause consequences of security breaches. This article conducts experimental studies of

"exploitable software faults", the software faults that can be exploited to result in security breaches, and shows both the capability of the fuzzing technology in finding exploitable software faults and the built-in security defense capability of programs against exploitable software faults. The article exposes interesting insights on how security-oriented exploitation and reliability faults are related. *Multi-layered Monitoring for Virtual Machines* by Dr. Pham describes a solution of VM monitoring for both reliability and security purposes. The solution covers all layers from hardware and hypervisor up to applications. It provides a quite comprehensive description of VM monitoring technologies. The audience will understand the challenges, pros and cons of VM monitoring technologies after reading this article. The three articles are part of the ongoing efforts that combine dependability research and security research.

The last article in this section, Prof. Bagchi's *Security for Software on Tiny Devices*, presents research challenges and potential approaches for providing security to software running on IoT devices. This is a very good introduction on software security on IoT devices. The unique challenges are clearly stated, and the discussions in the article span analysis techniques and algorithms, the enforcement of IoT software security that implements the analysis techniques and algorithms, and measurements, metrics and evaluations of IoT software security. The audience will obtain a clear view of state-of-the-art of the IoT software security from the article.

In summary, this section focuses on software dependability and presents a select set of state-of-the-art work on it. The audience of the section will get an up-to-date view of the software dependability research, especially its ongoing trends. This view is very important today as software dependability is gaining an unprecedented demand while undergoing a drastic change. Both are brought about by the wide and rapid adoption of technology advancements in cloud computing, AI, and other areas: IT services (and software) are growingly supporting more applications and scenarios including many in the critical domains such as public health, transportation traffic regulation and driving of vehicles, where traditionally IT technologies were not largely involved; at the same time, the technology advancements give rise to new approaches, many drastically different from traditional ones, to addressing software dependability issues.

Intelligent Software Engineering for Reliable Cloud Operations

Michael R. Lyu and Yuxin Su

Abstract Reliable Cloud operations are vital to our daily lives because many popular modern software systems are deployed in cloud systems. In this chapter, we discuss our experience in developing an AIOps (Artificial Intelligence for IT Operations) framework to improve the reliability of large-scale cloud systems with intelligence software engineering techniques. The comprehensive AIOps framework includes anomaly detection of key performance indicators, service dependency mining for failure diagnosis, and system incident aggregation for root cause analysis from various information sources like meter data, topology, alert, and incident tickets. We also conduct extensive experiments with production data collected from large-scale Huawei Cloud systems to demonstrate the effectiveness of intelligent software engineering techniques for reliable cloud operations.

1 Introduction

Modern software systems provide convenient services to our daily lives. In particular, IT enterprises start to deploy their applications and services on cloud computing platforms, such as search engines, instant messaging apps, and online shopping. Worldwide public cloud service revenue enjoys an impressive growth, as predicted by Gartner to reach 364 billion US dollars by 2022 [13].

Cloud services are large-scale distributed applications running across thousands of servers within datacenters. The most critical infrastructure of cloud computing is the data centers around the world. Data centers are massive hardware and software systems containing millions of servers, with high-speed interconnection networks. Each server is composed of hardware devices like CPU and memory, which runs an OS or virtual machine on top to manage the hardware resources. Software systems

M. R. Lyu (✉)
The Chinese University of Hong Kong, Hong Kong, Hong Kong
e-mail: lyu@cse.cuhk.edu.hk

Y. Su
Sun Yat-sen University, Guangzhou, China
e-mail: suyx35@mail.sysu.edu.cn

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_2

as cloud services are large-scale distributed applications running across thousands of servers within data centers.

As modern software systems have grown to an unprecedented scale, the tremendous complexity, scaling, and stringent performance of datacenter operations bring significant reliability challenges. Any cloud outage or breakdown will cause significant revenue loss, and harm customer trust and company reputation to cloud providers and service providers [6, 16]. According to Lloyd's report [21], a major failure that brings cloud outage for 3–6 d could result in a total loss of 19 billion US dollars revenue, most of which is not be covered by insurance. Worst of all, in a society highly dependent on IT infrastructure, cloud outages can affect everybody's life just like power outages. To this end, cloud resilience is of paramount importance.

Unfortunately, cloud failures leading to performance degradation or service interruptions have often occurred to major cloud operators. Cloud reliability issues are mainly due to the fact that tough cloud failures take a long time to mitigate manually. Cloud systems are actively undergoing continuous feature upgrades and system evolution by DevOps [9] paradigm, complex service dependency, load balance, and recovery procedures such as backup and restore; therefore, the statistical properties of system monitoring data may change from time to time. On-call engineers from different sectors equipped with multi-location, multi-source and multi-layer components have their specific responsibilities. Overall, the real root cause of cloud failures is hard to locate.

Traditionally, Software Reliability Engineering (SRE) aims to solve software reliability challenges by providing reliability models to track software failures. The tracked failure rates enable engineers to predict software reliability with analytical models using two or three parameters. The *Handbook of Software Reliability Engineering* [22] examined this process, and introduced the techniques to improve software reliability, including *fault avoidance*, *fault removal*, *fault tolerance*, and *fault prediction*.

This traditional analytical approach is not enough for today's complicated cloud software systems since modern cloud systems generate more complex and massive amounts of data concerning software reliability issues. To serve various users, cloud provides flexible infrastructure containing three major layers: *application layer*, *platform layer*, and *infrastructure layer*, displayed in Fig. 1. On-call engineers inspect the status of cloud from application and system logs, meter data generated from multiple components, and alerts triggered by rule-based monitor. Besides, top cloud systems provide customer service to collect most incidents, outages, or dissatisfaction from users. Customer service transfers feedback to on-call engineers. In order to obtain a comprehensive understanding of failures, on-call engineers from different sectors establish a war-room to discuss the problem and try to find possible solutions. This process generates incident tickets.

However, humans are not good at solving complex failure diagnosis problems associated with big data generated from large-scale cloud systems. But Artificial Intelligent (AI) algorithms have the opportunity to solve the complicated problems because AI algorithms are superior to human in big data analysis. For example, the

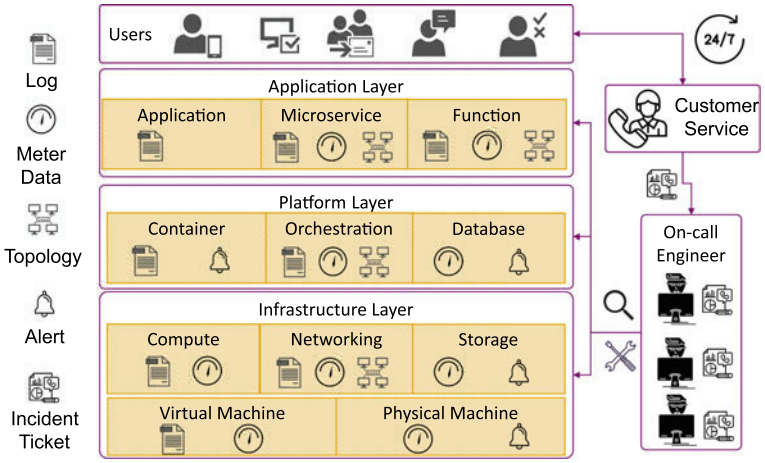


Fig. 1 Cloud systems generate a variety of data

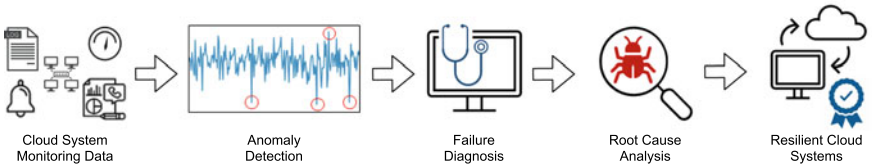


Fig. 2 The overall framework of resilient cloud systems with AIOps

Key Performance Indicator (KPI) "packet number" monitoring the cloud network may suddenly decrease because of anomalies happening in some network services. This may indicate a serious failure in the network. We would like to determine what failures are caused by the anomalies underneath, which is generally indicated by the sudden increase and drop of KPIs. Human maintainers generally assign different importance of system performance to distinct KPIs in the cloud. Generally, when diagnosing failures for large-scale cloud systems, an automated detection model with flexible importance assignment is more precise and quicker to signify the potential root cause than human maintainers.

In this chapter, we describe our experience on the development of AIOps (Artificial Intelligence for IT Operations) framework to tackle several reliability challenges commonly seen in industrial cloud systems. We provide a general end-to-end pipeline of intelligent software engineering illustrated in Fig. 2 to conduct *anomaly detection*, *failure diagnosis*, and *root cause analysis* with multiple sources of heterogeneous information such as meter data, topology, alert and incident ticket. Specifically, the root cause analysis in cloud systems differs from the traditional definition in software reliability engineering that aiming to identify the exact fault of a particular failure. In cloud systems, it is more practical to narrow down the scope of system components

associated with a failure. We also conduct extensive experiments with real-world large-scale cloud systems from Huawei to demonstrate the effectiveness of intelligent software engineering techniques for reliable cloud operations.

2 Anomaly Detection of Key Performance Indicators

2.1 Background

Key Performance Indicators (KPIs) are the most important data in the cloud, which are leveraged to monitor the health status of a machine, like network traffic, response delay and CPU usage. Anomaly detection over the KPIs is a critical tool to ensure the reliability and availability of the system, which aims to discover unexpected events or rare items in data. Different system components (e.g., microservices, servers) are tightly coupled, and cloud failures usually trigger anomaly performance in multiple KPIs. For example, a problematic load balance server is often accompanied by a burst on both round-trip delay and in-bound traffic rate, which will further increase CPU utilization.

Recent studies tackle this problem by constructing an $m \times m$ KPI inner-product matrix [36] or a complete graph [37] for m different KPIs to capture the pairwise KPI interaction, both of which yield an $\mathcal{O}(m^2)$ computation complexity. A real-world example is provided in Fig. 3, which is from a public dataset released by [31]. *CPU LOAD* and *ETH INFLOW* are highly correlated as their curves exhibit a very similar

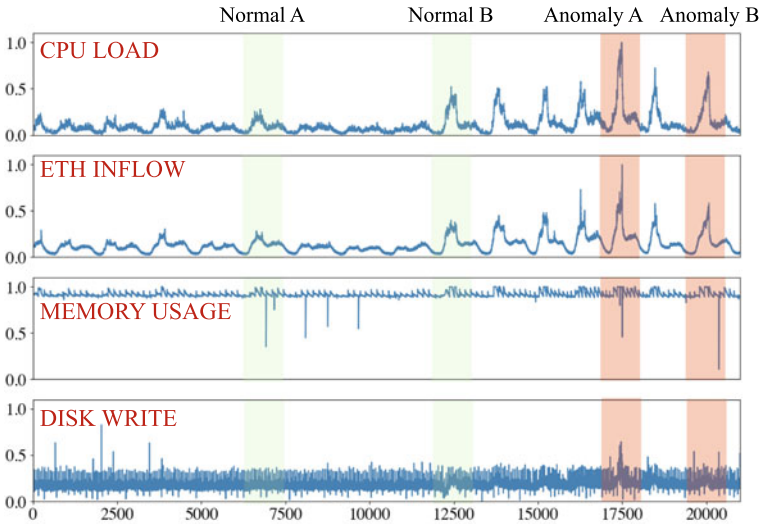


Fig. 3 Multivariate KPIs snippet from server machine dataset

trend. The correlated relationship provides an overall picture of the systems' health status. For example, in the segment marked as *Normal A*, we can see a clear spike in *MEMORY USAGE*, which would be flagged as an anomaly without a glance at the other three KPIs. Similar situation happens to segment *Normal B*, where boots can be clearly seen in both *CPU LOAD* and *ETH INFLOW*. Therefore, we need to consider the full set of multivariate KPIs to pursue an accurate anomaly detection, as shown in segment *Anomaly A* and *Anomaly B*. Besides the dependency between KPIs, we can also leverage historical KPI patterns to reduce false positives. Specifically, in Fig. 3, all KPIs have witnessed some abnormal spikes in history. However, they do not necessarily indicate the occurrence of failures.

In industrial systems, hundreds or even thousands of KPIs are being monitored. The dependencies among KPIs are very sparse, i.e., most KPIs are not or weakly dependent on other KPIs. Therefore, how to automatically learn the dependencies among different KPIs is critical towards efficient multivariate KPI anomaly detection. In the literature, many studies have shifted to anomaly detection on multivariate KPIs, which mainly resorts to different neural network models. For example, Omni-Anomaly [31] proposes to learn the normal patterns of multivariate time series by modeling data distribution through stochastic latent variables. Anomalies are then determined by reconstruction probabilities. Similarly, Malhotra et al. [24] used an LSTM-based (long short-term memory-base) encoder-decoder network to learn time series's normal patterns and Zhang et al. [36] used an attention-based convolutional LSTM network for the learning purpose. Although tremendous progress has been made, we still observe two major limitations of existing approaches: (1) the interactions among KPIs are not explicitly modeled, and (2) the efficiency falls behind industrial needs. Specifically, previous approaches [28, 31] detect anomalies on multivariate KPIs mainly by stacking different types of KPIs into a feature matrix and feeding it to sophisticated neural network models. Different from previous work, we argue that by properly modeling the interactions of KPIs along with feature and temporal dimensions, cost-effective neural network models can be leveraged for anomaly detection.

To overcome the aforementioned limitations, we introduce CMAnomaly illustrated in Fig. 4, which is an efficient unsupervised model for anomaly detection over multivariate KPIs. CMAnomaly consists of four phases, i.e., *data preprocessing*, *collaborative machine*, *model training*, and *anomaly detection*. The first phase preprocesses the data by applying normalization and window sliding. Particularly, the input types of KPIs can vary depending on the application scenario. In the next phase, the preprocessed data are fed to the proposed collaborative machine, which is the core component of CMAnomaly. The collaborative machine can properly capture the interactions among multivariate KPIs along with both feature and temporal dimensions. In the third phase, we train a forecasting-based anomaly detection model [10, 18], which detects anomalies based on prediction errors. Finally, the trained model will be applied to detect anomalies for new observations.

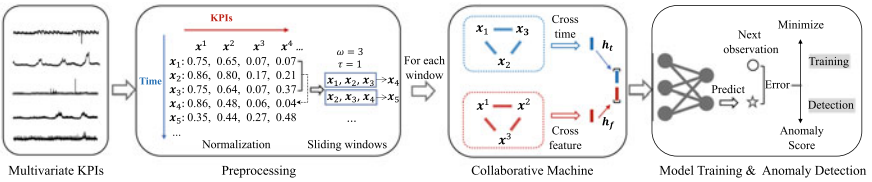


Fig. 4 Overall framework of CMAnomaly

2.2 Preprocessing

The input of multivariate KPIs is denoted as $X \in \mathbb{R}^{n \times m}$, where n is the number of different KPIs and m is the number of observations. The t-th row of X, denoted as $x_t = [x_t^1, x_t^2, \ldots, x_t^m]$, is an m-dimensional vector containing the observation of each KPI at timestamp t. Similarly, the k-th column of X, denoted as $x^k = [x_1^k, x_2^k, \ldots, x_n^k]$, is an n-dimensional vector containing the observations of the k-th KPI. Particularly, we denote $x_{i:j}^k = [x_i^k, x_{i+1}^k, \ldots, x_j^k]$ as a consecutive sequence of x^k from timestamp i to j. The objective of anomaly detection for multivariate KPIs is to determine whether or not a given x_t is anomalous, i.e., whether the entity is in abnormal status at timestamp t. For each timestamp t, our model calculates an anomaly score $s_t \in [0, 1]$, which represents the probability of x_t being anomalous. If s_t is larger than a pre-defined threshold θ, x_t will be predicted as an anomaly. The ground truth $\mathbf{y} \in \mathbb{R}^n$ is an n-dimensional vector consisting 0 and 1, where 0 indicates a normal point, and 1 indicates an anomalous one.

Different KPIs may have distinct scales, for example, the KPI monitoring the CPU execution, i.e., *CPU USAGE*, is in the range of 0% to 100%. However, the KPI monitoring the network traffic, i.e., *INBOUND PACKAGE RATE* can range from zero to millions of kilobytes. Therefore, data normalization is performed for each individual KPI to ensure the robustness of our model. We apply max-min normalization to each individual KPI, i.e., x^k, as follows:

$$x_{norm}^k = \frac{x^k - \min(x^k)}{\max(x^k) - \min(x^k)}, \tag{1}$$

where the values of $max(x^k)$ and $min(x^k)$ are computed in the training data, which will then be used for test data normalization. For simplicity, we omit the "norm" subscript in the following elaboration. The sliding window is to partition KPIs along the temporal dimension. Particularly, it consists of two attributes, i.e., window size ω and stride τ. The stride indicates the forwarding distance of the window along the time axis to generate multivariate KPI windows. As the stride is often smaller than the window size, there exists overlapping between two consecutive windows. We denote the s-th sliding window as:.

$$X_s = [x_{s\tau}, x_{s\tau+1}, \ldots, x_{s\tau+\omega-1}] \tag{2}$$

where $s \in [0, 1, 2, \ldots]$. X_s together with the observations at the next timestamp of the window, i.e., $\hat{x}_s = x_{s\tau+\omega}$, constitute a pair $(X_s, \hat{x}_s)$, where $X_s \in \mathbb{R}^{\omega \times m}$ and $\hat{x}_s \in \mathbb{R}^m$.

2.3 Multivariate KPIs Interactions

As shown in Fig. 3, historical patterns of KPIs provide important clues for anomaly detection on multivariate KPIs accurately. To explicitly capture the dependency between multivariate KPIs and their historical patterns, for each sliding window, denoted as $X_s \in \mathbb{R}^{\omega \times m}$, we calculate the pairwise inner product of all KPI feature vectors, i.e., $x^k_{s\tau, s\tau+\omega-1}$, $k \in [1, m]$, and temporal vectors, i.e., x_t, $t \in [s\tau, s\tau + \omega - 1]$.

$$h_f = b_0 + \sum_{i=1}^{m} w_i x^i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} \left\langle x^i, x^j \right\rangle v_i v_j \tag{3}$$

$$h_t = \hat{b}_0 + \sum_{i=1}^{\omega} \hat{w}_i x_i + \sum_{i=1}^{\omega} \sum_{j=i+1}^{\omega} \left\langle x_i, x_j \right\rangle \hat{v}_i \hat{v}_j \tag{4}$$

The cross-feature and cross-time KPI interactions, denoted as h_f and h_t, are formulated as Eqs. 3 and 4, respectively. In Eq. 3, $b_0, w_i, v_j, v_j \in \mathbb{R}$ are trainable parameters, $x^i, x^j \in \mathbb{R}^\omega$ are the i-th and j-th column of X_s with each column representing all the observations of a KPI in the corresponding window, and $< \cdot, \cdot >$ is the operation of inner product. These equations are composed of three terms: the first term is a trainable bias, the second term is a weighted sum of all KPIs without explicit interaction, and the third term is the core part of the proposed collaborative machine, which models the pairwise KPI interactions.

2.4 Collaborative Machine for Anomaly Detection

The last two phases of CMAnomaly are model training and anomaly detection. In the detection phase, the well-trained model predicts the next KPI values given preceding observations. In the training phase, as most multivariate KPIs would reflect the normal status of an entity, the model will learn the normal patterns of KPIs, i.e., what the next observations would be given previous ones. Although there could be anomalies in the training data, they tend to be forgotten by the model as they rarely appear. Consequently, in the detection phase, the model will predict "normal" KPI values based on the learned patterns. If the real observations deviate from the predicted ones by a significant margin, an anomaly may happen, i.e., the entity is not in its normal status. Therefore, such deviation measures the likelihood of the occurrence of the anomaly.

Our framework supports various types of neural network models for anomaly detection. The anomaly detection model can be formulated as follows:

$$\tilde{h}_{i+1} = \sigma(\tilde{h}_i \tilde{X}_i + \tilde{b}_i), i = 0, 1, \ldots, L-1, \tag{5}$$

where L is the number of layers of the Multilayer Perceptron (MLP) model, $\tilde{W}_i$, $\tilde{b}_i$ are trainable parameters with customized size, and $\sigma(x) = max(0, x)$ is the ReLU activation function. We simultaneously consider the cross-feature and cross-time KPI interactions by concatenating h_f and h_t, which is the input to the model, i.e., $\tilde{h}_0 = concat(h_f, h_t)$. $\hat{y} = \tilde{h}_L \in \mathbb{R}^m$ is the prediction result produced by the last layer of the MLP model, which contains the predicted values for all KPIs at the next timestamp.

Anomaly detection model is optimized by minimizing the following mean square error (MSE) loss $\mathcal{L}$ between the predictions and the ground truth observations:

$$\mathcal{L} = \sum_{i=1}^{N} \left\| \hat{y}_i - \hat{x}_i \right\|_2, \tag{6}$$

where N is the number of training sliding windows. $\hat{y}_i \in \mathbb{R}^m$ and $\hat{x}_i = x_{i\tau+\omega} \in \mathbb{R}^m$ are the predicted and ground truth observations for the i-th window, respectively.

With the minimization of loss $\mathcal{L}$ during training, CMAnomaly can learn from the normal patterns in the training data by updating all trainable parameters, e.g., $v_i v_j$ denoting the interaction weights. After the model is trained, we compute an anomaly score for each window X_i in the testing data. Then, we first calculate the MSE between the predicted and ground truth observations, and then apply the sigmoid function to rescale the score to the range [0, 1], which represents the probability of the occurrence of an anomaly:

$$s_i = \phi\left(\frac{1}{m} \left\| \hat{y}_i - \hat{x}_i \right\|_2\right) \tag{7}$$

where $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. To determine whether an anomaly has happened, a threshold θ should be defined for the anomaly score. The timestamps with a large anomaly score, i.e., $s_i \geq \theta$, should be regarded as anomalous points.

In reality, the threshold can be set by on-site engineers based on their experience. A large threshold imposes a strict anomaly detection policy, which may miss important system failures, i.e., low recall. However, a small threshold increases the sensitivity to KPI changes, resulting in false alarms, i.e., low precision.

Table 1 Accuracy comparison on Huawei Cloud dataset

Methods	Precision	Recall	F1
OmniAnomaly	0.6639	0.8382	0.7283
LSTM-VAE	0.8273	0.7436	0.7560
CMAnomlay	**0.9179**	0.8202	**0.8368**

2.5 Experiments

In this part, we evaluate CMAnomaly using both public data and industrial data. We collected real-world KPIs from Huawei Cloud to conduct a more comprehensive evaluation. Huawei Cloud contains a large number of nodes supporting tens of millions of users worldwide. Therefore, to provide a stable 24 × 7 service, the status of each component of the network is closely monitored with KPIs. The engineers can fix problematic components timely if the anomalies of KPIs can be automatically detected and reported in real-time. To evaluate our method in a practical scenario, we collected a 30-day-long KPIs dataset with 13 network components within Jan. 2021. Each of the network components has 70∼200 different types of KPIs. We use the first 20 d of KPIs as the training data and the rest as the testing data. Then, several experienced engineers were invited to manually label the anomalous points in the testing data.

To study the effectiveness of CMAnomaly, we compare its performance with two most effective open-source anomaly detection methods, i.e., LSTM-VAE [28] and OmniAnomaly [31] on the dataset collected from Huawei Cloud.

The experimental results are shown in Table 1. In particular, the precision of OmniAnomaly is the lowest, but the recall is the highest because the complex architecture of OmniAnomaly incurs more trainable parameters, which makes it easier to overfit the training data. Therefore, OmniAnomaly is more sensitive to capture more anomalies but has the most false positive alarms. LSTM-VAE has a more light-weight design than OmniAnomaly, so LSTM-VAE suffers less overfitting. As a result, LSTM-VAE only raises the anomaly score when the new observation deviates more from the prediction. In this case, although higher precision is achieved, LSTM-VAE has the lowest recall because it cannot effectively find all possible anomalies. CMAnomaly can balance precision and recall better and achieves the best F1 score, ∼0.08 higher than the second-best one achieved by LSTM-VAE. The collaborative machine facilitates CMAnomaly to capture the dependency of the training KPIs effectively. Therefore, CMAnomaly avoids overfitting the noisy points existing in the training data, e.g., usual spikes as shown in Fig. 3. CMAnomaly reports a higher anomaly score only when the dependent KPIs are anomalous, thus achieving the highest precision. Moreover, CMAnomaly keeps the sensitivity to detect more true positive samples thanks to its ability to capturing the dependency.

3 Service Dependency Mining for Failure Diagnosis

3.1 Background

Service reliability is one of the key challenges that cloud providers have to deal with. The common practice nowadays is developing and deploying small, independent, and loosely coupled cloud microservices that collectively serve users' requests. The microservices communicate with each other through well-defined APIs. Under this architecture, microservice management frameworks like Kubernetes will be responsible for managing the life cycles of microservices. Developers can focus on the application logic instead of the bothering tasks of resource management and failure recovery. It enables agile development and supports polyglot programming, i.e., microservices developed under different technical stacks can work together smoothly.

However, the loosely coupled nature of microservices makes it difficult for engineers to conduct system maintenance. Different microservices in a large cloud system are usually developed and managed by separate teams. Each team only has access to their own services as well as services that are closely related, which means they only have a local view of the whole system [32]. As a result, failure diagnosis, fault localization, and performance debugging in a large cloud system become more complex than ever [12, 33]. Despite various fault tolerance mechanisms introduced by modern cloud systems, it is still possible for minor anomalies to magnify their impact and escalate into system outages.

Although microservice management frameworks provide automatic mechanisms for failure recovery, unplanned service failures may still cause severe cascading effects. For example, failures of critical services that provide basic request routing functions will impact the invocation of cloud services, slow down request processing, and deteriorate customer satisfaction. Therefore, evaluating the impact of service failures rapidly and accurately is critical to the operation and maintenance of cloud systems. Knowing the scope of the impact, reliability engineers can emphasis on services that have more significant impacts on others.

3.2 Tracing Analysis

For commercial cloud providers, it is crucial to troubleshoot and fix failures in a timely manner because massive user applications may be affected even by a small service failure [4]. In large-scale cloud systems, a request is usually handled by multiple chained service invocations. As clues to defective services are hidden in the intricate network of services, it is difficult for even knowledgeable SRE personnel to keep track of how a request is processed in the cloud system. All the services and dependencies in a cloud system collectively construct a directed graph of services, which is also called a dependency graph. The dependency graph of a cloud system can be very complicated.

Fig. 5 A span generated by the train-ticket benchmark

Span ID	e22f30bdbfd09134
Parent Span ID	b42a04bf18997d5d
Name	ts-preserve-service
Timestamp (μs)	1618589098705000
Duration (μs)	1126
Result	SUCCESS
Trace ID	c0d17d481f47bdd9
Additional Logs	……

Distributed tracing provides an approach to monitor the execution path of each request in a dependency graph. For chained service invocations, e.g., service `A` invokes service `B` and service `B` invokes service `C`, it is important to know the status of each service invocation, including the result, the duration of execution, etc. By adding hooks to the services and microservices of the cloud system, a distributed tracing system [11] can record the contextual information of each service invocation. Such records are called *span logs*, abbreviated as `spans`. A span represents a logical unit of execution that is handled by a microservice in a cloud system. All the spans that serve for the same request collectively form a directed graph of spans. Such directed graph of spans generated by request is called a piece of *trace log*, abbreviated as a `trace`. With a trace, engineers can track how the request propagates through the cloud system. Collectively analyzing the traces of the entire cloud system can help engineers obtain in-depth latency reports that could assist failure diagnosis, fault localization, and surface performance degradation in the cloud system.

Although the actual implementation of distributed tracing system varies a lot, the types of information they record are similar. For clarity, we formally describe the attributes of spans as follows. Suppose we have a trace T composed of spans $\{s_1, s_2, \ldots, s_n\}$, a span $s_i \in T$ contains the following attributes.

s_i^{id} The ID of span s_i,
s_i^{pid} The ID of the parent span of s_i,
s_i^{tid} The ID of the trace that s_i belongs to,
s_i^{name} The name of service/microservice corresponding to s_i,
s_i^{ts} The time stamp of s_i,
s_i^{d} The duration of execution of s_i, and
s_i^{r} The result of execution of s_i.

Figure 5 illustrates a span generated by the train-ticket benchmark [38]. It means that service `ts-preserve-service` was invoked at 04:58 on April 17, 2020. The duration of execution is 1126 μs and the execution result is SUCCESS.

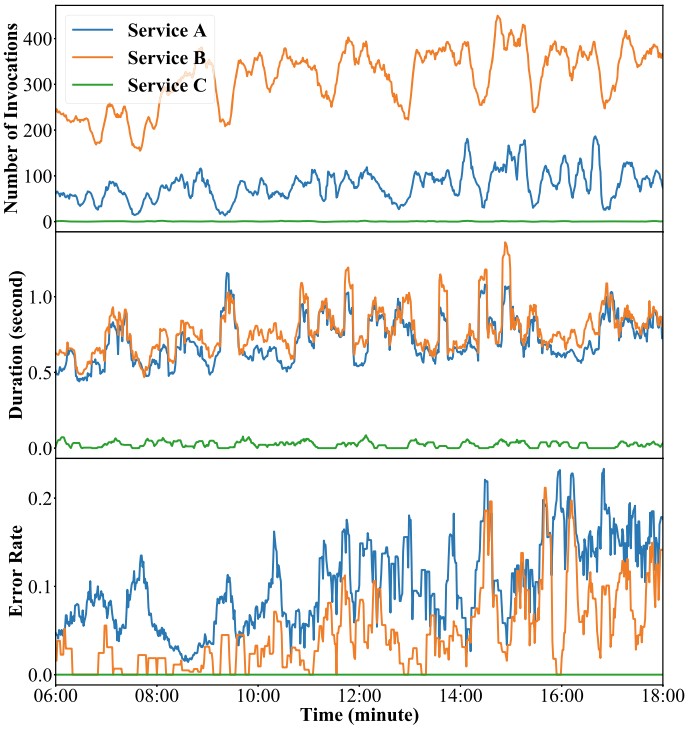


Fig. 6 The statuses of service A, B and C. A invokes B and C but B has a greater effect on A

3.3 Intensity of Service Dependency

Existing tools treat the dependency as a binary relation, i.e., if the caller service invokes the callee service then the caller is dependent on the callee. We suggest that this binary dependency metric is not fine-grained enough for cloud maintenance. Figure 6 shows the statuses of three services[1] A, B, and C in Huawei Cloud. Service A invokes both service B and service C. Service B encountered failures. The x-axis represents time in minute. The y-axes represent the number of invocations per minute, the average duration of invocations per minute, and the error rate per minute of A, B, and C. Although service A invokes service B and service C, it is obvious that the statuses of B and C influence the status of A in different degrees.

The reason is that the functionalities provided by service A and B are creating virtual machines, and allocating block storage, respectively. Creating a virtual machine requires allocating one or more block storage. Thus, the failure of service B inevitably affects service A. On the contrary, due to the fault tolerance mechanism of service A, the failure of service C will not affect service A significantly.

[1] For confidentiality reasons, we cannot reveal the names of the related services.

Thus, it is more accurate to say that the intensity of dependency between service A and service B is higher than the intensity of dependency between service A and service C. Ideally, if the development team of every cloud microservice accurately provide the intensity of dependencies for every dependent service, the failure diagnosis could be accelerated. On-call engineers (OCEs) can prioritize the services that exhibit the higher intensity of dependency instead of inspecting all the dependent services if they have accurate intensity information. However, due to the complexity and the fast evolving nature of cloud systems [2], manually maintaining the dependency relations with intensity is very difficult. As a result, OCEs often struggle in diagnosing failures due to the lack of intensities.

3.4 Dependency Strength Mining

In order to relieve the pressure on OCEs, we introduce a framework called AID [35] to predict the Aggregated Intensity of service Dependency in large-scale cloud systems. The intuition is that direct service invocation incurs direct dependency to some degree. To properly capture service dependency, AID consists of three steps: candidate selection, status generation, and intensity prediction. We will introduce the details in the following parts:

3.4.1 Candidate Selection

Given the raw traces, AID first generates a set of candidate service pairs (P, C) where service P directly invokes service C. In general, direct service invocations can be divided into two categories, i.e., synchronous invocations and asynchronous invocations. Modern tracing mechanisms can keep track of both synchronous and asynchronous invocations [27]. Given all the raw traces of a cloud system, in this step, we generate a candidate dependency set $Cand$. The candidate dependency set $Cand$ contains service invocation pairs $(P_1, C_1), (P_2, C_2), \cdots, (P_n, C_n)$. Each pair (P_i, C_i) in the candidate dependency set denotes that the service named P_i invokes the service named C_i at least once. Therefore, service P_i depends on service C_i. This step is to shrink the search space of possible dependent pairs because the service invocations indicate direct dependencies.

3.4.2 Service Status Generation

The status of one service is composed of three aspects of dependency, i.e., number of invocations, duration of invocations, and error of invocations. Each aspect of the service's status contains one or more KPIs, depending on the actual implementation of the distributed tracing system. As service invocations occur repeatedly, the three statuses of service invocations can derive three aspects of service dependency:

Number of Invocations	The number of invocations from the caller to the callee.
Duration of Invocations	The duration of invocations.
Error of Invocations	The number of successful invocations from the caller to the callee.

Inspired by the common practice in cloud monitoring [1], we distribute the spans of one service into many bins according to the spans' timestamps. Each bin accepts spans whose timestamp is in a short, fixed-length period. We denote the length of the short period as τ. For example, the span shown in Fig. 5 will be put in the bin of `ts-preserve-service` at time 04:58, 17 April 2020. We can then represent the status of a cloud service in a short period by the statistical indicators of all the spans in the corresponding bin. Formally, given all the spans in the cloud system over a long period T, we first initiate $\mathbf{M} \times \mathbf{N}$ empty bins of the predefined size τ. $\mathbf{M}$ is the number of microservices. $\mathbf{N}$, determined by $\frac{T}{\tau}$, is the number of bins. Then we distribute all spans into different bins according to their timestamp s^{ts} and service name s^{name}. After that, we can calculate the following three types of indicators as the KPIs for each bin.

$invo_t^m$	Total number of invocations (spans) in the bin;
err_t^m	Error rate of the bin, i.e., the number of errors divided by the number of invocations;
dur_t^m	Averaged duration of all spans in the bin;

where t is the time of the bin and m is the service name of the bin. If a service is not invoked in a particular bin (i.e., the corresponding bin is empty), all the KPIs will be zero. In this scenario, we obtain the KPIs of every service S at every period t. Ordering the bins by t, we get three time series of KPIs for each cloud service, denoted as $invo^S$, err^S, and dur^S as the status of each cloud service.

3.4.3 Intensity Prediction

The intensity prediction steps quantitatively predict the intensity of dependency by measuring the similarity between two service's statuses. The similarity between two service's statuses is a normalized and weighted average of the similarity of all the KPIs of the two services. We calculate the similarity between two KPIs by a dynamic time warping algorithm (DTW) [19] and aggregate all the similarities to get the overall similarity.

DTW automatically warps the time in chronological order to make the two status series as similar as possible and get the similarity by summing the cost of warping. It utilizes dynamic programming to calculate an optimal matching between two status series. Given two services P, C, and their status series $invo^P$, $invo^C$, err^P, err^C, dur^P, and dur^C, the warping from the callee C to the caller P is specially designed for the cloud environment.

For all $(P_i, C_i) \in Cand$, we calculate similarities between their status series, denoted as $d_{invo}^{(P_i,C_i)}$, $d_{err}^{(P_i,C_i)}$, and $d_{dur}^{(P_i,C_i)}$. We normalize the similarity across the

whole candidate set with a min-max normalization with Eq. 8, where $status \in \{invo, err, dur\}$.

$$d_{status}^{(P_i,C_i)} = \frac{d_{status}^{(P_i,C_i)} - \min(d_{status}^{(P,C)})}{\max(d_{status}^{(P,C)}) - \min(d_{status}^{(P,C)})} \tag{8}$$

The intensity of dependency between P_i and C_i is the average similarity of all three similarities between their status series.

$$I^{(P_i,C_i)} = \frac{1}{3} \sum_{status \in S} d_{status}^{(P_i,C_i)}, S = \{invo, err, dur\} \tag{9}$$

Finally, we can build the dependency graph with intensity from the candidate set and the corresponding intensity values.

3.5 Experiments

In this part, we evaluate AID on both simulated dataset and industrial dataset from Huawei Cloud system. For the simulated dataset, we deploy train-ticket [38], an open-source microservice benchmark, for data collection. Apart from the simulated dataset, we also collected a 7-day-long trace dataset with 192 microservices in April 2021 from a region of Huawei Cloud to evaluate AID. Table 2 displays the detailed information about these two datasets.

Since there is no existing work that measures the intensity of service dependency, we employ Pearson correlation coefficient, Spearman correlation coefficient, and Kendall Rank correlation coefficient as the baseline. Particularly, we calculate correlation on the status series of a candidate dependency pair (P, C). For the baselines, we directly use the implementation from Python package `scipy`.[2] We map the correlation to [0, 1] with the function $f(x) = (x + 1)/2$. The intensities of dependencies are then produced in the same way as Eq. 9.

We employ Cross Entropy (CE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), as calculated in Eq. 10 to evaluate the effectiveness of AID in predicting the intensity of dependency. Specifically, cross entropy calculates the difference between the probability distributions of the label and the prediction. Mean absolute error and root mean squared error measure the absolute and squared error. Lower CE, MAE, and RMSE values indicate a better prediction.

[2] https://www.scipy.org/.

Table 2 Dataset statistics

Dataset	Train-ticket	Huawei cloud
# Microservices	25	192
# Spans	17,471,024	About 1.0e10
# Strong	18	67
# Weak	1	8

Table 3 Performance comparison of different methods on two datasets

Dataset	Method	CE	MAE	RMSE
Train-Ticket	Pearson	0.6872	**0.3305**	0.4388
	Spearman	0.7512	0.3735	0.4697
	Kendall	0.6464	0.3749	0.4577
	AID	**0.4562**	0.3435	**0.3859**
Huawei Cloud	Pearson	0.6076	0.4524	0.4563
	Spearman	0.6030	0.4501	0.4537
	Kendall	0.6258	0.4636	0.4656
	AID	**0.3270**	**0.1751**	**0.3044**

$$
\begin{aligned}
CE &= \frac{1}{N}\sum_{i=1}^{N} -[y_i \cdot \log(p_i) + (1-y_i)\cdot \log(1-p_i)] \\
MAE &= \frac{\sum_{i=1}^{N} |y_i - p_i|}{n} \\
RMSE &= \sqrt{\frac{\sum_{i=1}^{N}(y_i - p_i)^2}{N}}
\end{aligned}
\tag{10}
$$

The overall performance is shown in Table 3, where we mark the smallest loss for each loss metric and dataset. AID achieves the best performance on Huawei Cloud dataset and reduces the loss by 45.8%, 61.1%, and 33.2% in terms of cross entropy, mean absolute error, and root mean squared error respectively. On the simulated dataset, AID achieves the best performance in terms of cross entropy and root mean squared error. The improvement of AID on the simulated dataset is smaller than that on the industrial dataset. This is because the benchmark for simulation did incorporate very few fault tolerance mechanisms, making most of the dependencies strong. Moreover, since the service invocations of the Train-Ticket benchmark are very fast, the statuses of Train-Ticket's services are relatively similar, making simple baselines work as well as our approach.

The most time-consuming operations are the candidate selection and service status generation steps because we have to iterate over all the spans in the cloud system. Theoretically, the time complexities of the candidate selection and service status generation steps are $O(S)$, where S is the number of spans to process in the cloud

system. For the intensity prediction step, the time complexity is $O(kN^2)$, where $N = \frac{T}{\tau}$ is the number of bins and k is proportional to the warping window w. In practice, the intensity prediction step takes 155 seconds on average to process two status series both with 1440 bins on a laptop. Since the similarity calculations of different (P, C) pairs are independent, we could easily parallelize the intensity prediction step to further improve the time efficiency.

4 Incident Aggregation for Root Cause Analysis

4.1 Background

In general, cloud systems employ a hierarchical topology, i.e., the stack of *application layer*, *platform layer*, and *infrastructure layer*. Each service embodies the integration of code and data required to execute a complete and discrete functionality. Different services communicate with each other through virtual networks using protocols such as Hypertext Transfer Protocol (HTTP) and Remote Procedure Call (RPC). Such communications among services constitute the complex topology of the large-scale cloud system.

In large-scale cloud systems, failures are inevitable, which may lead to performance degradation or service unavailability. When a failure happens, system monitors will render a large number of incidents to capture different failure symptoms, which can help engineers quickly obtain a big picture of the failure and pinpoint the root cause. For example, "Special instance cannot be migrated" is a critical network failure in the VPC (Virtual Private Cloud) service and the incident "Tunnel bearing network pack loss" is a signal for this network failure, which is caused by the breakdown of a physical network card on the tunnel path.

With the complex topology, a failure occurring to one service tends to have a cascading effect across the entire system. Representative service failures include slow response, request timeout, service unavailability, etc., which could be caused by capacity issues, configuration errors, software bugs, hardware faults, etc. To quickly understand failure symptoms, a large number of monitors are configured to monitor the states of different services in a cloud system [6]. A monitor will render an incident when certain predefined conditions (e.g., "CPU utilization rate exceeds 80%") are met. Typical configurations of monitors include setting thresholds for specific metrics (e.g., RPC latency, error counter), checking service/device availability or status, etc.

Due to the large scale and complexity of cloud systems, the number of incidents is overwhelming in existing incident management systems [5, 6]. These incidents are triggered by the same root cause and describe the failure from different aspects. Thus, they can be aggregated to help engineers understand and diagnose the failure. When a service failure occurs, aggregating related incidents can greatly reduce the number of incidents that need to be investigated and accelerate the process of root cause analysis. An example is presented in Table 4, where items in the first five rows

Table 4 Examples of incident aggregation. Incidents in group 1 are related to a network failure; incidents in group 2 are caused by a hardware problem (disk error)

No.		Incident Title	Time	Pod	Severity
1	Incident group 1	Abnormal running state of virtual machine	2020/10/09 19:40	pod01	Low
2		Virtual network interface receive lost ratio over 20%	2020/10/09 19:40	pod02	High
3		Traffic burst seen in Nginx node	2020/10/09 19:40	pod02	Low
4		Traffic burst seen in LVS (Linux Virtual Server) node	2020/10/09 19:41	pod09	Medium
5		OSPF (Open Shortest Path First) protocol state change	2020/10/09 19:41	pod04	Medium
6	Incident group 2	Excessive I/O delay of storage disk	2020/10/12 14:34	pod09	Medium
7		Component failure	2020/10/12 14:34	pod05	High
8		Hard disk failure	2020/10/12 14:34	pod09	Medium
9		Database account login error	2020/10/12 14:34	pod18	Medium
10		Monitor detected customer impacting incident for Storage in [AZ1]	2020/10/12 14:35	pod10	Medium

and the bottom five rows belong to two groups of aggregated incidents, respectively. Particularly, the first group shows a virtual network failure. Note that only No.3 and No.4 incidents have similar words, while the others do not. Meanwhile, the second group describes a hardware failure, and more specifically, a storage disk error. Engineers can benefit from such incident aggregation as the problem scope is narrowed down to each incident cluster. Without automated incident aggregation, engineers may need to go through each incident to discover the existence of such a problem and then collect all related incidents to understand the problem.

To identify correlated incidents, one straightforward way is to measure the text similarity between two incident reports [8]. For example, incidents that share a similar title are likely to be related. Besides textual similarity, system topology (e.g., service dependency, network IP routing) is also an important feature to resort to for incident aggregation. Due to the dependencies among online services, failures often impose a cascading effect on other inter-dependent services. Service dependency graph can help track related incidents caused by such an effect. However, more often than not, the impact of a failure does not manifest itself completely over the system topology. This issue is ubiquitous in production systems, which has not yet been properly addressed in existing work. Moreover, the patterns of incidents are collectively influenced by different factors such as their topological and temporal locality. Existing investigations [8] combine them by a simple weighted summation, which may not be able to reveal the latent correlations among incidents.

4.2 Root Cause Analysis of System Incident

Although related incidents are indeed generated around the same time, many other cloud components are also constantly rendering incidents. These incidents are mostly trivial issues and therefore become background noise. Incident aggregation based on temporal similarity would suffer from a high rate of false positives. In production environments, some simple incidents are constantly being reported, e.g., “High CPU utilization rate”. These incidents will appear in many transactions (a collection of

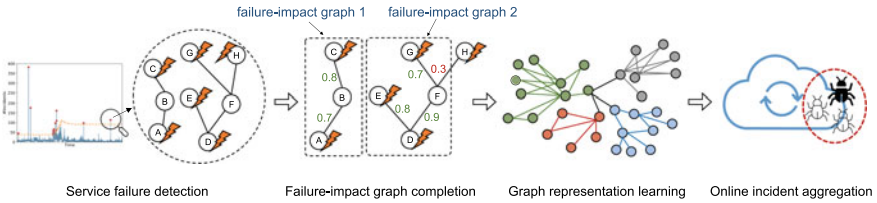


Fig. 7 The overall framework of GRLIA

items that appear together). Text (e.g., incident title and summary) similarity is an important metric for incident correlation, which has been widely used in existing work [8]. However, in reality, related incidents, including the critical ones, do not necessarily have similar titles. Unrelated incidents might be put into the same frequent item set due to sharing such incidents. Particularly, these simple incidents cannot be trivially removed as they provide necessary information about a system and a burst of such incidents could also indicate serious problems. Failing to correlate such critical incidents greatly hinders root cause diagnosis. To accurately correlate incidents, we need to estimate the impact graph of service failures. Incidents alone are insufficient to completely reflect the impact of failures on the entire system. Therefore, we need to utilize more fine-grained information on the failures.

In this part, we introduce GRLIA [7] (standing for Graph Representation Learning based Incident Aggregation), which is an incident aggregation framework to assist engineers in failure understanding and diagnosis. The main motivation is to capture the co-occurrences among incidents by learning from historical failures. In online scenarios, such correlations can be leveraged to distinguish correlated incidents that are generated in a streaming manner. The overall framework of GRLIA is illustrated in Fig. 7, which consists of four phases, i.e., *service failure detection*, *failure-impact graph completion*, *graph representation learning*, and *online incident aggregation*.

The first phase tries to identify the occurrence of service failures and retrieves different types of monitoring data including incidents, KPI time series, and service system topology. In the second phase, we try to identify the incidents that are triggered by each individual failure detected above. More often than not, it is hard to precisely identify the impact scope of failures, which hinders the learning of incidents' correlations. Therefore, we utilize the trends observed in KPI curves to auto-complete the failure-impact graph. After obtaining the set of incidents associated with each failure, in the third phase, an embedding vector is learned for different types of incidents by leveraging existing graph representation learning models [15, 39]. Such representation encodes not only the temporal locality of incidents, but also the network topology. Finally, the learned incident representation will be employed for online incident aggregation by considering their cosine similarity and topological distance.

4.2.1 Incident Similarity

In the first phase, the number of incidents per minute is calculated and incident bursts are regarded as the occurrence of service failures. For each failure, the incidents collected from the entire system are not necessarily related to it. This is because: (1) while some services are suffering from the failure, others may continuously report incidents (could be trivial and unrelated issues); and (2) multiple service failures could happen simultaneously. Therefore, we need to identify the set of incidents for each individual failures that are generated due to the cascading effect.

To this end, the concept of community detection is exploited. Community detection algorithms aim to group the vertices of a graph into distinct sets, or communities, such that there exist dense connections within a community and sparse connections between communities. Each community represents a collection of incidents generated due to a common service failure, in which the correlation among incidents can be explored. A comparative review of different community detection algorithms is available in [34]. In this work, we employ the well-known *Louvain* algorithm [3], which is based upon modularity maximization. The modularity of a graph partition measures the density of links inside communities as compared to links between communities. For weighted graphs, the modularity can be calculated as follows [3]:

$$M = \frac{1}{2m} \sum_{i,j} [W_{i,j} - \frac{k_i k_j}{2m}] \delta(c_i, c_j) \tag{11}$$

where W_{ij} is the weight of the link between node i and j, $k_i = \sum_j W_{ij}$ sums the weights of the links associated with node i, c_i is the community to which node i is assigned to, $m = \frac{1}{2} \sum_{ij} W_{ij}$, and the $\delta(u, v) = 1$ if $u = v$ and 0 otherwise.

To better understand the identification of failure-impact graph using community detection, an illustrating example is depicted in Fig. 7 (phase two). In this case, except for node B and F, other nodes all report incidents. By conducting community detection, we obtain two communities: $\{A, B, C\}$ and $\{D, E, F, G\}$, which are regarded as the complete impact graph of their respective failure. The weight between nodes is provided with their link. We can see that intra-community links all have a relatively large weight. Such partition can achieve the best modularity score for this example. Particularly, node H is excluded from the second community due to the small weight of its connection to node F.

To apply community detection, the weight between two nodes should be defined. Inspired by [20], we combine fine-grained signals, i.e., KPIs, with incidents to calculate the similarity between two nodes and use the similarity value as the weight. Specifically, the weight is composed of two parts, i.e., incident similarity and KPI trend similarity.

The incident similarity is to compare the incidents reported by two nodes. Typically, if two nodes encounter similar errors, they will render similar types of incidents. Jaccard index is employed to quantify such similarity, which is defined as the size of the intersection divided by the size of the union of two incident sets:

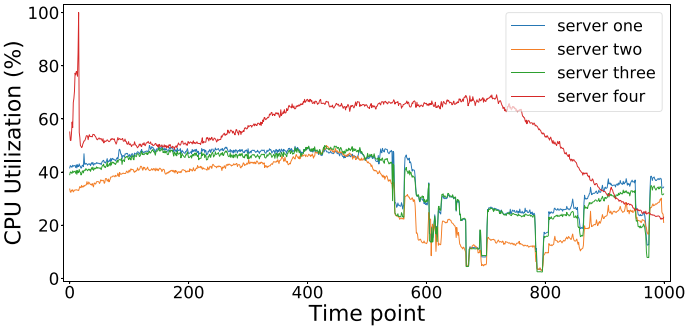


Fig. 8 CPU usage curve of four servers

$$Jaccard(i, j) = \frac{|inc(i) \cap inc(j)|}{|inc(i) \cup inc(j)|} \tag{12}$$

where $inc(i)$ is the incidents reported by node i. In particular, we allow duplicate types of incidents in each set by assigning them a unique number. This is because the distribution of incident types also characterizes failure symptoms.

4.2.2 KPI Trend Similarity

Some services may remain silent when failures happen, which hinders the tracking of related incidents. To bridge this gap, we resort to KPIs, which are more sophisticated monitoring signals. Intuitively, the KPI trend similarity measures the underlying consistency of cloud components' abnormal behaviors, which cannot be captured by incidents alone. An example is shown in Fig. 8, which records the CPU utilization of four servers. Clearly, the curve of the first three servers exhibits a highly similar trend, while such a trend cannot be observed in server four.

The implication is that the first three servers are likely to be suffering from the same issue, and thus should belong to the same community. We adopt dynamic time warping (DTW) [19] to measure the similarity between two temporal sequences with varying speeds. We observe the issue of temporal drift between two time series. This is common as different cloud components may not be affected by a failure simultaneously during its propagation. Therefore, DTW fits our scenario.

The final problem is which KPIs should be utilized for similarity evaluation. Normal KPIs which record the system's normal status should be excluded as they provide trivial and noisy information. Therefore, anomalies in each KPI is detected by a tool called Extreme Value Theory (EVT) [30], which is a popular statistical tool to identify data points with extreme deviations from the median of a probability distribution. Only abnormal KPIs shared by two connected cloud components will

be compared. Particularly, when there exists more than one type of abnormal KPIs, we use the average similarity score calculated as follows:

$$DTW(i, j) = \frac{1}{K} \sum_{k=1}^{K} dtw(t_k^i, t_k^j) \tag{13}$$

where K is the number of KPIs to compare for node i and j, t_k^i is the k^{th} KPI of node i, and $dtw(u, v)$ measures the DTW similarity between two KPI time series u and v, which is normalized for path length. The weight W_{ij} between node i and j is computed by taking the weighted summation of two similarities as follows:

$$W_{ij} = \alpha \times Jaccard(i, j) + (1 - \alpha) \times DTW(i, j) \tag{14}$$

where the balance weight α is a hyper-parameter. In our experiments, if two nodes both report incidents, we set it as 0.5; otherwise, it is set to be 0, i.e., only the KPI trend similarity is considered.

Finally, for each discovered community, the incidents inside it form a complete impact graph of the service failure. Note that in online scenarios, we cannot directly adopt the techniques introduced in this phase for incident aggregation. This is because they involve a comparison between different KPIs, which are not complete until the failures fully manifest themselves. Thus, the comparison is often delayed and inefficient. Moreover, it could be error-prone without fully considering the historical cases.

4.2.3 Cascading Effect of System Incident

In this part, we model the set of incidents triggered by a failure as the impact graph of the failure (or *failure-impact graph*), as illustrated by Fig. 9. Specifically, service A encounters a failure and the impact propagates to other services along with the system topology. The circled area indicates the impact graph of the failure, where irrelevant incidents in service D and G (in a different color) are excluded. In general, the system topology can have many different forms such as the dependencies of services [23], the configured IP routing of cloud network [26], etc. Intuitively, it might seem that the impact graph can be easily constructed by tracing incidents along the system topology. However, our industrial practices reveal that they are usually incomplete. An example is given in Fig. 10, where service B occasionally fails to report any incident. Previous work may perceive it as two separate failures, which is undesirable.

We have identified the following main reason for the missing incidents: System monitors that report incidents are configured with rules predefined by engineers. Due to the diversity of cloud services and conditions, the impact of a failure may not meet the rules of some monitors. For example, if a server generates incidents when its CPU usage exceeds 80%, then any value below the threshold will be unqualified.

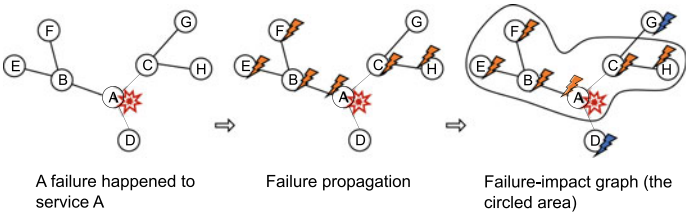


Fig. 9 An illustration of cloud failures' cascading effect. The irregular circle in the third subfigure shows the failure-impact graph

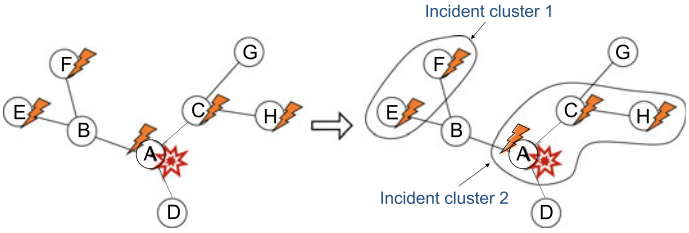


Fig. 10 An example of incomplete failure-impact graph

As a consequence, the monitors will not report any incident and thus the tracking of failure's impact is blocked.

To ensure the continuity of online services, cloud systems are designed to include a certain fault tolerance capability. In this case, some abnormal conditions can be borne by service systems and thus no incidents will be reported. Therefore, the impact of a failure may not manifest itself completely over the system topology.

Recent studies on incident management [6, 17] have demonstrated the incompleteness and imperfection of monitor design and distribution in cloud systems. Thus, along the service dependency chain, a service in the middle may remain silent, which impedes the tracking of a failure's cascading effect. Therefore, although there are many incidents generated by cloud systems, they are often scattered.

4.2.4 Graph Representation Learning for Incident Aggregation

In cloud systems, resources (e.g., microservices and devices) are naturally structured in a graph form such as service dependency and network IP routing. Thus, graph representation learning [15] can be an ideal solution to deal with the above issues. Graph representation learning is an important and ubiquitous task with applications ranging from drug design to friendship recommendation in social networks. It aims to find a representation for graph structure that preserves the semantics of the graph.

A typical graph representation learning algorithm learns an embedding vector for all nodes of a graph. For example, Chen et al. [8] employed *node2vec* [14] to learn a feature representation for cloud components. Different from them, we

propose to learn a representation for each unique type of incident, which could appear in multiple places of the graph. In our framework, we employ DeepWalk [29] because of its simplicity and superior performance. DeepWalk belongs to the class of shallow embedding approaches that learn the node embeddings based on random walk statistics. The basic idea is to learn an embedding ϑ_i for node v_i in graph $\mathcal{G}$ such that:

$$EMB(\vartheta_i, \vartheta_j) \triangleq \frac{e^{\vartheta_i \cdot \vartheta_j}}{\sum_{v_k \in \mathcal{V}} e^{\vartheta_i \cdot \vartheta_k}} \approx p_{\mathcal{G},T}(v_j|v_i), \tag{15}$$

where $\mathcal{V}$ is the set of nodes in the graph and $p_{\mathcal{G},T}(v_j|v_i)$ is the probability of visiting v_j within T hops of distance starting at v_i. The loss function to maximize such probability is:

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} -log(EMB(\vartheta_i, \vartheta_j)), \tag{16}$$

where $\mathcal{D}$ is the training data generated by sampling random walks starting from each node.

For each failure-impact graph, incident sequences are generated through random walk starting from every node inside the graph. In reality, each node usually generates more than one incident when failures happen. Our tailored random walk strategy therefore contains two hierarchical steps. In the first step, a node is chosen by performing random walks on the node level; in the second step, an incident will be randomly selected from those reported by the chosen node. If a node contains duplicate types of incidents, these types will be kept because the frequency is an important feature of incidents (it impacts the probability of being selected).

Following the original setting of [14], we set the walk length as 40, i.e., each incident sequence will contain 40 samples. Finally, the incident sequences will be fed into a Word2Vec model [25] for embedding vector generation. The Word2Vec model has two important hyper-parameters: the window size and the dimension of an embedding vector. We set the window size as ten by following [14] and set the dimension as 128. In particular, by considering the topological distance between incidents, we can alleviate the problem of background noise. This is because as the distance increases, the impact of noisy incidents gradually weakens.

For time series data, anomalies often manifest themselves as having a large magnitude of upward/downward changes. We utilize EVT to predict unusual events by finding the law of extreme values which usually reside at the tail of a distribution. Moreover, it requires no hand-set thresholds and makes no assumptions on data distribution. We follow [30] to detect bursts in the time series of the number of incidents per minute. The bursts are regarded as the occurrence of service failures. This algorithm can automatically learn the normality of the data in a dynamic environment and adapt the detection method accordingly.

With the learned incident representation from the last phase, we can conduct incident aggregation in a production environment, where the incidents come in a

streaming manner. Each group of aggregated incidents represents a specific type of cloud issue, such as hardware issue, network traffic issue, network interface down, etc. The EVT-based method also plays a role in this phase by continuously monitoring the number of incidents per minute. If a failure is alerted, online incident aggregation will be triggered.

When two incidents, say i and j, appear consecutively, GRLIA measures their similarity. If the similarity score is greater than a predefined threshold, they will be grouped together immediately. In particular, the similarity score consists of two parts, i.e., *historical closeness* (HC) and *topological rescaling* (TR), which are defined as follows:

$$\begin{aligned} HC(i,j) &= \frac{\vartheta_i \cdot \vartheta_j}{\|\vartheta_i\| \times \|\vartheta_j\|} \\ TR(i,j) &= \frac{1}{max(1, d(i,j) - \mathcal{T})} \end{aligned} \tag{17}$$

where ϑ_i and ϑ_j are the embedding vectors of incident i and j, respectively; $d(i,j)$ is the topological distance between i and j, which is the number of hops along their shortest path in the failure-impact graph; and $\mathcal{T}$ is the threshold for considering the penalty of long distance. That is, the topological rescaling becomes effective (i.e., <1) only if their distance is larger than $\mathcal{T}$.

In our experiments, $\mathcal{T}$ is set as 4. A very large $\mathcal{T}$ would learn incorrect correlations; while a very small $\mathcal{T}$ would miss important correlations. Our experiments show similar performance when $\mathcal{T}$ is in [3, 21]. Cosine similarity is adopted for calculating the historical closeness, which is related to the co-occurrences of two incidents in the past. Finally, the similarity between i and j can be obtained by taking the product of $TR(i,j)$ and $HC(i,j)$:

$$\begin{aligned} sim(i,j) &= TR(i,j) \times HC(i,j) \\ &= \frac{1}{max(1, d(i,j) - \mathcal{T})} \times \frac{\vartheta_i \cdot \vartheta_j}{\|\vartheta_i\| \times \|\vartheta_j\|} \end{aligned} \tag{18}$$

We set an aggregation threshold λ for $sim(i,j)$ to consider whether or not two incidents are correlated:

$$cor(i,j) = \begin{cases} 1, & if\ sim(i,j) \geq \lambda; \\ 0, & otherwise. \end{cases} \tag{19}$$

In our experiments, λ is empirically set as 0.7. In particular, the distance of an incident to a group of incidents is defined as the largest value obtained through element-wise comparison.

Table 5 Dataset statistics

Dataset	Training period	Evaluation month	#Incidents	#Failures
Dataset1	2020 May - 2020 July	2020 Aug.	~18k/~8k	105/46
Dataset2	2020 May - 2020 Aug.	2020 Sept.	~26k/~10k	151/52
Dataset3	2020 May - 2020 Sept.	2020 Oct.	~36k/~8k	203/38

4.3 Experimental Results on Root Cause Analysis

Incident aggregation is a typical problem across different cloud systems. In this part, we collect real-world incidents from a large-scale Huawei Cloud system to evaluate the proposed framework.

In particular, the service system comprises a large and complex topological structure. In the layer of infrastructure, platform, and software, it has multiple instances of virtual machines, containers, and applications, respectively. In each layer, their dependencies form a topology graph. The cross-layer topology is mainly constructed by their placement relationships, i.e., the mappings between applications, containers, and virtual machines. Like other cloud enterprises, the resources of Huawei Cloud are hosted in multiple regions and endpoints worldwide. Each region is composed of several availability zones (isolated locations within regions from which public online services originate and operate) for service reliability assurance. The incident management of Networking Service is also conducted in such a multi-region way with each region having relatively isolated issues. In our experiments, we collect incidents generated between May 2020 and November 2020, during which the Networking Service reported a large number of incidents from the largest ten availability zones, which are composed of thousands of nodes on average.

The number of distinct incident types is more than 3000. Particularly, we conduct three groups of experiments using incidents reported in the first four months, the first five months, and all months, respectively. In all periods, incident aggregation is applied to the failures happened in the last month based on the incident representations learned from previous months. Table 5 summarizes the dataset. For the column *#Incidents* (resp. *#Failures*), the first figure calculates the incidents (resp. failures) generated during the training period, while the second figure shows that of the evaluation month. Particularly, some failures are of small scale and can be quickly mitigated; while some are cross-region and become an expensive drain on company's revenue. We can see each failure is associated with roughly 200 incidents, demonstrating a strong need for incident aggregation. Although we conducted evaluation on a single online service system, we believe the framework can be easily applied to other cloud systems and bring them benefits.

To evaluate the effectiveness of GRLIA, experienced domain engineers manually labeled the related incidents. Thanks to the well-designed incident management system with user-friendly interfaces, the engineers can quickly perform the labeling. Note that the manual labels are only required for evaluating the effectiveness of our

Table 6 Experimental results of service failure detection using thresholding and GRLIA

Datasets	Metric	Thresholding	GRLIA
Dataset1	Precision	0.711	0.917
	Recall	0.913	0.957
	F1 Score	0.799	0.937
Dataset2	Precision	0.831	0.944
	Recall	0.942	0.981
	F1 Score	0.883	0.962
Dataset3	Precision	0.648	0.925
	Recall	0.921	0.974
	F1 Score	0.761	0.949

framework, which is unsupervised. To calculate the KPI trend similarity, we adopt the following five KPIs, which are suggested by the engineers:

CPU utilization	refers to the amount of processing resources used by a computing device.
Round-trip delay	records the amount of time it takes to send a data packet plus the time it takes to receive an acknowledgement of that data packet.
Port in-bound/out-bound traffic rate	refers to the average amount of data coming-in to/going-out of the port.
In-bound package error rate	calculates the error rate of the package that a network interface receives.
Out-bound package lost rate	calculates the lost rate of the package that a network interface sends.

These KPIs are representative ones that characterize the basic states of the Networking Service system. In particular, CPU utilization is monitored for each network device such as switch, router, and server, while the remaining KPIs are monitored for all interfaces of each device. Each KPI is calculated or sampled every minute. We select data with a time span of two hours for time series comparison. Note that the set of KPIs can be tailored for different systems. For example, a database service may also care about the number of failed database connection attempts, the number of SQL queries, etc.

In the experiments, we employ precision, recall, and F1 scores to evaluate the binary classification problem of whether a given incident is the root cause. Specifically, precision measures the percentage of incident bursts that are successfully identified as service failures over all the incident bursts that are predicted as failures: $Precision = \frac{TP}{TP+FP}$. Recall calculates the portion of service failures that are successfully identified by GRLIA over all the actual service failures: $Recall = \frac{TP}{TP+FN}$. Finally, F1 score is the harmonic mean of precision and recall: $F1\ Score = \frac{2 \times Precision \times Recall}{Precision+Recall}$. TP is the number of service failures that are correctly

discovered by GRLIA; FP is the number of trivial incident bursts (i.e., no failure is actually happening) that are wrongly predicted as service failures by GRLIA; and FN is the number of service failures that GRLIA fails to discover.

We compare GRLIA with the simple thresholding based method on three datasets and report precision, recall, and F1 score. Since both approaches require no parameter training, we use them to detect failures for both the training data and the evaluation data. Particularly, the threshold of the baseline method is #incidents/min$>$50, which is recommended by field engineers. Moreover, the ground truth is obtained directly from the historical failure tickets, which are stored in the incident management system.

The results are shown in Table 6, where GRLIA outperforms simple thresholding in all datasets and metrics. In particular, GRLIA achieves F1 scores of more than 0.93 in different datasets, demonstrating its effectiveness in service failure detection. Indeed, we observe that some failures may not always incur a large number of incidents at the beginning. However, if ignored, they could become worse and end up yielding more severe impacts across multiple services. Simple thresholding does not possess the merit of threshold adaptation based on the context, and thus produces many false positives. GRLIA outperforms the simple thresholding method as it is able to automatically set the threshold.

5 Conclusions

This chapter introduces an anomaly detection method designed to tackle the anomalous status in key performance indicators, a failure diagnosis framework with the analysis of the intensity of service dependency, and root cause analysis techniques from an incident aggregation perspective to identify the fault system incidents. CMAnomaly can learn the pairwise cross-feature and cross-time interactions between KPIs with linear time complexity to quickly obtain a big picture of a system's health status for anomaly detection. AID predicts the intensity of dependencies between cloud microservices to construct an accurate dependency graph. In cloud incident management, GRLIA is a practical root cause analysis tool to capture the interaction with other incidents in temporal and topological dimensions.

Besides, we have discussed some practical experience to improve the reliability of large-scale Huawei Cloud systems and reported several significant improvements in anomaly detection, failure understanding and diagnosis, and root cause analysis. We believe our end-to-end pipeline of the integrated intelligent software engineering framework is a meaningful choice to assist engineers in reliable cloud operations.

References

1. Aceto G, Botta A, De Donato W, Pescapè A (2013) Cloud monitoring: a survey. Comput Netw 57(9):2093–2115
2. Alam ABMB, Haque A, Zulkernine M (2018) CREM: a cloud reliability evaluation model. In: IEEE Global Communications Conference, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, 9–13 Dec 2018. IEEE, pp 1–6
3. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theor Experiment 2008(10):P10008
4. Chen J, He X, Lin Q, Xu Y, Zhang H, Hao D, Gao F, Xu Z, Dang Y, Zhang D (2019) An empirical investigation of incident triage for online service systems. In: Sharp H, Whalen M(eds) Proceedings of the 41st international conference on software engineering: software engineering in practice, ICSE (SEIP), Montreal, QC, Canada, 25–31 May 2019. IEEE/ACM, pp 111–120
5. Chen Z, Kang Y, Gao F, Yang L, Sun J, Xu Z, Zhao P, Qiao B, Li L, Zhang X et al (2020) Aiops innovations of incident management for cloud services
6. Chen Z, Kang Y, Li L, Zhang X, Zhang H, Xu H, Zhou Y, Yang L, Sun J, Xu Z, Dang Y, Gao F, Zhao P, Qiao B, Lin Q, Zhang D, Lyu MR (2020) Towards intelligent incident management: why we need it and how we make it. In: ESEC/FSE '20: 28th ACM joint European software engineering conference and symposium on the foundations of software engineering, Virtual Event, USA, 8–13 Nov 2020
7. Chen Z, Liu J, Su Y, Zhang H, Wen X, Ling X, Yang Y, Lyu MR (2021) Graph-based incident aggregation for large-scale online service systems. In: Proceedings of the 36th IEEE/ACM international conference on Automated Software Engineering (ASE), IEEE
8. Chen Y, Yang X, Dong H, He X, Zhang H, Lin Q, Chen J, Zhao P, Kang Y, Gao F et al (2020) Identifying linked incidents in large-scale online service systems. In: Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, pp 304–314
9. Devops. https://aws.amazon.com/devops/what-is-devops/
10. Du M, Li F, Zheng G, Srikumar V (2017) Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 conference on Computer and Communications Security, (CCS). ACM, pp 1285–1298
11. Fonseca R, Porter G, Katz RH, Shenker S, Stoica I (2007) X-trace: a pervasive network tracing framework. In: Balakrishnan H, Druschel P (eds) 4th symposium on Networked Systems Design and Implementation (NSDI 2007), 11–13 Apr 2007, Cambridge, Massachusetts, USA, Proceedings. USENIX
12. Gan Y, Zhang Y, Hu K, Cheng D, He Y, Pancholi M, Delimitrou C (2019) Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In: Proceedings of the twenty-fourth international conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, 13–17 Apr 2019. ACM, pp 19–33
13. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3% in 2020. https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020
14. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp 855–864
15. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. arXiv preprint arXiv:1709.05584
16. He S, Lin Q, Lou JG, Zhang H, Lyu MR, Zhang D (2018) Identifying impactful service system problems via log analysis. In: Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the Foundations of Software Engineering—ESEC/FSE 2018, pp. 60–70

17. Huang P, Guo C, Zhou L, Lorch JR, Dang Y, Chintalapati M, Yao R (2017) Gray failure: the achilles' heel of cloud-scale systems. In: Proceedings of the 16th workshop on hot topics in operating systems, pp 150–155
18. Hundman K, Constantinou V, Laporte C, Colwell I, Söderström T (2018) Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In: Proceedings of the 24th international conference on Knowledge Discovery & Data Mining, (KDD), pp 387–395
19. Keogh EJ (2002) Exact indexing of dynamic time warping. In: Proceedings of 28th international conference on very large data bases, VLDB 2002, Hong Kong, 20–23 Aug 2002. Morgan Kaufmann, pp 406–417. https://doi.org/10.1016/B978-155860869-6/50043-3
20. Liu P, Chen Y, Nie X, Zhu J, Zhang S, Sui K, Zhang M, Pei D (2019) Fluxrank: a widely-deployable framework to automatically localizing root cause machines for software service failure mitigation. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). IEEE, pp 35–46
21. Lloyd's Estimates the Impact of a U.S. Cloud Outage at $19 Billion. http://www.eweek.com/cloud/lloyd-s-estimates-the-impact-of-a-u.s.-cloud-outage-at-19-billion
22. Lyu MR et al (1996) Handbook of software reliability engineering, vol 222. IEEE Computer Society Press, CA
23. Ma SP, Fan CY, Chuang Y, Lee WT, Lee SJ, Hsueh NL (2018) Using service dependency graph to analyze and test microservices. In: 2018 IEEE 42nd annual Computer Software and Applications Conference (COMPSAC), vol 2. IEEE, pp 81–86
24. Malhotra P, Ramakrishnan A, Anand G, Vig L, Agarwal P, Shroff G. LSTM-based encoder-decoder for multi-sensor anomaly detection **abs/1607.00148**
25. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119
26. Natarajan A, Ning P, Liu Y, Jajodia S, Hutchinson SE (2012) NSDMiner: automated discovery of network service dependencies, IEEE
27. OpenTracing: opentracing spring cloud (2021). https://github.com/opentracing-contrib/java-spring-cloud
28. Park D, Hoshi Y, Kemp CC. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. **abs/1711.00614**
29. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 701–710
30. Siffer A, Fouque PA, Termier A, Largouet C (2017) Anomaly detection in streams with extreme value theory. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1067–1075
31. Su Y, Zhao Y, Niu C, Liu R, Sun W, Pei D (2019) Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD international conference on Knowledge Discovery & Data Mining, KDD
32. Wang Y, Li G, Wang Z, Kang Y, Zhou Y, Zhang H, Gao F, Sun J, Yang L, Lee P, Xu Z, Zhao P, Qiao B, Li L, Zhang X, Lin Q (2021) Fast outage analysis of large-scale production clouds with service correlation mining
33. Wang P, Xu J, Ma M, Lin W, Pan D, Wang Y, Chen P (2018) Cloudranger: root cause identification for cloud native systems. In: El-Araby E, Panda DK, Gesing S, Apon AW, Kindratenko VV, Cafaro M, Cuzzocrea A (eds) 18th IEEE/ACM international symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, 1–4 May 2018. IEEE Computer Society, pp 492–502
34. Yang Z, Algesheimer R, Tessone CJ (2016) A comparative analysis of community detection algorithms on artificial networks. Sci Rep 6:30750
35. Yang T, Shen J, Su Y, Ling X, Yang Y, Lyu MR (2021) AID: efficient prediction of aggregated intensity of dependency in large-scale cloud systems. In: Proceedings of the 36th IEEE/ACM international conference on Automated Software Engineering (ASE), IEEE

36. Zhang C, Song D, Chen Y, Feng X, Lumezanu C, Cheng W, Ni J, Zong B, Chen H, Chawla NV (2019) A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In: The thirty-third AAAI conference on Artificial Intelligence, AAAI
37. Zhao H, Wang Y, Duan J, Huang C, Cao D, Tong Y, Xu B, Bai J, Tong J, Zhang Q (2020) Multivariate time-series anomaly detection via graph attention network. http://arxiv.org/abs/2009.02040
38. Zhou X, Peng X, Xie T, Sun J, Ji C, Li W, Ding D (2021) Fault analysis and debugging of microservice systems: industrial survey, benchmark system, and empirical study. IEEE Trans Softw Eng 47(2):243–260
39. Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2018) Graph neural networks: a review of methods and applications. arXiv preprint arXiv:1812.08434 (2018)

Data Analytics: Predicting Software Bugs in Industrial Products

Robert Hanmer and Veena Mendiratta

Abstract Achieving high software reliability in products is a costly process. Faults found late in the development cycle are the costliest to fix. Defect prediction models are developed prior to and during various stages of testing to predict the faults remaining or to predict which software modules are more prone to failures. Increasingly machine learning models are used for this purpose, using various code metrics and defect data. In this paper we will review the need for targeted testing and various machine learning approaches for defect prediction. Additionally, we will present a new methodology for improving software reliability during product development based on the results from the analytics models, which we demonstrate with a small case study.

1 Introduction

It is costly to achieve high software reliability in large-scale products. The most cost-effective means is to find and remove defects early. More effort and expense is involved to fix the defects found late in the cycle than those found early in development.

The typical methodology for achieving highly reliable software involves using some type of model to predict the faults remaining during various stages of testing (software reliability growth models), or predict which software modules are more prone to failures with an analytics model (using metrics data and machine learning algorithms). Both of these methods allow test plans to be made based on the model predictions to focus testing effort on those areas most needing it.

R. Hanmer (✉)
Nokia, Naperville, IL, USA
e-mail: robert.hanmer@nokia.com

V. Mendiratta
Northwestern University, Evanston, IL, USA
e-mail: veena.mendiratta@northwestern.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_3

1.1 The Problem

Much of the software development in industry involves adding code—new features or bug fixes—to existing systems for a new release, and maintaining the reliability of the software across releases. This results in a continuous cycle of analysis, prediction and modifying the code to improve its reliability. We propose here a new methodology for improving software reliability based on the results from analytics models which we demonstrate with a small experimental case study. The context in which we examined the analytics models is that of large telecommunication systems. We believe the findings will be relevant to any large-scale industrial system.

1.2 Our Contributions

We will present a focused review of machine learning applications in software reliability, in particular, in the context of code metrics data and defect data from lab tests as well from field operations. We discuss complexity metrics that are used to assess the relative risk within different programming modules. Their use lets us predict the most dynamic modules which are likely to be the most buggy. We describe the complexity metrics used in our case study and the enhancements we tried.

We will show our methodology, the model results from a production software system including several experiments with the data and the insights gained, the potential gains from using this methodology, and the future work areas to further explore the methods.

The goal of this work is to reduce the risk of new faults while testing a new release of the system in the most efficient manner possible.

2 Review of ML Applications

In this section our goal is to provide an overview of research in ML applications of software reliability engineering with an emphasis on software defect prediction and a focus on more on recent work (as there are many reviews of earlier work).

In an industrial setting, the type of software reliability model developed depends on the phase of the software development process for which the model is used. In the context of machine learning models, Shafiq et al. [25] present a useful taxonomy of the application of ML approaches for software engineering during different phases of the software development life cycle from a research perspective. The phases listed are: requirements engineering, architecture and design, implementation, quality assurance and analytics, and maintenance. Our review of the literature is focused on aspects of the quality assurance phase, along with a few examples from the maintenance phase.

Some of the early work on applying ML methods in reliability modeling focused primarily on the pre-testing phase where the model outputs were used to guide the software testing effort. Gokhale et al. [6] and Ma et al. [17] used software metrics to predict fault-proneness by software module. While the work of Khoshgoftaar et al. [11–13] applied alternative estimation techniques for nonlinear regression with software metrics as explanatory variables in order to predict the number of faults in a program module.

There is a considerable body of more recent work on the application of ML techniques for software defect prediction, and there are several survey papers that review this work. Pandey et al. [21] present an extensive review of various studies applying machine learning for software fault prediction citing over 200 research articles in the period 1990–2019, and also compared the performance between machine learning and statistical techniques. Their study found that the prediction ability of machine learning techniques for classifying class/module as fault/non-fault prone is better than classical statistical models. Li et al. [16] present a systematic literature review of unsupervised machine learning techniques for software defect prediction published between January 2000 and March 2018. Their meta-analysis shows that unsupervised models are comparable with supervised models for both within-project and cross-project prediction. Xu et al. [29] present a comparative study of clustering-based unsupervised defect prediction models using an open-source dataset including 27 project versions with 3 types of features. They found that: different clustering-based models have significant performance differences and the performance of models in the instance-violation-score-based clustering family is superior to that of models in hierarchy-based, density-based, grid-based, sequence-based, and hybrid-based clustering families. Further, the impacts of feature types on the performance of the models are related to the indicators used, and the clustering-based unsupervised models do not always achieve better performance on defect data with the combination of the 3 types of features.

Recent work has also investigated change metrics in conjunction with code metrics to improve the performance of fault prediction models [2, 23]. Other recent work researches defect prediction based on code features, where the goal is to determine if a piece of code contains bugs. For instance, Wang et al. [28] defined an approach that learns semantic features for defect prediction. The approach takes tokens from the source code of the training and test datasets as input, and generates semantic features, which are then used to build and evaluate the models for predicting defects. Shippey et al. [26] proposed a defects prediction approach based on learning features from N-grams extracted from Abstract Syntax Trees.

Maddipati and Srinivas [18] apply principal component analysis (PCA) to identify the most relevant attributes in identifying defective prone modules. Further, to address the issue of class imbalanced learning they propose a cost sensitive adaptive fuzzy inference system for constructing the classifier to predict software defects. To address the issue of the imbalanced data distribution of training datasets for defect prediction, Ding and Xing [4], present a software defect prediction method using pruned histogram-based (which deals with the convergence issue) isolation forest.

With respect to software testing, Durelli et al. [5] review the state-of-the-art of how ML has been applied to automate and streamline software testing.

In the operational phase ML applications typically include: anomaly detection—find undesired patterns in the data; root cause analysis—investigate what caused the anomalous behavior; failure prediction—monitor metrics to predict failures based on knowledge of abnormal patterns and their causes; and preventive maintenance—prevent failures before they occur based on predictions. The specific techniques used depend on the type of data that is available, common types being: system logs which provide a rich source of data for anomaly detection and prediction and root cause analysis where supervised, unsupervised, and reinforcement learning methods are used; and increasingly, data from automated smart sensors is used for preventive maintenance, and coupled with log data, can be used for root cause analysis. Candido et al. [1] present a survey on different log analysis techniques using machine learning for anomaly detection and prediction and root cause analysis. An evaluation of several supervised and unsupervised ML methods for anomaly detection using log data is presented by He et al. [8]. Their findings include: supervised anomaly detection methods present higher accuracy when compared to unsupervised methods; the use of sliding windows (instead of a fixed window) can increase the accuracy of the methods; and the methods scale linearly with the log size. In practice, the data available for analysis is often unlabeled or weakly labeled, thereby precluding the use of supervised learning techniques.

An important ML application in the operational phase is preventive maintenance; in addition there are domain specific applications and techniques, in particular for root cause analysis. There is considerable work in the optical networking space using ML techniques for preventive maintenance. Several tutorial papers are available on the topic [19, 22] that introduce automated methods for failure detection and prediction; and localization and identification (root cause analysis). Typically, these methods cover both hardware and software failures and errors. Kim et al. [14] present an approach for anomaly detection and root cause analysis in mobile networks using unsupervised ML techniques and finite state machines respectively.

3 Review of Software Testing

In a project with hundreds or thousands of modules not every one can be tested extensively in each release. Choosing the right modules to focus test is an important part of testing in addition to designing the tests. *Module* here refers to a collection of code that does one thing and that may include more than one method/function/file. A collection of module changes are delivered together as a *release.*

Some of the coding changes in a release will be trivial. Some will be more complicated and involve several modules. The complicated ones might be because the new addition is itself complicated or it might be because it is a change in an overly complex module or across modules. The complexity makes it easier to make a mistake, adding a defect into the code.

A study by Huang [9] of the Hadoop open source package looked at 4218 issue reports spanning almost six years. The issues examined were all valid issues (i.e. where a fix actually corrected the problem). They found that most of the issues (more than 79%) are not closely related to issues in other subsystems. Identifying those modules that, by themselves, are most dynamic can help target testing. They also found that between 26 and 33% of issues had causes similar to other issues within the same subsystem. Looking at modules with similar characteristics in terms of their dynamic nature or complexity metrics can help identify modules that should have targeted testing. They also found that correlated issues required almost twice the effort to fix as uncorrelated issues. This can cause the issue correction to span several releases or to delay the delivery of a release.

3.1 Dynamic Modules

Testing the modules that are directly changed in a release is an obvious starting point when preparing the testing plan. These are the most dynamic modules, where it is most likely that a human has introduced a defect. Some modules in an application are more active than other modules. For simplicity we limit discussion to 3 kinds of activity: new enhancements, fault corrections (fixes) and fixes to those fixes ("fix on fix").

Knowing which modules are more active can guide the testing effort. Systems used to track the introduction of new features into software modules provide insight into where the new enhancements are being made. Those actual locations (files) in the module receiving the change may need to be refined by examining the code repositories. Entries there will reflect both the new enhancements and defect remediation. Systems that track the identified defects the defect tracking systems are another source of information and are needed to identify which code changes were the result of fixing defects or introducing new features.

3.2 The "fix on fix" Problem

Sometimes when changes/fixes are made to software they contain new defects. Fixes are then created to fix these previous fixes. This has been problem for a long time. In 1989 Levendel [15] studying a large telecommunications system reported the following: "A model simulation was conducted, and it was found that one defect is reintroduced for every three defect repairs. In the long run, this means that 50% of the original defects are reintroduced due to imperfect repairs. These defects are introduced later in the process and are subjected to a similarly long time to detection. Also, they may elude detection since a large amount of the test program has already been executed."

Huang et al. [9] found that about 10.5% of the issues arose due to fixes on fixes in Hadoop.

3.3 Complexity Metrics

Complexity metrics have long been used to identify modules that are more likely to contain defects.

In [22] the authors review various software metrics and assess their applicability for software fault prediction. They found that the traditional metrics related to size and complexity were less successful in predicting faults than metrics that are related to object-oriented constructs or those derived from the software construction process. Traditional metrics include lines of code and McCabe's complexity measure. One conclusion from this paper is that metrics should be studied in large industrial contexts to determine the best context-specific metrics.

The context that this chapter reflects is that of large, long-lived telecommunications systems. Previous studies have looked at which metrics are best in this context.

Levendel's [15] primary focus was the reliability of the system, obtained through metrics of the number of defects in the system. The way defects flow through the system, being first introduced by developers then being corrected and then maybe being corrected again (*fix on fix* as discussed above) was studied.

Ohlsson [20] looked at a different telecommunications system. The goal in this study was to predict the number of faults that will be present in a module before the module is written. A key parameter they used was the number of *signals* between modules. They compared measures that can be directly computed based on the design, such as McCabe's Cyclomatic Complexity, as well of measures obtained by examination, such as the number of objects in the functions and subroutines. From these, additional metrics were computed. The study did not draw any conclusions that can be generalized to select which variables to use for a specific modeling exercise, but they found that simple measures findable by examination were as useful as more complicated measures.

Graves [7] looked at software aging and latent defects by taking a snapshot of the system's metrics and then analyzing the incidence of trouble reports over the next two years. They found that the simple metrics such as lines of code were not effective at predicting the presence of defects.

The most effective predictor that Graves identified was a *weighted time damp model* that used the accumulation of all past module changes to predict the potential for faults in the module along with a defect removal rate. Using the number of changes in a module in combination with the module's age was identified as the most generalizeable linear model in their study. Factoring in measurable quantities such as the number of lines of code by combining them arithmetically with other metrics did not improve the predictive ability.

Among Graves other finding is that the number of developers in a file, the "too many cooks" effect, was not a predictor of error rates. Neither was the frequency with which multiple modules are changed simultaneously ("in tandem") a good predictor of the presence of being defects in the modules.

3.3.1 Traditional Metrics

For illustrative purposes several metrics will be highlighted. These will be used in the case study presented in Sect. 4.2.

The first metric is the *McCabe complexity metric* which measures the number of execution paths through a function or method. Each function or method has a complexity of one plus one for each branch statement such as if, else, for, foreach, or while. A complexity count is added for each logical combination ("and" or "or") in the logic within if, for, while or similar conditional statements.

The *median methods implemented per class* is the median of the number of methods (functions) defined for the classes within the module.

The *average statements per method* and *mean lines* are computed simply by looking at the number of lines in each method (or function) within a module. *mean lines* is the overall size of the entire module.

Another of the metrics is *Maximum Block Depth.* This metric looks at the depth of nested blocks of code. While nesting can be used alone in most languages, nested blocks are almost always introduced with execution control statements such as "if", "case" and "while". As the depth grows, the code gets harder to read and understand because with each new nested depth level, more conditions must be evaluated if you want to know when the code will be executed. Average block depth is the weighted average of the block depth of all statements in a file or module.

For blocks at levels zero and one (i.e. only one or no level of nesting) two metrics *Statements and Block Level 0* or *1* are also easily computable.

The *Line Number of Deepest Block* metric is also related to the Maximum Block Depth. It is an indication of how much processing occurs within a module before that deepest block is enter. This can be a sign that the how much preparation is needed before that deepest module is entered.

The *Percent Branch Statements* is the number of statements that cause a break in sequential execution. These statments that break the flow are: *if, else, for, while, goto, break, continue, switch, case, default*, and *return.*

Reducing Complexity. Complexity can be reduce by a number mechanisms. The changes might add to the overall number of simple metrics such as total lines, but will reduce the average complexity of the overall module.

- A method or function can be refactored by breaking a routine into smaller routines. While the total (sum) complexity will be little changed the maximum in any routine will be reduced.
- Specialized refactoring can be done to the code to pull most commonly executed parts out into efficient, less complex parts; this differs from the previous suggestion

in that it looks at code and focuses on the most likely to be encountered parts of the code.
- Since the code is not new (i.e. release N, N > 3) complexity can be reduced by pruning legs of code that are now known to not be executed.

3.3.2 Microservice Metrics

The computing industry is moving towards architectures made up of microservices. Microservices involve many different modules by definition. While the term is *micro*-services, not all the services will be small. But since the services tend to be modules providing a single well-defined functionality, just about any fix to a module could be a fix on fix.

In the microservices world the "traditional" metrics of software complexity are not generally discussed. A new language has taken over to discuss the quantity and rapidity of making changes within an application built of microservices.

One metric that is commonly discussed in the context of microservices is that of *technical debt*. "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt" [3].

The other use of metrics in microservices is used during the conversion of a legacy application to a microservice architecture. These metrics compare the legacy application with a possible microservice implementation. In [24] a new complexity metric for a microservice implementation is created. As a monolith is decomposed into microservices, the complexities of the decomposed functionalities are assessed and combined into this new metric which is compared with the legacy monolith which has a presumed value of zero. Competing microservice architectural implementations can then be compared against each other, with the implementation with the lowest metric value being considered the best.

The errors caused by faults in microservice architectures are mitigated in running systems differently than in non-microservice systems. The nature of microservice architectures provides potentially massive redundancy which can hide individual errors. This might be a reason predicting the faults in individual microservices has not been perceived as necessary; modeling at the application level is more appropriate in this case. To this end, Jagadeesan and Mendiratta [10] present a modeling framework for the reliability of microservice-based applications (with a service mesh), in which applications may operate in a degraded mode if non-critical microservices are considered to have failed. The modeling framework includes a micro-level of detailed operational behavior of small sets of microservices communicating with one another via a service mesh which can be useful for iterative system design. At the macro-level the reliability framework is suitable for typical application deployments comprised of thousands of service meshed microservices.

The usefulness of traditional complexity measures on the design and construction of microservice based systems is a topic for future research.

4 Case Study

Analytics models for predicting software defects are run in the pre-testing phase, and a typical model (run by software module) works as follows:

- a model is trained using code metrics and fault data from an existing release (Release N), and
- code metrics from the new release (Release N+1) are input to the trained model to predict software faults for the new release.

If a classification model is used the modules are classified in terms of fault-proneness. If a regression model is used the model predicts the expected number of faults. Another output of these models is Variable Importance which is a ranking of the code metrics in terms of their contribution to the model fault prediction.

Our approach, shown in Fig. 1, is to work with fault data and code metrics to examine how these metrics could be modified to improve software reliability.

To test the validity of this approach we train a model with Release N data. We then run the model (score the data) for Release N+1 code metrics and obtain the Variable Importance charts. Based on these charts the code metrics in Release N+1 are modified and the model rerun with the modified data and the fault prediction results compared. This type of approach falls in the area of model explainability and research in this area is gaining traction in other contexts.

The work in [27] presents research on counterfactual explanations, a class of explanation that provides a link between what could have happened had the inputs to a model been changed in a particular way. Such explanations can help the ML model developers identify, detect, and fix issues in the system under study and improve the performance. In the Sect. (4.2) we describe how this approach in used in our modeling.

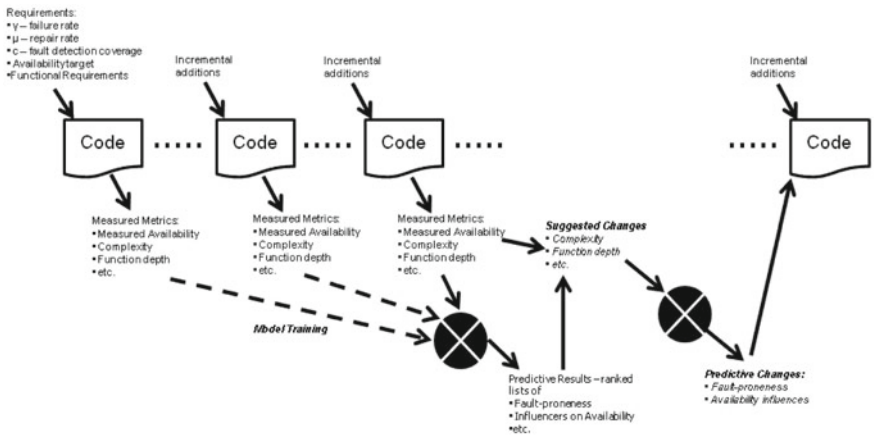


Fig. 1 Proposed approach

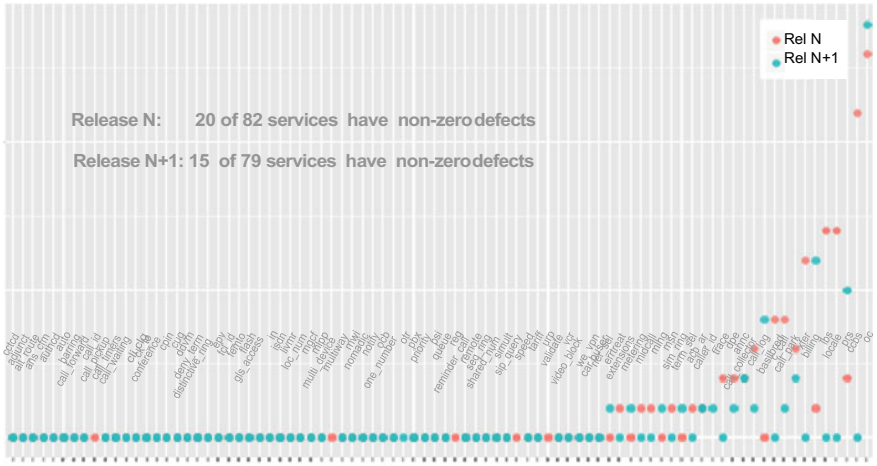


Fig. 2 Defects per service and release

4.1 Data Collection and Processing

Various code metrics were extracted with the use of open source software tools. The metrics were extracted per service, e.g., calling type, etc. where the metrics or a given service are computed over each program file. Services are generally realized through several modules of many files each. The data was pre-processed to aggregate file-level metrics into service-level metrics, i.e., compute min, max, mean, median across file-level metrics to create model features (variables). A snapshot of the defect data is shown in Fig. 2. Note from Fig. 2 that for Release N, 20 of 82 services have non-zero defects while for Release N+1, 15 out of 79 services have non-zero defects, indicating that less than 25% of the services have defects in a given release.

4.2 Models

A regression model was used to predict the number of defects by service where, the target variable is the number of defects, and the predictive variables are created from the code metrics.

The Random Forest algorithm, which uses an ensemble of regression trees to boost predictive power and accuracy, was used for defect prediction. The algorithm constructs a succession of unpruned trees (independently) using a different bootstrap sample of the data. In constructing the tree, each node is split using the best among a subset of randomly chosen variables. The defect prediction averages the prediction of all the trees. Model evaluation is based on a comparison of predicted and observed defects.

In addition to the prediction of the number of defects, the random forest algorithm provide another set of outputs, namely the variable importance which identifies which variables are important for the prediction.

The variable importance measures are calculated in two ways.

1. Percentage increase in Mean Square Error (MSE) is based upon the mean decrease of accuracy in predictions on the OOB (out-of-bag) samples when a given variable is excluded from the model. It is computed by permuting the values of the OOB samples: for each tree, the prediction error (MSE) on the out-of-bag portion of the data is recorded. Then the same is done after permuting each predictor variable. The difference between the two are averaged over all trees, and normalized by the standard deviation of the differences.
2. Increase in node purity is analogous to Gini-based importance, and is calculated based on the reduction in sum of squared errors whenever a variable is chosen to split. It is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees. In the case of regression trees, the node impurity is measured by the training RSS (residual sum of squares).

In both cases, the higher a variable is on the chart—that is, a higher value of the percentage increase in MSE or total increase in node purity—the higher is the importance of the variable in the model.

In Fig. 3 we present the variable importance plots using the two measures defined above for a run of the defect prediction model.

Based on these results, and as described in Sect. 3.3.1, for illustrative purposes, the following 6 variables are modified and the models rerun to determine the impact on the number of predicted defects: *complexity*, *percent branch statements*, *block depth*, *line number of deepest block*, *statements at block level 0*, and *statements at*

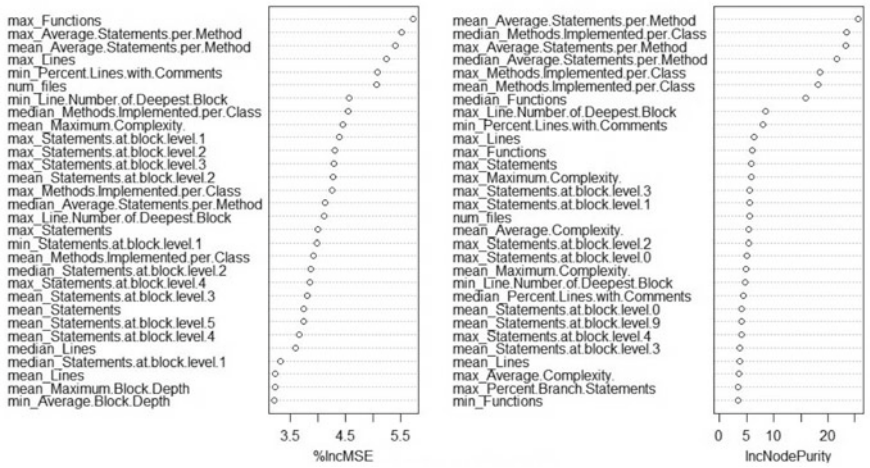


Fig. 3 Variable importance with original metrics

block level 1. The approach used was to modify the above variables (1 variable per model run) in the appropriate direction and observe the impact on model results.

The modified value was determined by: if value > median, value = median; else value = value.

Figure 4 shows the variable importance charts with the complexity variable modified. Note from the figure that no complexity related variables appear in the chart—this is due to the complexity metric being modified, i.e. lowered.

The model predictions (normalized) based on the 6 modified variables are shown in Table 1. For each variable the results (normalized) show the predicted defects based on a regression tree model and the random forest model; and are compared with the observed data. Modifying the first three metrics shows a decrease in the number of predicted defects as compared to the observed defects. Modification of the latter three metrics does not exhibit a similar change which confirms the results

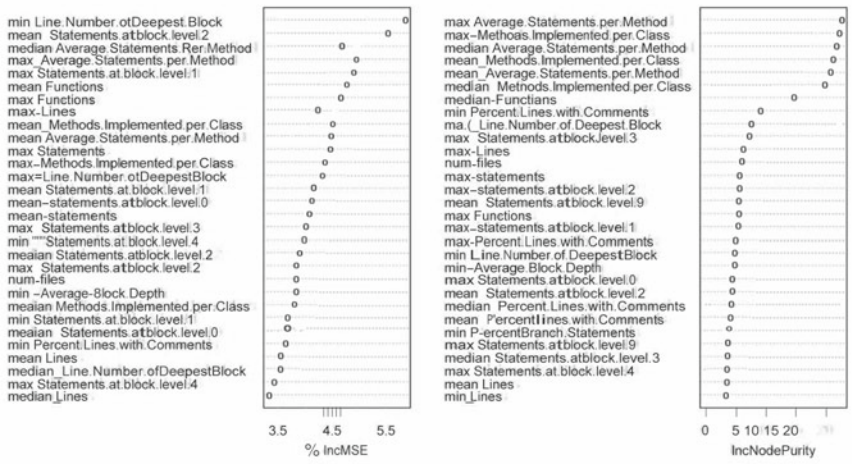


Fig. 4 Variable importance with modified metrics

Table 1 Observed and predicted defects based on modified metrics

Variable	Observed	Regression tree	Random forest
Complexity	100	87	96
Block depth	100	90	92
% Branch statements	100	99	95
Line number deepest block	100	99	99
Statements at block level 0	100	100	100
Statements at block level 1	100	100	100

of Graves et al. [7] that simple metrics (like statements at a level) have less impact on the predictive ability of the models.

The purpose of this case study was to explore the idea of reducing defects by changing the variables that are ranked high in importance in the appropriate direction through programming changes without impacting functionality and performance. Initial results show that it is possible. However, we worked with a small dataset which can impact the quality of the results; to gain confidence in this approach it is necessary to test the method with larger and more varied data sets.

5 Learnings, Thoughts, Musings

Based on our experience with architecting and modeling high reliability software systems we summarize our learnings as applied to data analytics and metrics for modeling in this section.

In a given release many services have no faults, possibly because those services were not changed and hence not tested in that release; this skews the data. It is recommended to only use the data for services that are modified in the release. Use information about the age of individual software units, e.g. maturity of the code, number of releases the service existed, number of times the service was tested, the number of tests performed, etc. The works of [15, 7] both confirm that this should be useful in the prediction of defect-prone modules. Introduce information about which services have been extensively tested for each release. This may require that new metrics of prior testing be recorded and made available.

A number of steps can be taken to validate and further improve the classifiers and build more accurate defect prediction models. These steps include collecting metrics and defect data from additional releases. The longer term studies cited earlier [7, 9, 15] all show the benefits of watching the defect removal evolve over time. The analysis and refinement can continue with larger datasets and additional metrics. Our exploratory case study looked at a small part of a large system. We picked just a few metrics to enhance in our case study. Future work can explore additional metrics and combinations of metrics. An additional source of defect data is failure data from the field.

Our method produces a large set of data and indicators that can help understand future releases and not just narrow guidance as provided by traditional Software Reliability Growth methods. These indicators (variable importance) can be used to guide software refactoring to improve software reliability.

References

1. Cândido JB, Aniche MF, van Deursen A (2019) Contemporary software monitoring: a systematic mapping study. arXiv preprint arXiv:1912.05878

2. Choudhary GR, Kumar S, Kumar K, Mishra A, Catal C (2018) Empirical analysis of change metrics for software fault prediction. Comput Electr Eng 67:15–24
3. Cunningham W (1992) The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4(2):29–30
4. Ding Z, Xing L (2020) Improved software defect prediction using pruned histogram-based isolation forest. Reliab Eng Syst Safety 204:107170
5. Durelli VH, Durelli RS, Borges SS, Endo AT, Eler MM, Dias DR, Guimarães MP (2019) Machine learning applied to software testing: a systematic mapping study. IEEE Trans Reliab 68(3):1189–1212
6. Gokhale SS, Lyu MR (1997) Regression tree modeling for the prediction of software quality. In: Proceedings of the third ISSAT international conference on reliability and quality in design. Citeseer, pp 31–36
7. Graves TL, Karr AF, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661. https://doi.org/10.1109/32.859533
8. He S, Zhu J, He P, Lyu MR (2016) Experience report: System log analysis for anomaly detection. In: 2016 IEEE 27th international symposium on software reliability engineering (ISSRE). IEEE, pp 207–218
9. Huang J, Zhang X, Schwan K (2015) Understanding issue correlations: a case study of the hadoop system. In: Proceedings of the sixth ACM symposium on cloud computing, SoCC '15. Association for Computing Machinery, New York, NY, USA, pp 2–15. https://doi.org/10.1145/2806777.2806937
10. Jagadeesan LJ, Mendiratta VB (2020) When failure is (not) an option: Reliability models for microservices architectures. In: 2020 IEEE international symposium on software reliability engineering workshops (ISSREW). IEEE, pp 19–24
11. Khoshgoftaar T, Bhattacharyya B, Richardson G (1992) Predicting software errors, during development, using nonlinear regression models: a comparative study. IEEE Trans Reliab 41(3):390–395
12. Khoshgoftaar T, Munson J (1990) Predicting software development errors using software complexity metrics. IEEE J Selected Areas Commun 8(2):253–261
13. Khoshgoftaar T, Munson J, Bhattacharya B, Richardson G (1992) Predictive modeling techniques of software quality from software measures. IEEE Trans Softw Eng 18(11):979–987
14. Kim C, Mendiratta VB, Thottan M (2020) Unsupervised anomaly detection and root cause analysis in mobile networks. In: 2020 International conference on communication systems & networks (COMSNETS). IEEE, pp 176–183
15. Levendel Y (1989) Defects and reliability analysis of large software systems: field experience. In: 1989 The nineteenth international symposium on fault-tolerant computing. Digest of Papers. IEEE Computer Society, pp 238–239
16. Li N, Shepperd M, Guo Y (2020) A systematic review of unsupervised learning techniques for software defect prediction. Inform Softw Technol 122:106287. https://doi.org/10.1016/j.infsof.2020.106287. URL https://www.sciencedirect.com/science/article/pii/S0950584920300379
17. Ma Y, Guo L, Cukic B (2007) A statistical framework for the prediction of fault-proneness. In: Advances in machine learning applications in software engineering. IGI Global, pp 237–263
18. Maddipati SS, Srinivas M (2021) Machine learning approach for classification from imbalanced software defect data using PCA & CSANFIS. Mater Today: Proc
19. Musumeci F, Rottondi C, Corani G, Shahkarami S, Cugini F, Tornatore M (2019) A tutorial on machine learning for failure management in optical networks. J Lightwave Technol 37(16):4125–4139
20. Ohlsson N, Alberg H (1996) Predicting fault-prone software modules in telephone switches. IEEE Trans Softw Eng 22(12): 886–894. https://doi.org/10.1109/32.553637
21. Pandey SK, Mishra RB, Tripathi AK (2021) Machine learning based methods for software fault prediction: a survey. Expert Syst Appl 172:114595. https://doi.org/10.1016/j.eswa.2021.114595.URL https://www.sciencedirect.com/science/article/pii/S0957417421000361
22. Rafique D, Velasco L (2018) Machine learning for network automation: overview, architecture, and applications [invited tutorial]. J Optical Commun Netw 10(10):D126–D143

23. Rhmann W, Pandey B, Ansari G, Pandey D (2020) Software fault prediction based on change metrics using hybrid algorithms: an empirical study. J King Saud Univ-Comput Inform Sci 32(4):419–424
24. Santos N, Rito Silva A (2020) A complexity metric for microservices architecture migration. In: 2020 IEEE international conference on software architecture (ICSA), pp 169–178. https://doi.org/10.1109/ICSA47634.2020.00024
25. Shafiq S, Mashkoor A, Mayr-Dorn C, Egyed A (2020) Machine learning for software engineering: a systematic mapping. arXiv preprint arXiv:2005.13299
26. Shippey T, Bowes D, Hall T (2019) Automatically identifying code features for software defect prediction: using AST n-grams. Inform Softw Technol 106:142–160
27. Verma S, Dickerson J, Hines K (2020) Counterfactual explanations for machine learning: a review. arXiv preprint arXiv:2010.10596
28. Wang S, Liu T, Tan L (2016) Automatically learning semantic features for defect prediction. In: 2016 IEEE/ACM 38th international conference on software engineering (ICSE). IEEE, pp 297–308
29. Xu Z, Li L, Yan M, Liu J, Luo X, Grundy J, Zhang Y, Zhang X (2021) A comprehensive comparative study of clustering-based unsupervised defect prediction models. J Syst Softw 172:110862 (2021). https://doi.org/10.1016/j.jss.2020.110862

From Dependability to Security—A Path in the Trustworthy Computing Research

Shuo Chen

Abstract The societal importance of trustworthy computing has become more and more obvious. It has two distinguishable yet related aspects: dependability and security. In this chapter, I will explain the commonality and difference of the two, and use my own experience as an example to show how a researcher grows his/her expertise through the dependability research and the security research.

1 About Trustworthiness

A fundamental question in computing is how to establish trustworthiness of computational results produced by a real-world system. When discussing the concept of trustworthiness, we must consider the adversary model. The adversary can be phenomena in the nature (e.g., hardware transient errors, communication disruptions and human errors) or intentional human attackers. The former is often considered as the adversary model for *dependability*, and the latter is for *security*. From the perspective of the system designers, implementers and operators, trustworthiness means that the system should be able to withstand these adversaries.

Although the two adversary models are distinguishable, the insights from dependability research and security research are coherent. For example, bit-flip is a basic adversary model, originally in the context of dependability. However, people's understanding about bit-flip has been evolving over a long time. It is now a topic frequently studied in the security community. In addition to the adversary model, formal verification and distributed consensus are also topics evolving from dependability to security. In the rest of this chapter, I provide my perspective in these areas.

S. Chen (✉)
Microsoft Research Asia, Beijing, China
e-mail: shuochen@microsoft.com

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_4

2 The Evolution of the Bit-Flip Adversary Model

A useful methodology to evaluate system dependability is fault injection. A fault injector simulates faults that the target system may encounter when it is deployed in the real world. Fault injectors often implement the bit-flip functionality. The functionality targets registers, data memory or code memory. During a test execution of a program, a bit is chosen to be flipped based on a pre-determined distribution. The execution is then resumed. There are two scenarios with high probabilities. The first is that the execution finishes with a correct result. This scenario is often referred to as "fault not manifested". The second high probability scenario is when the execution results in a crash or other exceptions. This is often referred to as "fail silence". Usually, "fault not manifested" and "fail silence" are considered the expected outcome without serious bad consequences. However, there is a non-negligible probability that the execution finishes but produces an incorrect result. This is often referred to as "fail-silence violation". It is the most interesting scenario for investigation.

2.1 Security Consequences Caused by Bit-Flips

My initial knowledge about the security consequences of random bit-flip faults comes from Boneh et al.'s paper in Eurocrypt'97 [1]. The paper shows that several cryptographic systems will be broken if bit-flips can be intentionally introduced during certain phases of the cryptographic computations. For example, an implementation of RSA is based on the Chinese Remainder Theorem (CRT). Boneh et al. show that if the attacker can introduce a bit-flip fault to cause the RSA algorithm to produce an erroneous signature of a message, and repeat the algorithm without the fault to produce the correct signature of the same message, then the secret signing key will be recovered.

My first two papers, published in 2001 and 2002, investigated the security consequences of bit-flips target Internet server programs (e.g., FTP and SSH) [2] and firewall programs (e.g., IPChains and Netfiler) [3]. My co-authors and I conducted fault injection experiments to show the existence of non-negligible probabilities of fail-silence violations resulting in security consequences. For example, injected faults could cause firewall programs to skip packet-filtering rules, or cause FTP's authentication to be bypassed. While these consequences are not surprising, the fact that the probabilities are non-negligible is.

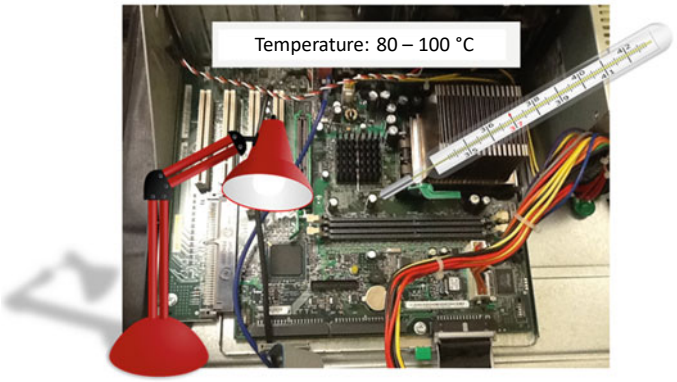


Fig. 1 Using a spotlight to introduce memory errors to JVM

2.2 *Fault Injection as a Weapon*

The papers mentioned above investigate the security consequences with the presence of injected faults, but do not discuss how the faults can be injected in the real-world settings. In this sense, the weaknesses discussed in these papers are not end-to-end exploitable security vulnerabilities.

Our work motivated Govindavajhala et al. to conduct a really surprising experiment in 2003 to show that memory faults can be intentionally injected by heating the PC with a spotlight in a close proximity [4]. (Note that the cover of the PC is removed, so the spotlight is more effective in raising the temperature of the PC's components.) The experiment shows that when the temperature is in the range of 80–100 °C, isolated and intermittent memory errors occur. The authors use this effect to target Java VM (JVM), of which the security assurance crucially depends on type safety. With the presence of memory errors, type safety no longer holds. This means that when the attacker's Java program runs on the JVM, the attacker can take control of the JVM, thus execute arbitrary native code on the victim machine. It is estimated that a single-bit-flip can give a 70% probability for the attack to succeed (Fig. 1).

2.3 *Software Memory Bugs as a Weapon*

The aforementioned research studies give an important insight for a more comprehensive understanding about memory bugs. Before 2005, the attacks exploiting software memory bugs, such as stack overflow, format string vulnerability and heap corruption, focus on the control flow: they use the bugs to rewrite important data that determine the victim program's control flow, e.g., return addresses and function pointers, so that the control flow jumps to an arbitrary binary code supplied by the attacker. They

Table 1 Source code of `getdatasock()`

```
FILE * getdatasock( … ) {
...
seteuid(0);
setsockopt( ... );
...
seteuid(pw->pw_uid);
...
}
```

are referred to as the *control-data attacks*. In response to this attack pattern, many defensive techniques are proposed against them in the research community. Some protect return addresses, such as StackGuard [5] and Libsafe [6]; some rely on control flow integrity for security, such as system call based intrusion detection techniques [7–13], control data protection techniques [14–16], and enforcement mechanisms for non-executable memory [17, 18].

Despite the research community's familiarity of the control-data attack, it is reasonable to ask whether the dominance of control-data attacks is due to an attacker's inability to construct non-control-data attacks, i.e., attacks that do not alter any control-data but still cause security consequences as serious as the control-data attacks. My co-authors and I understood from our previous research that, given a real-world program, its built-in code logic is already susceptible under the bit-flip adversary. In other words, we understood that even if the victim program's control flow is intact, when the code runs on the data slightly corrupted, the consequence can be devastating. This insight might be natural to the dependability community, but was fairly surprising in the security community.

In 2005, we published a paper with the title "Non-Control-Data Attacks Are Realistic Threats" [19]. The paper shows that many types of data, other than control-data, are also crucial to security, including configuration data, user input, user identity data and decision-making data. For example, Table 1 shows the source code of a function in WU-FTPD, which is one of the most widely used FTP servers. WU-FTPD has a format string vulnerability that can be triggered when receiving a "Site Exec" command. Like most other format string vulnerabilities, this vulnerability allows the attacker to overwrite the value of an arbitrary memory location specified by the attacker. Essentially, this vulnerability is a memory fault injector. The function in Table 1 is named `getdatasock`. What it does is to temporarily set the effective UID of the process to the root UID. This is fulfilled by calling `seteuid(0)`. Then, the code does certain operations with the root privilege, such as calling `setsockopt`. In the end, the code restores the effective UID of the process to the user's UID, which is stored in `pw->pw_uid` on the heap. Now, consider what can happen when a format string vulnerability exists. The attacker can exploit the vulnerability to overwrite `pw->pw_uid` to 0, then call function `getdatasock`. The consequence is that `seteuid(pw->pw_uid)` does not restore the process's effective UID, so the attack stays at the root privilege level. All files in the filesystem can be overwritten, including the crucial ones for user authentication, such as /etc/*passwd*

Table 2 Code of `servconnection()`

```
int        serveconnection(int
sockfd) {
char  *ptr;//pointer  to  the
URL.
// ESI is allocated
// to this variable.
...
1: if (strstr(ptr,"/..”))
            reject the request;
2: log(...);
3: if (strstr(ptr,"cgi-bin”))
4:          Handle CGI request
...
}
```

and /etc/*shadow*. This means that the attack obtains the total control of the victim machine.

Another example is about the buffer overflow vulnerability in an HTTP server called GHTTPD. Buffer overflow is also like a memory fault injector, which can overwrite data on the stack. The vulnerable code is shown in Table 2. Function `servconnection()` calls another function `log()`, which contains a buffer overflow bug. A pointer variable `ptr` is on the stack, so it can be overwritten by the attacker because of the bug. Now the question is how the attacker can take control of the victim machine. Although the source code of `servconnection()` is very long, we show two important states in line 1 and line 3. Line 1 rejects any URL that contains a "`/..`" substring. Line 3 implements the CGI functionality of the HTTP server, which allows an HTTP request to invoke an executable on the server. For the security reason, all the invocable executables are stored in a specific path, e.g. `/usr/local /ghttpd/cgi-bin`. An HTTP request http://foo.com/cgi-bin/bar will invoke the executable `/usr/local/ghttpd/cgi-bin/bar`. The checking in line 1 is crucial for the CGI functionality. Suppose a request http://foo.com/cgi-bin/../../../../bin/sh is not rejected, the executable */bin/sh* will be executed, giving the user a command shell. The attacker hence gets the same privilege as the HTTP server. To carry out the attack, the attacker sends a long HTTP request in which the first part is to exploit the buffer overflow bug in order to overwrite the value of `ptr` to be the address of the second part of the request. The second part is the string containing "*/../../../../bin/sh*". This accomplishes the attack.

Our paper [19] investigates other memory bugs in real-world programs. It gives a substantial amount of evidence to show that, when a memory bug allows data corruption, the victim program's existing semantics are usually sufficient to let the attacker get a total control.

2.4 *Rowhammer—A Bit-Flip Security Threat in DRAM*

Since 2014, the security threat caused by bit-flips in DRAM has become an extensively researched topic. The threat and the corresponding exploits are referred to as Rowhammer. The root cause of Rowhammer is the scaling-down of the DRAM process technology. DRAM cells become increasingly likely to charge and discharge between each other, thus have a non-negligible probability to result in bit-flips. This phenomenon was initially described by several patent disclosures by Intel, then studied by the research community. Authors of reference [20] study specifically DDR SDRAM. Figure 2 illustrate the rows of memory cells in the DRAM. One of the rows is the victim row that the attack wants to introduce bit-flips into. It is sandwiched between two adjacent rows. The study demonstrates that the bit-flip probability of the victim row can be substantially increased if the attacker frequently activates the two adjacent rows. Therefore, suppose the attacker can know sufficiently the data contents in these three rows, purposeful bit-flips can be introduced.

There are several follow-up studies based on reference [20]. For example, people understand that ECC (error correcting code) is a technique to mitigate bit-flips, so it is natural to ask whether the Rowhammer threat exists in ECC-protected DRAM. Cojocar et al. conduct a study to reverse-engineer the ECC mechanism. They construct a new Rowhammer attack which can succeed in certain ECC-protected DRAMs [21]. In the separate study, Cojocar et al. develop a methodology to evaluate how cloud servers are vulnerable to the Rowhammer threat [22].

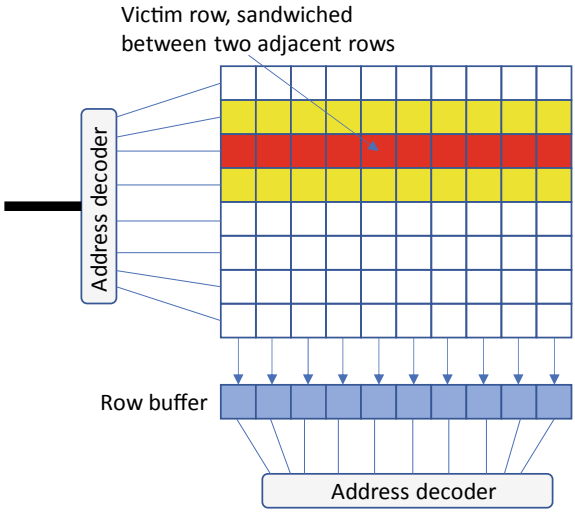


Fig. 2 An illustration about the Rowhammer attack

3 Formal Methods

Formal methods are an important rigorous approach to enhance correctness of a system. My initial knowledge about formal methods was from the reliability context. For example, Rosu et al. developed a formal approach to check the measurement unit (e.g., imperial vs. metric) safety policies for mission-critical programs, such as those written by NASA JPL (Jet Propulsion Laboratory) [23]. This type of safety violations (e.g., an imperial quantity is added to a metric quantity) can hide deeply in a complex program developed by many teams. It is impossible to exhaustively test all the execution paths of the program. Formal methods provide a unique power to statically examine the program to expose bugs with a level of completeness with respect to a given abstraction.

A good example of formal methods for software reliability is the Static Driver Verifier (a.k.a. the SLAM technology) for Windows [24]. Windows needs to accommodate a huge number of device drivers, which run in the kernel space. In the past, Microsoft did not have a quality control mechanism to ensure that the drivers were reliable, so the kernel panic (a.k.a. "blue screen") frequently occurred. Static Driver Verifier enabled Microsoft to implement a Windows driver certification program—a driver succeeding in the verification would be digitally signed by Microsoft, and users were strongly discouraged to install unsigned drivers.

3.1 Formal Methods for Browser Security

Fascinated by the rigorousness of formal methods, I have worked on several security projects that applied formal methods to real-world systems. The first project was about examining Internet Explorer (IE) browser's graphic interface (GUI) logic for security bugs [25]. We studied the GUI code and built a formal model to describe how a user (potentially an attacker) could use Javascript and HTML to spoof the browser's address bar and the status bar. For example, Fig. 3 shows the consequences of two bugs that we discovered. The status bar spoofing bug allowed the attacker to construct a hyperlink in an email. When the user examined the target URL of the hyperlink, the status bar showed https://www.paypal.com, the actual target was the attacker's website. The address bar spoofing bug allowed the attacker to construct a page which could make the address bar and the content window out of sync, so that the address bar (including the SSL certificate) showed https://www.paypal.com, but the content window displayed the attacker's website. Obviously, the combination of status bar spoofing and address bar spoofing would make a powerful phishing attack.

In this work, we used the Maude rewriting logic system [26] to model IE's GUI logic, including the mouse event handling logic and the address update logic during navigation. In the end, we discovered thirteen GUI spoofing bugs in IE 6, eleven of which were fixed when IE 7 was released.

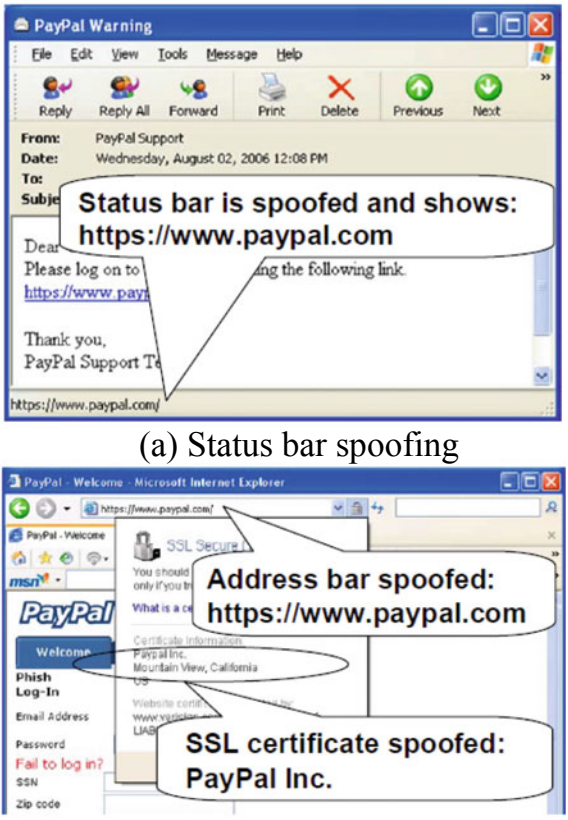


Fig. 3 Browser's GUI security bugs

3.2 Formal Methods for Authentication Protocols

Formal verification was the core technology in my research project that explicated the security assumptions of authentication SDKs. Major cloud providers, such as Facebook, Google, and Microsoft, provide single-sign-on authentication services (SSO) for website developers to integrate. With SSO, a website does not need to implement its own authentication infrastructure, but only needs to call an SSO service that it trusts. The SSO service is called the identity provider, or IdP. The website is called the relying party, or RP. The security goal is for the RP to authenticate the client as "Alice", if the client is able to authenticate to the IdP as "Alice".

Identity provider companies release SSO SDKs, and publish developer's guides to show the sequence of steps to integrate them into website code. However, an important question remains: if developers follow the guides in reasonable ways, will the resulting applications be secure? Our study shows that the answer today is "No". Many apps built using the SDKs we studied have serious security flaws. This is

not due to direct vulnerabilities in the SDK, but rather because achieving desired security properties by using an SDK depends on many implicit assumptions that are not readily apparent to app developers. These assumptions are not documented anywhere in the SDK or its developer documentation. In several cases, even the SDK providers are unaware of the assumptions.

The goal of our work [27] is to systematically identify the assumptions to use an SDK to produce secure applications. Our approach involves a combination of manual effort and automated formal verification. Any counterexample found by the verification tool indicates either (1) that our system models are not accurate, in which case we revisit the real systems to correct the model; or (2) that our models are correct, but additional assumptions need to be captured in the model and followed by application developers. The explication process is an iteration of the above steps so that we document, examine and refine our understanding of the underlying systems for an SDK. In the end, we get a set of formally captured assumptions and a semantic model that allow us to make meaningful assurances about the SDK: an application constructed using the SDK following the documented assumptions satisfies desired security properties.

The formal language we used in this study is Boogie [28]. It is an imperative language, so translating SDK code in a web language (e.g., PHP or C#) to Boogie is straightforward. Figure 4 shows an example PHP function translated into Boogie. The Boogie language allows the programmer to add assertions and invariants. The Boogie verifier will then statically verify whether the assertions and invariants hold in all circumstances.

```
protected function getUserFromAvailableData() {
  if ($signed_request) {
    ...
    $this->setPersistentData('user_id',
                             $signed_request['user_id']);
    return 0;
  }
  $user = $this->getPersistentData('user_id', $default = 0);
  $persist_token =
            $this->getPersistentData('access_token');
  $access_token = $this->getAccessToken();
  if ($access_token &&
        !($user && $persist_token == $access_token)) {
          $user = $this->getUserFromAccessToken();
          if ($user)
            $this->setPersistentData('user_id', $user);
          else $this->clearAllPersistentData();
    }
    return $user;
}
public function getLogoutUrl() {
  return $this->getUrl(
      'www',   'logout.php',
       array_merge(array(
         'next' => $this->getCurrentUrl(),
         'access_token' => $this->getAccessToken(), ), ...));
  }
```

```
procedure   {:inline 1} getUserFromAvailableData() returns (user:User) {
 if (IdP_Signed_Request_Records__user_ID[signed_request]!= _nobody)
   {   ...
     user := IdP_Signed_Request_Records__user_ID[signed_request];
     call setPersistentData__user_id(user);
     return;
   }
   call user := getPersistentData__user_id();
   call persisted_access_token := getPersistentData__access_token();
   call access_token := getAccessToken();
   if (access_token >= 0 &&
     !(user != _nobody && persisted_access_token == access_token)) {
         call user := getUserFromAccessToken(access_token);
         if (user != _nobody) {
             call setPersistentData__user_id(user);
         } else {
             call clearAllPersistentData();
         }
   }
   return;
}
procedure {:inline 1} getLogoutUrl()
  returns (API_id: API_ID, next__domain: Web_Domain, next__API:
         API_ID,   access_token: int)   {
             API_id := API_id_FBConnectServer_login_php;
             call access_token := getAccessToken();
             call   next__domain,   next__API := getCurrentUrl();
}
```

Fig. 4 Example of a PHP function and its Boogie model

4 Distributed Consensus

Another important topic that are originated in the dependability community and later plays an important role in security is distributed consensus. A legendary paper about this topic is the paper titled "The Byzantine General Problem" [29] by Lamport, Shostak and Pease. Despite the interesting title, the core problem studied in this paper is first presented in an earlier paper by the same set of authors. The earlier paper is titled "Reaching Agreement in the Presence of Faults" [30], which is clearly set in the fault tolerance context. The protocols proposed in reference [29] and later papers establish a well-known area in dependability and distributed systems, namely BFT (or Byzantine Fault Tolerance).

The fault tolerance capability of BFT comes from the redundancy of the nodes. However, it is different from other redundancy-based fault tolerance mechanisms. For example, triple modular redundancy (TMR) runs three replicas for a computation, and uses the majority voting to decide the output. TMR tolerates one faulty replica. However, TMR needs to have a component to do the voting. Although it can be substantially simpler than every replica, it can be faulty itself. Unfortunately, TMR cannot tolerate the faults of the voting component. BFT, on the other hand, does not assume any component to be reliable. Instead, the assumption of BFT is about the total number of the faulty components.

From fault tolerance to security. BFT and other distributed consensus protocols play an important role to ensure reliability of cloud platforms. However, it was somewhat unexpected that decentralized computing became the big wave of technological innovations. In this wave, consensus protocols are no longer a reliability mechanism, but form the foundation of decentralized trust.

Computing with decentralized trust enables scenarios that were hard to imagine before. For example, it was surprising that this new computing paradigm could be deployed in the global scale, enabling the worldwide community to issue a new currency (e.g., Bitcoin) without trusting any central bank. It was even more surprising that the new paradigm supported general purpose computing (e.g., by smart contracts on Ethereum). Decentralized trust enables many exciting possibilities, but its core mechanism is distributed consensus.

Consensus protocols can be categorized into two categories. The first is suitable for communities with open membership, which allow everyone to join. The Bitcoin network and the public Ethereum network are such communities. The second category is suitable for consortiums, which are formed by entities with clear identities.

Proof-of-Work (PoW) [31] is a representative consensus protocol in the first category. The goal of the protocol is to ensure that no member can dominate community. However, the open membership makes it impossible to base the protocol on identities, because a member can create an arbitrary number of identities. The core idea of PoW is to base the consensus on every member's actual computational power, which cannot be arbitrarily created. The PoW mechanism in Bitcoin is illustrated in Fig. 5. Each block contains the SHA256 hash value of its previous block, as well as the transactions contained in the current block. In addition, there is a nonce value, which

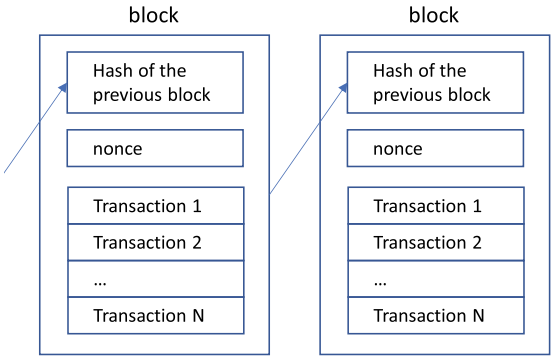


Fig. 5 Proof of work in Bitcoin

is crucial to the PoW mechanism. Any member who wants to construct a valid block to be accepted by the community (i.e., to represent the community consensus) needs to find such a nonce value that makes the hash of the current block begin with a predetermined number of zero bits. Because there is no known algorithm to calculate such a value efficiently, the only way to obtain it is to repeatedly try different value and calculate SHA256. This means that the probability for a member to represent the consensus is proportional to its actual computational power.

Besides the community with open membership, the other type of community is consortium, which consists of members with well-known identities. For example, government agencies, companies, and international organizations can form consortiums. In the consortium setting, traditional consensus protocols are valid. Xiao et al. provide a survey about blockchain consensus protocols, including those used in the consortium settings [32]. The survey covers Byzantine fault tolerant (BFT) protocols and crash fault tolerant (CFT) protocols. It categorizes protocols by different synchrony assumptions. Popular protocols include Raft [33], PBFT [34] and a few others. In 2019, Facebook announced the project to build a consortium blockchain called Libra. The consensus protocol, namely HotStuff [35], is derived from BFT.

5 Summary

Dependability (fault tolerance) is an important aspect of trustworthy computing. It is about investigating adversarial circumstances of a system and designing a mechanism for a system to be robust despite these circumstances. It is often distinguishable from security, for which the adversarial circumstances are intentionally created or controllable by a human attacker. For this reason, the adversary assumptions for security research often appear to be more direct and imminent. They are often deterministic, while the assumptions for dependability research are often probabilistic.

However, both research areas are fundamentally about quality of programming, thoroughness of testing, and good redundancies in design. My research career began in the dependability community. Then I worked on projects about security consequences of faults, later focused on security research. I realize that it is important for researchers to appreciate the commonality of the two disciplines. Among the three areas described in this chapter, the strong relevance of the bit-flip adversary and the distributed consensus was not foreseeable when I was initially exposed to the concepts. There was a decades-long research history of each topic in the dependability community, but the research value was revived stronger than before in the security community in the last 5 years. Regarding formal methods, I was not surprised by their values in security. It was reasonable to anticipate that logic-proof-based approaches would be needed to complement traditional software testing approaches. However, it is still very impressive to see a great amount of new formal techniques indeed make concrete contributions in many security domains.

References

1. Boneh D, DeMillo RA, Lipton RJ (1997) On the importance of eliminating errors in cryptographic computations. In: Proceedings of advances in cryptology: Eurocrypt'97, pp 37–51
2. Xu J, Chen S, Kalbarczyk Z, Iyer RK (2001) An experimental study of security vulnerabilities caused by errors. In: IEEE international conference on dependable systems and networks (DSN), Göteborg, Sweden
3. Chen S, Xu J, Iyer RK, Whisnant K (2002) Modeling and analyzing the security threat of firewall data corruption caused by instruction transient errors. In: IEEE international conference on dependable systems and networks (DSN), Washington DC
4. Govindavajhala S, Appel AW (2003) Using memory errors to attack a virtual machine. In: Proceedings of the IEEE symposium on security and privacy
5. Cowan C, Pu C, Maier D, Hinton H, Walpole J, Bakke P, Beattie S, Grier A, Wagle P, Zhang Q (1998) Automatic detection and prevention of buffer-overflow attacks. In: Proceedings of the 7th USENIX security symposium, San Antonio, TX
6. Baratloo A, Tsai T, Singh N (2000) Transparent runtime defense against stack smashing attacks. In: Proceedings of USENIX annual technical conference
7. Feng H, Giffin J, Huang Y, Jha S, Lee W, Miller B (2004) Formalizing sensitivity in static analysis for intrusion detection. In: Proceedings of the 2004 IEEE symposium on security and privacy
8. Forrest S, Hofmeyr S, Somayaji A, Longsta T (1996) A sense of self for Unix processes. In: Proceedings of the IEEE symposium on security and privacy
9. Feng H, Kolesnikov O, Fogla P, Lee W, Gong W (2003) Anomaly detection using call stack information. In: Proceedings of the IEEE symposium on security and privacy
10. Gao D, Reiter M, Song D (2004) Gray-box extraction of execution graphs for anomaly detection. In: Proceedings of the 11th ACM conference on computer and communication security
11. Giffin J, Jha S, Miller B (2004) Efficient context sensitive intrusion detection. In: Proceedings of the symposium on network and distributed system security
12. Hofmeyr SA, Forrest S, Somayaji A (1998) Intrusion detection using sequences of system calls. J Comput Secur 6(3)
13. Sekar R, Bendre M, Dhurjati D, Bollineni P (2001) A fast automaton-based method for detecting anomalous program behaviors. In: Proceedings of the IEEE symposium on security and privacy

14. Crandall JR, Chong FT (2004) Minos: control data attack prevention orthogonal to memory model. In: Proceedings of the 37th international symposium on microarchitecture
15. Smirnov A, Chiueh T (2005) DIRA: automatic detection, identification and repair of control-data attacks. In: Proceedings of the 12th network and distributed system security symposium (NDSS), San Diego, CA
16. Suh G, Lee J, Devadas S (2004) Secure program execution via dynamic information flow tracking. In: Proceedings of the 11th international conference on architectural support for programming languages and operating systems. Boston, MA
17. Andersen S, Abella V, Data execution prevention. Changes to functionality in Microsoft Windows XP service pack 2, part 3: memory protection technologies. http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2mempr.mspx
18. Otachi E. What is data execution prevention in Windows 10. https://helpdeskgeek.com/windows-10/what-is-data-execution-prevention-in-windows-10/
19. Chen S, Xu J, Sezer EC, Gauriar P, Iyer RK (2005) Non-control-data attacks are realistic threats. In: Proceedings of USENIX security symposium
20. Kim Y, Daly R, Kim J, Fallin C, Lee JH, Lee D, Wilkerson C, Lai K, Mutlu O (2014) Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. In: Proceedings of the international symposium on computer architecture (ISCA)
21. Cojocar L, Razavi K, Giuffrida C, Bos H (2019) Exploiting correcting codes: on the effectiveness of ECC memory against Rowhammer attacks. In: Proceedings of the IEEE symposium on security and privacy
22. Cojocar L, Kim J, Patel M, Tsai L, Saroiu S, Wolman A, Mutlu O (2020) Are we susceptible to Rowhammer? An end-to-end methodology for cloud providers. In: Proceedings of the IEEE symposium on security and privacy
23. Rosu G, Chen F (2003) Certifying measurement unit safety policy. In: Proceedings of the IEEE international conference on automated software engineering (ASE)
24. Ball T, Cook B, Levin V, Rajamani SK, SLAM and static driver verifier: technology transfer of formal methods inside Microsoft. Microsoft Research Technical Report MSR-TR-2004-08
25. Chen S, Meseguer J, Sasse R, Wang HJ, Wang Y-M (2007) A systematic approach to uncover security flaws in GUI Logic. In: Proceedings of the IEEE symposium on security and privacy
26. Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N et al (2002) Maude: specification and programming in rewriting logic. Theor Comput Sci 285(2):2002
27. Wang R, Zhou Y, Chen S, Qadeer S, Evans D, Gurevich Y (2013) Explicating SDKs: uncovering assumptions underlying secure authentication and authorization. In: Proceedings of the USENIX security symposium
28. Boogie: an intermediate verification language. http://research.microsoft.com/en-us/projects/boogie/
29. Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. ACM transactions on programming languages and systems
30. Pease M, Shostak R, Lamport L (1980) Reaching agreement in the presence of faults. J ACM
31. Nakamoto S, Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf
32. Xiao Y, Zhang N, Lou W, Thomas Hou Y (2020) A survey of distributed consensus protocols for Blockchain networks. In: IEEE communications surveys & tutorials, vol 22
33. Ongaro D, Ousterhout J (2014) In search of an understandable consensus algorithm. In: 2014 USENIX annual technical conference (USENIX ATC 14), pp 305–319
34. Castro M, Liskov B (1999) Practical byzantine fault tolerance. In: Proceedings of symposium on operating systems design and implementation (OSDI)
35. Yin M, Malkhi D, Reiter MK, Gueta GG, Abraham I, HotStuff: BFT consensus in the lens of Blockchain. [arXiv:1803.05069] https://arxiv.org/pdf/1803.05069.pdf

Assessment of Security Defense of Native Programs Against Software Faults

Keun Soo Yim

Abstract This chapter explores the possibility of building a unified assessment methodology for software reliability and security. The fault injection methodology originally designed for reliability assessment is extended to quantify and characterize the security defense aspect of native applications. Native application refers to system software written in C/C++ programming language. Specifically, software fault injection is used to measure the portion of injected software faults caught by the built-in error detection mechanisms of a target program (e.g., the detection coverage of assertions). To automatically activate as many injected faults as possible, a gray box fuzzing technique is used. Using dynamic analyzers during fuzzing further helps us catch the critical error propagation paths of injected (but undetected) faults, and identify code fragments as targets for security hardening. Because conducting software fault injection experiments for fuzzing is an expensive process, a novel, locality-based fault selection algorithm is presented. The presented algorithm increases the fuzzing failure ratios by 3–19 times, accelerating the speed of experiment. The case studies use all the above experimental techniques in order to compare the effectiveness of fuzzing and testing, and consequently assess the security defense of native benchmark programs.

1 Introduction

We, dependable computing and fault tolerance community, have long wanted to establish a unified methodology for quantitative assessment of software reliability and security [1–5]. To this end, this study extends a quantitative reliability analysis methodology (i.e., *fault injection* [6]) for software security analysis. Specifically, it demonstrates how an augmented fault injection methodology presented in this study is used to analyze and evaluate the security defense aspects of C/C++ programs. Here, C/C++ programs are used as targets because those are essential

K. S. Yim (✉)
Google, Alphabet Inc., attn yim, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
e-mail: yim@google.com

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_5

building blocks of computer systems (e.g., system software and performance critical user space services).

In mobile cloud computing platforms, various C/C++ application programs and libraries are run using high system privileges. Such privileged, user-space C/C++ programs directly access the application programming interfaces (APIs) of an underlying operating system (OS) kernel and device drivers so as to provide essential services (such as multimedia, networking, database, and general utilities) to other application programs. A critical vulnerability (e.g., a buffer overflow) in a privileged C/C++ program, however, enables attackers to subvert the program and use the obtained system privilege. Because at the time of writing many such C/C++ programs are developed and maintained by third party entities (e.g., as open source software projects), many mobile cloud computing platforms are directly exposed to vulnerabilities introduced in the code bases of upstream open source projects. Thus, before taking any upstream code releases, characterizing and understanding the security defense aspects of the code are one of the key capabilities needed to protect the security of the downstream, integrated mobile cloud software platforms. For example, with a trustworthy security characterization technique, one can select more secure implementations than the other implementations, and also identify certain parts of a given implementation where extra security protection (i.e., hardening) is needed.

This study specifically focuses on characterizing the software *security exploitability*. By the definition used in this study, security exploitability of a program has both the vulnerability metric aspect (e.g., the probability of containing vulnerabilities) and the defense metric aspect (e.g., attack surface size) that are described in [7]. Here, the vulnerability metric is inversely proportional to the security testing coverage. For example, if a program is well tested, then the program is less likely to have vulnerabilities especially when the testing well covers common security defects. Based on this observation, the first research objective of this study (RO1) is *to quantitatively evaluate the effectiveness of fuzzing and structural testing in finding exploitable software faults*. Here, exploitable software fault refers to a software bug or defect that any users can exploit by only altering the input values of a program or the environments that any non-privileged users can control. While gray box fuzzing is generally regarded as an effective technique for finding vulnerabilities and bugs, it was unclear how much vulnerability detection coverage a given set of fuzz tests provide for a target C/C++ program. Structural tests written by domain experts have the same assessment challenge. That is, it is unclear how much extra security testing coverage is provided by given structural tests over gray box fuzzing, and vice versa. Thus, this study presents an assessment technique that can be used for RO1. The technique is useful for guiding the selection, execution, and maintenance of fuzzing and structural testing.

Software faults uncaught by fuzzing and testing are then shipped to users as part of a program binary. Since fail-fast or fail-stop is a desired system dependability property (e.g., to prevent malfunctioning), privileged C/C++ programs often have built-in error detectors (e.g., assertions) in their production binaries. Such built-in error detectors can preemptively detect certain attempts to exploit vulnerabilities. Thus, the second research objective (RO2) of this study is *to assess the security*

defense of a target program against vulnerability exploitation attempts. More specifically, this study quantifies the probability of the built-in error detectors of a target program catching systematic attempts to exploit software faults introduced as a result of classical (e.g., common types of) coding mistakes. This study further shows how to use the measured security defense coverage data of a target program in order to guide the design and extension of the built-in error protection mechanisms of the target program.

This chapter presents an *experimental assessment methodology* that can be used to analyze and evaluate the security exploitability of a given C/C++ program. The methodology can be used to quantitatively evaluate the effectiveness of fuzzing and testing, and the coverage of runtime error detectors of a target program. That is, the methodology allows us to measure two kinds of conditional probabilities: (a) the conditional probability of common software faults (i.e., injected via software fault injection) leading to symptoms of successful exploitations of vulnerabilities (e.g., memory corruption errors); and (b) the conditional probability of exploitable software faults [i.e., a result of (a)] being caught by a built-in error detectors of a target program. These two conditional probabilities are measured by using the following three key techniques:

- ***Software Fault Injection with Gray Box Fuzzing***. In this study, a software fault injection technique is used to inject classical software faults and evaluate the detection coverage of fuzzing. If an injected software fault is caught by fuzzing, then the software fault is exploitable (i.e., showing some characteristics of a vulnerability). To maximize the chances of activating the injected faults, a state-of-the-art, code coverage guided gray box fuzzing technique is used. To further increase the chance of detecting activated faults, dynamic analyzers (e.g., for out-of-bound memory access detection) are used that help us identify a subset of activated software faults that are potentially harmful for the system security. Based on those two sub-techniques, the exploitable software fault count is derived from a given set of generated classical software faults. Then the same experiment process is followed one more time after turning on the built-in error detectors. That is to derive the numerator and denominator of a formula that captures the conditional probability associated with the detection coverage of the built-in error detectors.
- ***Insights from Quantitative Comparison***. The first technique helps us quantify the coverages of both fuzzing and structural testing for any target C/C++ program. For example, one can run fuzzing (or structural testing) and count how many of the exploitable software faults lead to any fuzzing (or testing) failures. By comparing the coverages of those two, one can assess the relative effectiveness of each technique and consequently direct associated security test engineering works. Furthermore, using the error propagation paths of exploitable software faults caught during fuzzing, one can further characterize the security defense aspects of a target program and develop deep source code level insights.
- ***Fault Selection Algorithm to Speedup***. In practice, using software fault injection for fuzzing and structural testing takes a long time. The experiment time depends on the examined software fault count and total test execution time. For

example, if 10,000 fault samples are studied and the test takes on average 1 hour, then the experiment can take ~10,000 machine hours (=417 machine days). If 3% of the injected software faults are activated, then it means 97% of the experiment resources is wasted. To avoid such large computing resource wastes and accelerate the experiment process, this study also presents a fault selection algorithm that is designed to select faults that are *likely* exploitable. The presented algorithm captures and uses fault locality properties by using an iterative fault selection process. The evaluation result shows that the algorithm improves the fault manifestation ratio by 3–19 times over a random sampling technique, and consequently improves the efficiency of software fault injection experiments.

The rest of this chapter is organized as follows. Section 2 reviews the related works. Section 3 describes the presented methodology. Section 4 describes the presented fault selection algorithm. Section 5 describes the experimental setup. Section 6 analyzes the experimental results. Section 7 discusses the implications before concluding this chapter in Sect. 8.

2 Related Work

This section reviews the existing software fault injection techniques that form the basis of the presented methodology. *Fault injection* is an experimental methodology originally designed to validate and characterize error detection and recovery techniques of mission-critical or high availability computing systems. Initially, hardware-implemented fault injection techniques (e.g., hardware pin-level instrumentation tools [8–10] using beam radiations [11, 12], light lamp [13], and supply power voltage disturbances [14]; and circuit-level simulations [15, 16]) are used to validate, characterize, and evaluate various kinds of building blocks of computer systems (such as microprocessor, cache, and main memory). Software fault injection techniques are then developed to emulate various kinds of errors in software states. Since this study focuses on software techniques, let us further classify software fault injection techniques into three sub-types: emulating hardware faults in software states; emulating software faults (e.g., classical software bugs) directly in program source code, program artifacts (e.g., program binaries), or program runtime states; and emulating security attacks.

Emulating Hardware Faults. Techniques emulating hardware-induced errors (e.g., using N-bit flip or stuck-at fault models) in computer architectural states (e.g., registers) or software states are traditionally called SoftWare-Implemented Fault Injection (SWIFI). SWIFI naturally uses fault injection framework software.[1] Some SWIFI frameworks implement memory corruption techniques (FIAT

[1] UIUC DEPEND research group founded by Professor Ravishankar K. Iyer (*Fellow* of AAAS, ACM, and IEEE) has been one of the leading academic research groups in this field. Many SWIFI tools reviewed in this section were built by the DEPEND research group.

[17], Hauberk [18]) or hardware breakpoint-based techniques (NFTAPE [19], Xception [20], and MAFALDA [21]). Some other SWIFI frameworks use WIFI frameworks use `ptrace()` debugging APIs (FERRARI [22]), a binary rewriting technique (DOCTOR [23]), a symbolic execution technique (SymPLFIED [24]), a stochastic activity network (SAN) model [25], or multiple mixed techniques (FINE [26], DEFINE [27], and FTAPE [28]). Among these existing framework techniques, breakpoint-based ones are the most widely used especially for the purpose of validating system software.

Emulating Software Faults. As software reliability gained interests in the information technology (IT) industry, Chillarege[2] and his fellow IBM researchers defined a software fault model, namely ODC (Orthogonal Defect Classification) [29]. ODC has multiple classification axes. One of the axes is for the following five types of software faults: assignment, checking, interface, algorithm, and function. ODC was built based on their two field measurement studies: using IBM DBMS (database management system) running on a particular OS [30] and a Tandem Unix OS [31].

Some studies [32, 33] showed that SWIFI can be used to emulate not only bit flips but also software interface faults. However, [34] showed that the SWIFI results of a software interface fault model is not exactly same as the fault injection results of an ODC-based fault model (e.g., in terms of the failure type distributions). That is partly because software interface faults are not exactly the same as software design faults that are modeled by ODC. The ODC fault model thus enables us to study various types of software design faults. Other studies [35, 36] further evaluated such fault models and showed how to setup the experimental environments for SWIFI or ODC-based fault models, showing the conditions when to use SWIFI or ODC.

We note that in general ODC fault model is useful to gain insights into an analyzed software engineering process. It can, for example, help us improve the software engineering process by classifying the identified software defects and analyzing the statistical causation relationships. Based on the distribution of the software faults in each ODC fault type, one can assess the development stage and identify part of the engineering process that heavily influenced any identified process issues or any observed probability distributions.

The ODC fault model is later extended by Durães and Madeira [37, 38]. Their work considered the fault nature (i.e., missing construct, wrong construct, and extraneous construct) that is added to the ODC classification system as an additional axis. Their field studies show that a large portion of software faults seen in the real world are due to omissions (i.e., missing construct) or wrong constructs. Based on that observation, [39] refines the software fault model for fault injection applications. The selected fault types (e.g., omission faults) are relatively straightforward to emulate by using a binary translation technique (e.g., by skipping some instructions). A software fault injector that implements this extended fault model is G-SWFIT [37, 40]. G-SWFIT is a post-link-time binary translator that looks for specific instruction sequence patterns (e.g., procedure calls without return value), checks specific conditions (e.g., identified

[2] Ph.D. alumni of UIUC DEPEND group and *IEEE Fellow* for the contributions on software reliability.

call is not the only statement of the caller function), and finally rewrites the binary to inject a modeled software fault.

There are other software defect models used for mutation testing [41] and software engineering studies. FAUST [42] used the control flow, array boundary, computational, and post/pre increment/decrement software fault types.[3] Reference [43] used the unaligned pointer, aligned bridging pointer, aligned looping pointer, memory leak, and blocked thread fault types. References [44, 45] used other common software fault types in order to evaluate the specific dependability aspects of file system cache and DBMS, respectively. Reference [46] changed single instructions to emulate various programming errors (e.g., uninitialized local variables), while [47] used a null pointer fault, which is backed by the common bug analysis. Because those defect types were empirically chosen but sometimes lack statistical evidences, this study uses afore described extended ODC fault model.

Emulating Security Attacks. Attack injection approach can be used to assess the protection coverage of security techniques of target software systems [48]. For example, in [49], the coverage of intrusion detection system (IDS) for web servers was studied by injecting realistic vulnerabilities in web applications and then emulating attacks that can exploit the injected vulnerabilities. In that study, its vulnerability model is described in the application-level, making it easy to understand and use the model. It, on the other hand, limits its application scope to specific application attack types on a particular application program type (e.g., SQL injection and cross-site scripting attacks [50] against web services). That study is designed to evaluate the coverage of a separate protection system (e.g., IDS) and thus is not for direct evaluation of the security of a target software itself (e.g., web application) or associated tests.

DBench [51] provides well-defined availability, feedback, and stability benchmarking procedures for various kinds of software. It injects faults into nearby components (such as hardware, OS, middleware, or applications) that are interacting with a system under benchmark (SUB) which can be a native application program in the context of this study. The presented assessment methodology is different from attack injection and DBench because this study directly injects software faults into the SUB. A key benefit of the presented methodology is that it helps us quantify specific security aspects of the SUB without having to use any benchmark targets (BTs).

Fault Selection Strategies. Let us then classify fault injection techniques as a function of the fault selection strategy. Typical SWIFI experiments select a subset of faults by using random sampling or other statistical sampling techniques (such as stress or path-based selection techniques [52]) because SWIFI can derive a numerous number of software faults. On the other hand, fault injection experiments using a software fault model (e.g., G-SWFIT) can examine all the generated software faults. In practice, when a large number of software faults are generated (e.g., because the program binary size is large) or there is a constraint in the experiment time

[3] In other words, mutant types.

or resources, G-SWFIT experiments can also examine only part of the generated software faults by using some sampling techniques (e.g., a random sampling).

Because many generated software faults are caught by existing test cases, [40] presented two machine learning (ML) techniques in order to select software faults that are unlikely covered by available test cases. Because the presented methodology benefits from the maximized fault manifestation ratios (e.g., faults that can be caught by fuzzing but not testing) and the reduced fault injection experiment time and resources, this study further presents the locality-based software fault selection algorithm.

3 Design

This section presents the novel coverage assessment methodology of security defense mechanisms of C/C++ programs. Figure 1 gives an overview of the methodology.

3.1 Software Fault Injection to Assess Fuzzing Coverage

In the past, fuzzing was typically used to find unknown vulnerabilities in target software and consequently estimate the target software security level. While fuzzing can discover many critical vulnerabilities, it generally takes a long time to find all or most of the vulnerabilities. As a result, in practice only part of the vulnerabilities is identified by fuzzing, resulting in providing insufficient sample data for large-scale security assessment studies. Because of this tradeoff, the cost of running fuzzing against a target program is often unacceptably high, especially when the purpose

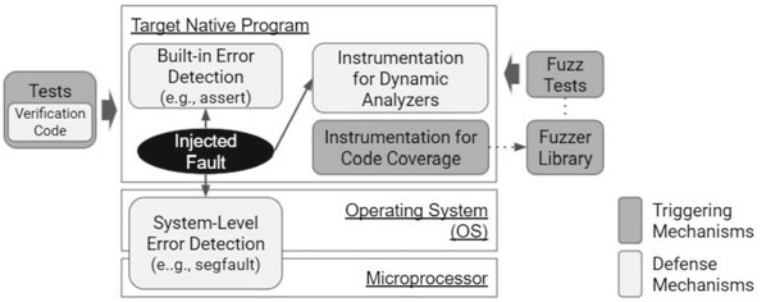


Fig. 1 Overview of the presented assessment approach. Software faults injected into the source code of a target native application program are triggered by tests and fuzzing and detected by multiple error detection mechanisms. The ratio of detected software faults and total injected software faults reveals the strength of the security defense mechanisms of a target application program

is security assessment. That is, assessment approach trying to find all the hidden software defects is costly from a practical usage point of view.

Instead, the presented methodology assesses the detection coverages of security defense techniques (e.g., fuzzing) and uses that to assess the level of security defense of a target program. Specifically, this study uses a software fault injection technique and injects realistic software faults into the target program source code. The software fault injection technique uses an extended ODC software fault model. The used fault types are: MFC (Missing Function Call), MVIV (Missing Variable Initialization using a Value), MVAV (Missing Variable Assignment using a Value), MVAE (Missing Variable Assignment using an Expression), MIA (Missing `If` construct Around Statements), MIFS (Missing `If` construct plusss Statements), MIEB (Missing `If` construct plus statements plus `Else` Before statements), MLAC (Missing Logical AND Clause in branch condition), MLOC (Missing Logical OR Clause in branch condition), MLPA (Missing small, Localized Part of the Algorithm), WVAV (Wrong Value Assigned to Variable), WPFV (Wrong Variable used in Parameter of Function call), and WAEP (Wrong Arithmetic Expression in Parameter of Function Call) fault types [38].

The presented methodology then uses a gray box fuzzing technique to uncover the injected software faults. In this way, any experimenter can quantitatively evaluate the coverage of fuzzing that is used for a target program (e.g., the detected software fault count over the injected fault count). All the baseline fault types (i.e., the extended ODC fault model summarized in Sect. 2) are used to further characterize the detection coverage as a function of fault type. While software fault injection and fuzzing are both common, to the best knowledge of the author of this chapter, this study is the first work that uses software fault injection for the assessment of fuzzing coverage.

To maximize the chance of activating the injected software fault, the presented methodology employs LLVM `libFuzzer`[4] fuzzing framework, which uses `Sancov` (Sanitizer Coverage) to measure the code coverage of a target program during fuzzing, realizing code coverage-guided gray box fuzzing. `Sancov`, for example, can instrument a target program binary so that at the edge of each basic block, the instrumentation code can update an in-memory bitmap table (per segment) and accurately track the execution count of each basic block. Based on this, `libFuzzer` library generates an input data that is a variable-length string (or uses a provided corpus as initial seed data) and feeds the generated string to a user-written fuzzing logic function, which converts the input string to function calls. The `libFuzzer` library measures the code coverage of each input data and generates a next input data set by using a genetic algorithm variant (e.g., crossover and mutation operations). That procedure is repeated until a software failure is seen or a certain limit (e.g., timeout) is reached.

[4] `libFuzzer`, https://llvm.org/docs/LibFuzzer.html.

3.2 Classification of Fuzzing Failure Types

During fuzzting, as shown in Fig. 1, any error propagated as a result of the activation of an injected software fault can be caught by: (a) built-in error detectors of a target program, (b) underlying system-level error detectors (e.g., segfault by MMU sent as a signal), or (c) dynamic analyzers (e.g., AddressSanitizer and LeakSanitizer [53]) used as part of the fuzzing framework.

Because software faults detected by (a) built-in error detectors or (b) system-level error detectors are detected by the target program, those faults do not easily lead to a successful security attack at runtime even if there is an attempt to exploit them. Software faults detected by (c) dynamic analyzers indicate critical error propagation paths. Thus, that can be used to assess the fail-stop property of a target program (e.g., denial-of-service attacks) and identify unprotected control or data flows that can be suggested as extra protection targets. The presented methodology uses various available dynamic analyzers (e.g., for out-of-bound memory access or memory leak detection) that are implemented by instrumenting a target program binary. As a result, this helps us identify a subset of the injected software faults that are uncaught by the built-in protection mechanisms of a target program but still potentially harmful for system security.

In order to particularly improve the failure type classification accuracy of exploitable software faults, dynamic analyzers are used. The employed dynamic analysis technique (e.g., AddressSanitizer) preemptively detects various types of runtime errors regardless of whether those errors are eventually caught by the built-in error detectors. For example, in each of the built binaries, a dynamic analyzer places instrumentation routines before every load or store instruction. That is to check the address operand and detect various kinds of memory overflow and underflow errors. A software fault is classified as an exploitable fault if its failure type (i.e., the type of a dynamic analyzer that detects its error) is under a certain category known as a typical milestone or symptom of a successful vulnerability exploitation attempts. Such failure types include: memory leak error, corruption of a memory copy function parameter, free of a non-allocated memory block, segmentation fault, stack overflow, heap overflow, and buffer overflow of a global variable. For example, let us assume a software fault causing an out-of-bound memory access error up on an activation. Attackers can exploit the software fault to: conduct a memory corruption attack; indirectly change the control (e.g., return address or stack pointer) or non-control data of a target program; and eventually subvert the program. Here, we note that the buffer overflow error is a key milestone of a successful vulnerability exploitation.

A large portion of injected software faults is benign and thus does not lead to any meaningful steps (e.g., memory corruption) towards a successful attack. Thus, this study focuses on analyzing the portion of software faults caught by the built-in error detectors, system-level error detectors, dynamic analyzers of fuzzing, and test verification mechanisms of testing. The presented methodology uses the measured failure type distribution to derive the numerators and denominators of formulas that

capture the coverage of the security defense mechanisms that a target program is equipped with.

3.3 *Quantitative Evaluation*

For a target program, the presented methodology assesses the coverages of both fuzzing and structural testing (i.e., unit, component, and integration tests written by domain experts). It then compares the coverages of those two types of defense techniques in order to build deep insights and increase the assessment confidence. While both fuzzing and structural testing are imperfect and their coverages depend on various factors (such as the amount of efforts put), quantitatively comparing both helps us assess the relative strength and effectiveness of each technique, and guide where to direct future security test engineering effort.

Figure 2 shows the overall flow of the presented quantitative evaluation process. The main input is the source code of a target program. The software fault injector (or mutator) is used to generate various software fault patches. It uses the fault selector module to select a subset of software faults for experiments. The target program is then patched by each of the selected software fault patches and compiled into a set of program binaries. Two kinds of binaries are generated: one for structural testing (i.e., mutated binaries where each binary has a selected software fault) and the other for fuzzing (i.e., mutated fuzzer binaries where each binary has a selected software fault, the fuzzing logic, the fuzzing library, and the instrumentation code needed for fuzzing).

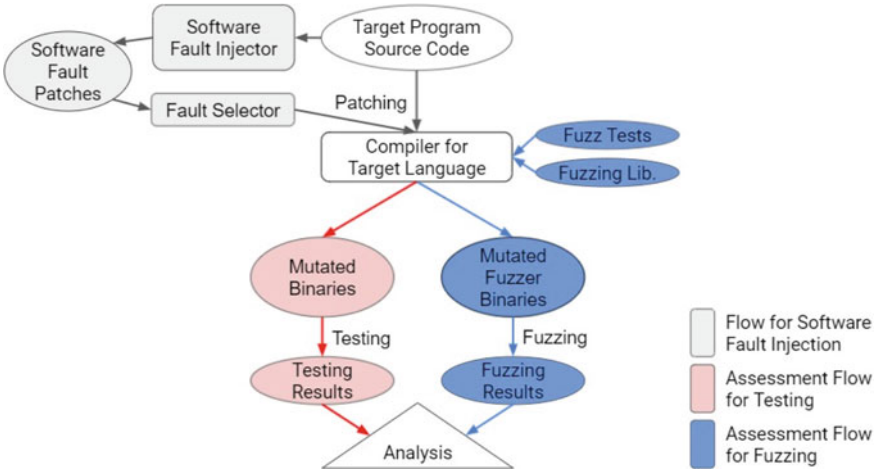


Fig. 2 Quantitative evaluation flows of two kinds of security defense mechanisms (i.e., testing and fuzzing)

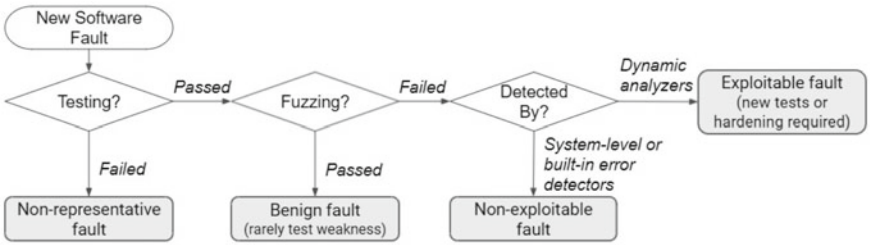


Fig. 3 The presented software fault classification process

The presented methodology runs the testing and fuzzing binaries and collects the testing and fuzzing results. It compares and analyzes the results in tandem. Figure 3 shows how an experiment result is classified. If all tests pass, then it means that the injected software fault is a *benign software fault* or there is a *weakness in the tests.* If a software fault breaks a test, the software fault is *not representative* [40] so the software fault is simply skipped. Another case is for an *exploitable software fault* that passes all the tests but is detected by the fuzzing.

3.4 Framework

The fault injector used in this study is a source-to-source translator where the input and output data is C/C++ program source code. Figure 4 shows the high-level flow of the fault injector where input data is the path of a source file. The fault injector reads and parses a given source file and other files that are directly or indirectly included in the given source file. It then runs the C/C++ preprocessor to generate a preprocessed source file that has the source code of all the included files. The fault injector then reads the preprocessed file and builds an abstract syntax tree (AST), which is used to search for patterns of each of the modeled fault types. For each of the identified fault injection target locations, the fault injection tool mutates the constructed AST and generates the mutated source code in the form of a source code patch (that contains only the information about how to update the original input file). Because the tool

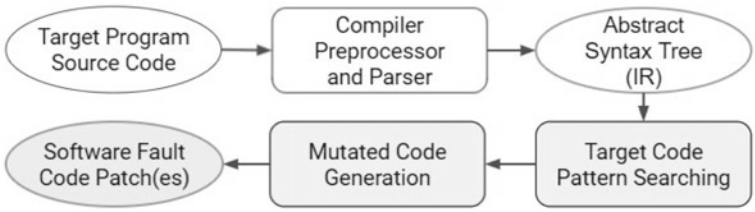


Fig. 4 Overview of the used software fault injector where IR stands for intermediate representation

identifies various fault locations from the source code, it generates various kinds of patches. We note that in its implementation, the tool is designed to generate patches in such a way that avoids compiler warnings, while correctly emulating every modeled software fault type.

4 Optimization

This section describes the presented locality-based fault selection algorithm. The algorithm is designed to accelerate software fault injection experiment by increasing the fault severity and fault activation ratio. To this end, this study defines and uses the two kinds of fault locality properties:

- *Spatial Locality in Fault Sensitivity*. If a software fault is activated and leads to a testing failure, some other faults injected into a nearby program entity are likely to manifest. The granularity of a nearby entity can be the entire code of the same function, all contents of the same source file, or all source code files in the same directory. The optimal granularity depends on a target program. By default, the function granularity is used in this study.
- *Temporal Locality in Fault Sensitivity*. If a software fault is activated and leads to a testing failure, the same fault is likely to manifest again even if the target program is changed or updated (e.g., new release versions).

We note that the software fault locality defined and used in this study directly deals with the fault sensitivity, while the definitions in [54] capture many other parameters (e.g., defect birth rate). Based on those two fault locality properties, a locality-based fault selection algorithm is designed as an iterative process. In every iteration, the algorithm selects N software faults from the software fault set S_{fault} that initially keeps all the generated software faults. Let us assume that in i-th iteration, k_i faults are manifested and causes testing failures where $0 \leq k_i \leq N$. The algorithm reads the location information (i.e., source file path and function name) L_i of each of the k software faults. It then removes the examined faults (F_i) from S_{fault}.

Then the algorithm selects faults (F_{i+1}) for the next iteration, $(i + 1)$-th. A user-provided parameter (r) is used to specify the percentage of faults (i.e., rN) to select from S_{fault}. The specified percentage of faults is selected from the software faults that are close to at least one location kept in L_i. That means that the rest $(1 - r)N$ software faults are selected from the other part of S_{fault} where the locations of the software faults are not close to any of the locations kept in L_i. This process is repeated until S_{fault} becomes empty or the experimenter wants to stop. In this way, the algorithm guarantees that from the second iteration, $r \times 100\%$ of the examined faults are selected based on the locality.

There can be many variants of this algorithm. For example, one can use a ranking scheme or a weight metric in such a way that more popular functions can be selected

more frequently. Here, the popular functions may mean frequently executed functions, according to performance profiling data. Alternatively, one can alter this algorithm so that $F(k_i)$ can keep the entire history (e.g., $F(k_{i+1})$ includes $F(k)$ and the new software faults). Then $F(k_i)$ always keeps all the selected, manifested faults seen until the $(i-1)$-th iteration. As a result, when selecting faults for the next iteration, it considers the locations of all previously selected software faults (not just the ones selected and manifested in the last iteration).

5 Experimental Methodology

This section describes the experimental setup.

1. *libarchive*, a library to read and write streaming archives in a variety of formats
2. *libpng*, a reference library for supporting the PNG (Portable Network Graphics) format
3. *openssl*, a general-purpose cryptography library specialized in TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols
4. *sqlite*, a database engine library for SQL (Structured Query Language)

Benchmark Programs. Table 1 lists the four selected native application programs:

Initially these four programs were compiled with their built-in error detectors. For example, because *sqlite* uses *assert*() defined in *assert.h* file, its compilation flags are configured in such a way that undefines `NDEBUG` macro and enables the assertions. For each benchmark program, all or part of its source code is selected as the fault injection targets. Selected SLOC (Source Lines of Code) column of Table 1 shows the number of selected source code lines per program. For *libarchive*, all the *.c* and *.h* files in *libarchive* sub directory that keeps the core *libarchive* engine are selected. For *libpng*, all the top-level *.c* and *.h* files are selected. The remaining unselected files are in *contrib*, *projects*, and *scripts* sub directories. For *openssl*, all the *.c* and *.h* files in *ssl* sub directory are selected because that keeps the core SSL code. Finally for *sqlite*, the entire program is selected because there are only two large files (*sqlite*3.*c* and *sqlite*3.*h*).

Table 2 shows how many software faults are then generated for the selected portion of the source code of each benchmark program (see T columns in the table). As a baseline, between 40 and 200 software fault samples are randomly selected per fault

Table 1 Benchmark programs used for evaluation

Program	Source repository	Version	Selected SLOC
libarchive	http://www.libarchive.org	v3.0.4	113,930
libpng	http://sourceforge.net/projects/libpng	1.2.59	28,562
openssl	http://github.com/openssl/openssl.git	1.0.2 m	66,391
sqlite	http://www.sqlite.org	2016-11-14	209,806

Table 2 Random fault sampling data (S: selected fault count, T: total generated fault count)

Fault type	*libarchive*		*libpng*		*openssl*		*sqlite*	
	S	T	S	T	S	T	S	T
MFC	96	150	185	890	94	1409	191	3865
MIA	38	96	84	825	75	1757	200	3956
MIEB	7	20	45	290	46	369	200	1055
MIFS	78	95	190	752	192	1816	200	3728
MLAC	59	80	100	421	101	618	200	1987
MLOC	32	36	102	250	140	481	200	948
MLPA	142	798	169	3388	155	3843	200	16,843
MVAE	95	187	190	715	177	1558	200	5615
MVAV	58	80	164	600	122	1280	200	2025
MVIV	50	70	72	75	166	209	200	1002
WAEP	102	135	148	534	56	1049	46	3813
WPFV	81	103	186	1419	163	2949	105	25,810
WVAV	123	144	136	891	122	1600	135	2857
Total	961	1994	1771	11,050	1609	18,938	2277	73,504

type per program (see *S* columns in the table). In case when the generated software fault count is less than 40, at least 5 software faults are selected. For *sqlite*, up to 200 software faults are randomly selected for each omission fault type because about 40% of the omission faults led to no fuzzing failures in the initial experiment.

Fuzzing and Structural Testing. As summarized in Table 3, fuzzing is used for all the four benchmark programs, while structural testing is used for the three programs (*libarchive*, *libpng*, and *openssl*).[5] Every used fuzzing defines a `LLVMFuzzerTestOneInput()` function that converts input byte strings provided by the `libFuzzer` library to target program API calls. The used fuzzing code has 58 source code lines for *libarchive*, 131 for *libpng*, 37 for *openssl*, and 85 for *sqlite*. The used structural tests are mostly end-to-end functional tests. For *libarchive*, all of its tests invoked by `make check` command are used. For *libpng*, its `pngtest` binary is used to run the tests while using a *.png* file as input data. For *openssl*, `util/selftest.pl` script is used to run all the contained tests in sequence and generate test reports. The test reports are used to check the test results (e.g., pass or fail).

To build fuzzer binaries, *clang* v5.0.0 and *gcc* v5.4.0 were used. The used compiler flags include: *'-fsanitize=address'* for address sanitizer and *'-fsanitizecoverage=trace-pc-guard,trace-cmp,trace-gep,trace-div'* that enables gray box fuzzing. For each fuzzing run, the experiment framework waits up to

[5] The used structural tests were contained in the benchmark programs. For the fuzzing, ones available at https://github.com/google/fuzzer-test-suite were used that were developed by a fuzzing team at Google.

Table 3 Characterization of fuzzing and structural testing of benchmark programs

Program	Testing	Fuzzing	Seeds	Dictionaries
libarchive	Evaluated	Evaluated	Used in fuzzing	Unavailable
libpng	Evaluated	Evaluated	Used in fuzzing	Used in fuzzing
openssl	Evaluated	Evaluated	Used in fuzzing	Unavailable
sqlite	Unavailable	Evaluated	Used in fuzzing	Used in fuzzing

1,000 seconds and stores the `stdout` and `stderr` messages to two separate files. Later, the framework analyzes the `stderr` log files in order to check whether each of the fuzzing runs has found any errors or failures. Initially, the framework executes the same fuzzing against the original programs for a sufficiently long time. That is to confirm that the fuzzing does not find any software defects or errors in the used benchmark program versions as long as no software fault is injected.

In order to maximize the fuzzing coverage, either seeds (i.e., initial byte strings) or dictionaries (i.e., input language keywords or magic values) supported by *libFuzzer* and American Fuzzy Lop[6] are used. Specifically, seeds are used for all four benchmark programs, while dictionaries are used for *libpng* and *sqlite*.[7]

In terms of the execution environment, high-performance Linux v4.10.0 machines on a public cloud data center are used for all fuzzing and testing experiments. Another version of a Linux machine is used to generate all the software fault patches.

6 Result

The experimental results summarized in this section demonstrate how the presented assessment methodology is successfully used to: assess the coverage and effectiveness of fuzzing and structural testing (Sect. 6.1); assess the built-in error detectors of target programs (Sect. 6.2); characterize the injected software faults and observed failures as a function of fault type (Sect. 6.3); and evaluate the key benefits of the presented fault selection algorithm (Sect. 6.4).

6.1 Effectiveness of Fuzzing and Testing

The randomly sampled software faults (see Table 2) are used to evaluate the effectiveness of fuzzing and structural testing. It shows that quantitative comparison of the two helps us build deeper insights than assessing only one (fuzzing or testing).

[6] American Fuzzy Lop (AFL), http://lcamtuf.coredump.cx/afl/.

[7] The seeds and dictionaries available at https://github.com/mirrorer/afl/ and https://github.com/google/fuzzer-test-suite were used. Otherwise, seeds were generated by running fuzzing for a sufficiently long period of time (e.g., for *sqlite*).

Table 4 Injected faults caught by fuzzing with (w/) and without (w/o) using seeds and dictionaries

Fault type	*libarchive*		*libpng*		*openssl*		*sqlite*	
	w/ (%)	w/o (%)	w/ (%)	w/o (%)	w/ (%)	w/o (%)	w/ (%)	w/o (%)
MFC	35.4	26.0	1.6	6.0	0	0	14.1	5.2
MIA	50.0	23.7	2.4	6.0	2.7	2.7	13.0	1.5
MIEB	42.9	57.1	0	6.7	8.7	8.7	14.0	0.5
MIFS	42.3	14.1	1.6	4.2	5.8	4.7	9.0	2.0
MLAC	35.6	11.9	1.0	2.0	1.0	1.0	10.0	2.5
MLOC	50.0	12.5	1.0	4.9	0.7	0.7	4.5	0
MLPA	31.7	17.6	1.8	4.1	1.9	1.9	25.0	2.5
MVAE	40.0	25.3	3.7	3.2	5.1	5.1	25.5	1.5
MVAV	44.8	27.6	5.5	3.0	2.5	2.5	17.5	0.5
MVIV	52.0	44.0	4.2	0	2.4	3.0	9.0	1.0
WAEP	40.2	24.5	6.8	2.0	3.6	3.6	0	0
WPFV	46.9	22.2	2.2	1.1	2.5	3.1	7.5	1.9
WVAV	22.8	11.4	7.4	7.4	2.5	2.5	6.8	1.5
Total	38.3	24.5	3.2	3.8	2.9	3.0	14.8	1.6

Fuzzing. Table 4 shows the percentages of injected software faults that are caught by the fuzzing (i.e., caused fuzzing failures) with and without using the seeds and dictionaries. The data shows that the effectiveness of fuzzing depends heavily on the following three parameters: sssss

Program Complexity. The fault detection ratio of fuzzing has large variations and heavily depends on the benchmark program. The used fuzzing was highly effective in *libarchive* that showed 23–52% detection ratios with the seeds and dictionaries and 11–57% without them. The fuzzing is still relatively effective in *sqlite* that shows ~14.8% average detection ratio with the seeds and dictionaries. On the other hand, less than 10% of the injected software faults is detected during the fuzzing in the two other programs, *libpng* and *openssl.*

To understand the underlying correlations, the following two program complexity parameters are further analyzed:

1. Target program size. The total SLOC is 199,177 lines for *libarchive*, 39,223 for *libpng*, 424,216 for *openssl*, and 209,806 for *sqlite*. Even though *libarchive* and *openssl* are medium size programs among the four benchmark programs, their fuzz tests were more effective than *libpng* that is the smallest program. It shows that the total SLOC alone is not a good metric to estimate the difficulty of finding software faults and evaluate the effectiveness of fuzzing.
2. Portion of code selected as fault injection targets. The ratio of selected SLOC and the total SLOC is 57.2% for *libarchive*, 72.8% for *libpng*, 15.7% for *openssl*, and 100% for *sqlite*. When a larger portion of the code is selected for examination (e.g., *libarchive* and *sqlite*), it sometimes shows higher fault detection ratios

especially when seeds and dictionaries are used. We however note that such correlation is not always true (i.e., not a sufficient condition) because *libpng* has a small detection ratio.

Fuzzing Logic. Even though a target program may have a significant implementation complexity and relatively small part of the target program is examined, if its fuzz test is thorough, then it can still achieve a high fault detection ratio. Basically faults injected into a function are detectable by fuzzing if there is call flow path from the used fuzzing logic to that target function. Conversely, faults injected into a function are undetectable or hard to detect if there is no call flow path from the used fuzzing logic to that target function. That shows that those two parameters (i.e., implementation complexity and fuzzing logic thoroughness) both matter. For example, in case of *libarchive*, its fuzzing logic is designed to call only 6 common API functions of *libarchive*. However, because the fuzzing logic targets the most common use case and covers large part of *libarchive* core code where the software faults are injected, the fault detection ratios are high.

The experiments using the seeds and dictionaries showed that carefully tuning fuzzing improves the fault detection ratio. With the valid seeds and dictionaries, *sqlite* shows a significant coverage improvement (i.e., from 1.6 to 14.8%). On the other hand, it is also observed that using the seeds and dictionaries sometimes can reduce the fault detection ratios for certain fault types (e.g., MFC, MIA, MIEB, MIFS, MLAC, MOLC, MLPA fault types of *libpng*).

Fault Type. The impact of fault type on fault detection ratio is then analyzed. Although it depends on the benchmark program, specific fault types show relatively low fault detection ratios. For example, in *libarchive*, MLAC and WVAV fault types show the lowest detection ratios. Also certain fault types showed relatively low fault detection ratios in the other benchmark programs (i.e., MLAC in *libpng*; MLAC and WVAV in *openssl*; and WVAV in *sqlite*). However, no specific fault type is found that always has higher than the average fault detection ratio in all the four benchmark programs. That implies that specific fault types can be either harder to detect or less likely to manifest than some other fault types, while the opposite is not always true.

In this experiment, the random fault sampling technique was used. That is, software faults are uniformly injected into the entire source code of a selected module of each target program. That is to remove two variables in the experiment. Specifically, one of the removed variables is the correlation between what is targeted by fuzzing and where many non-benign software faults are injected into. Because the used software faults are randomly sampled from all the generated software faults (i.e., the same as all the possible fault locations for each fault type), it further removes the fault density parameter from the experiment.

Fuzzing versus Testing. In order to compare the effectiveness of fuzzing and testing, the testing coverage is first evaluated. The tested three programs, *libarchive*, *libpng*, and *openssl* contains some automated tests in their source code repositories. Using each of the selected software faults, the framework builds a program binary and then runs the entire tests against each of the built binaries. The framework then analyzed the stdout and stderr logs and decided whether the tests are passed. Rarely,

some test runs do not finish (i.e., hang failure) that is detected by using a constant timer set in the framework.

Figure 5 shows the fault detection ratios of testing, fuzzing, and both combined.

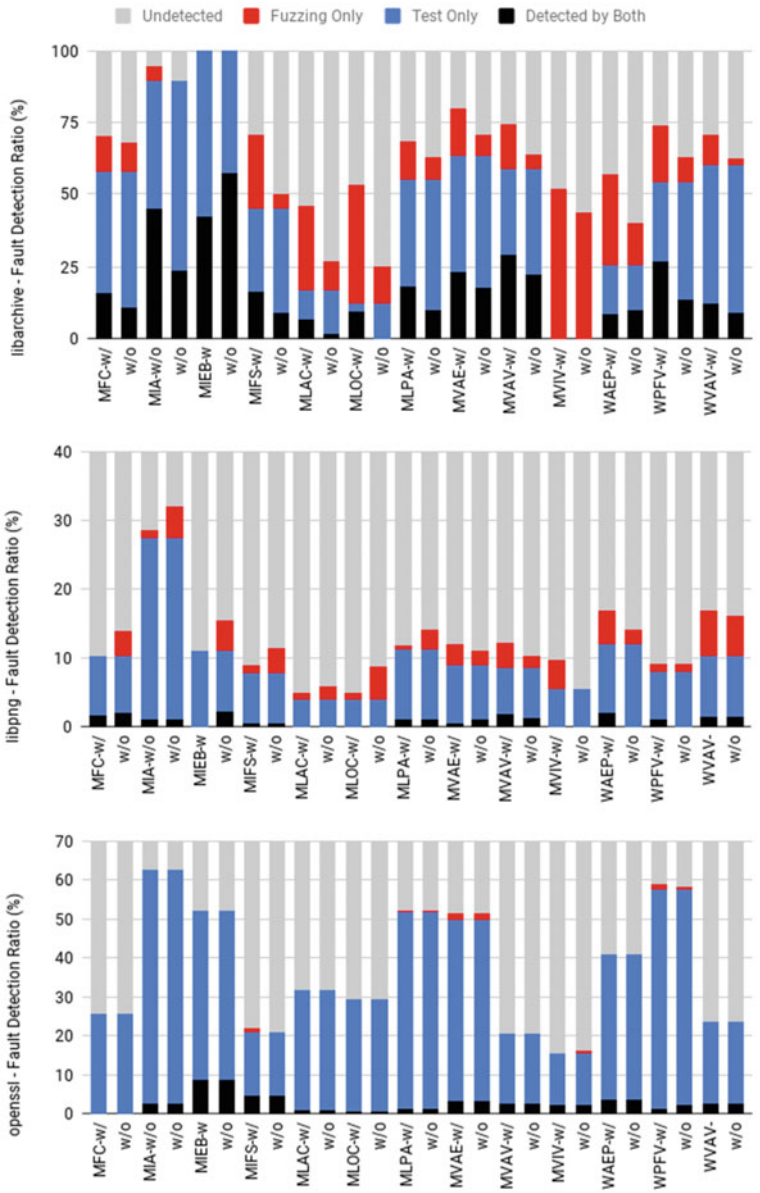


Fig. 5 Fault detection ratios by fuzzing and testing in *libpng* and *openssl* with (w/) and without (w/o) using the seeds and dictionaries

In *libarchive*, in total 17.3% of the software faults was detected by both fuzzing and testing, 30.9% was detected by testing only, and 20.1% was detected by fuzzing only (using the seeds). It shows that fuzzing and testing target different software faults (e.g., in terms of fault location and type), showing the importance of using both for software security and quality. For example, the tests provided 100% coverage for the injected MIEB software faults, while only the fuzzing detected some of the injected MVIV software faults. Although the exact ratios depend on the used software fault samples (see the fault counts in Table 2), this experiment evidences a practical case where the human-written tests and automated gray box fuzzing naturally targeted different kinds of software faults and become a complementary technique to each other.

The average detection ratios of fuzzing were similar in *libpng* and *openssl* (i.e., 3.2% for *libpng* and 2.9% for *openssl* with the seeds and dictionaries). When the seeds and dictionaries were not used, those two programs still showed the similar fault detection ratios, though sometimes slightly lower or slightly higher than the other program. On the other hand, testing *openssl* led to the higher fault detection ratio (i.e., 37.1% on average) than testing *libpng* (i.e., 10% on average). One may argue that *openssl* may have a high test coverage because its tests might be written rigorously or because it may have a low implementation complexity (e.g., in terms of cyclomatic complexity not the total SLOC) that can make the testing effective. It was found that the latter is not the case. Specifically, in the experiment, fuzzing *openssl* did not provide a significant fault detection coverage gain (i.e., 0.27% gain on average across all fault types and at most 1.7% gain for MVAE fault type) over the testing. That means that the program complexity of *openssl* was not sufficiently low for the fuzzing to provide a notable fault detection coverage gain over the testing. In fact, the *openssl* tests were relatively well-written and consisted of at least 1567 lines of C/C++ code, and 1980 lines of perl script code.

In *libpng*, without the seeds and dictionaries, the fuzzing provided the average fault detection coverage gain of 3% over the testing. That 3% gain is 10 times higher than the 0.3% fuzzing coverage gain seen in *openssl*. It implies that the fuzzing used in *libpng* is thus more effective than the one used in *openssl*. Moreover, we note that 3% average coverage gain is equivalent to 78% of the total software faults (3.9%) found by the fuzzing. That means the tests were, on the other hand, not highly effective because the tests were able to find only 22% of the software faults found by the fuzzing in *libpng*.

Implications. These experiments demonstrated how to use the presented methodology to quantitatively evaluate the effectiveness of fuzzing and conventional testing. The presented methodology can be used to not only show what kind of validation method (e.g., fuzzing or testing) is more effective for a given native program but also to guide users to prioritize one validation method over the other. For example, in the case of *openssl*, a user can decide to run the testing more frequently, while running the fuzzing only occasionally by considering the relatively high cost of fuzzing. Conversely, a user can decide to run the fuzzing more frequently in case of *libpng* because the fuzzing in *libpng* provided a notable extra fault detection coverage gain. For *libpng*, a user can also decide to write more test cases by considering the fact

that large part of software faults was only detected by the fuzzing. That is because those software faults detected by the fuzzing are usually exploitable in the field (e.g., buffer overflows), while testing is a cost-effective technique in terms of the time and computing resources required.

6.2 Coverages of Error Detectors

The presented methodology was used to evaluate the error detection mechanisms embedded in the benchmark programs. To this end, the fuzzing failures are classified into the nine types by using the dynamic analyzers of the fuzzing framework and the built-in error detectors of the target benchmark programs. The nine fuzzing failure types are:

1. *Assert*, when an error is caught by a built-in error detector (e.g., assertion) of a target program
2. *OOM (Out-of-Memory)*, when a target program and the fuzzer library combined uses the memory space more than a specified threshold (i.e., 2 GB per process in the experiment)
3. *LSan (LeakSanitizer)*, when a memory leak is detected
4. *ASan (AddressSanitizer): Memcpy Param Error*, when a *memcpy*() parameter value corruption is detected
5. *ASan: Free non-allocated*, when there is an attempt to free a non-allocated memory address
6. *ASan: GBO (Global Buffer Overflow)*, when a global variable has a buffer overflow
7. *ASan: SBO (Stack BO)*, when there is a stack buffer overflow
8. *ASan HBO (Heap BO)*, when there is a heap buffer overflow
9. *ASan: SEGV*, when there is a memory segmentation fault

Variations Between Target Programs. Figure 6 (top) shows the breakdown of the observed fuzzing failure types as a function of the benchmark program. The breakdown had significant variations between the programs. For example, in *openssl* and *sqlite*, 34% and 36.8%, respectively, of the injected faults was detected by the built-in error detectors (in other words assertions, see Assert part) without the seeds and dictionaries. However, the other two programs did not have strong assertions that were measurable in the experiment. Only *libpng* detected 1.5% of the faults by using its built-in error detectors. That means that one can add more security error detection techniques to those two programs and increase the coverages. We note that the bar graph for the label *sqlite* (n.a.) is for when *sqlite* was compiled without its assertions and the random fault sampling (total fault count: 1286) was used for the examination. The last bar graph shows the importance of keeping assertions in production code for better error detection.

In production, all or most of such dynamic analyzers are unavailable and only the built-in error detectors can be enabled and used. Thus, any bugs that are undetected by

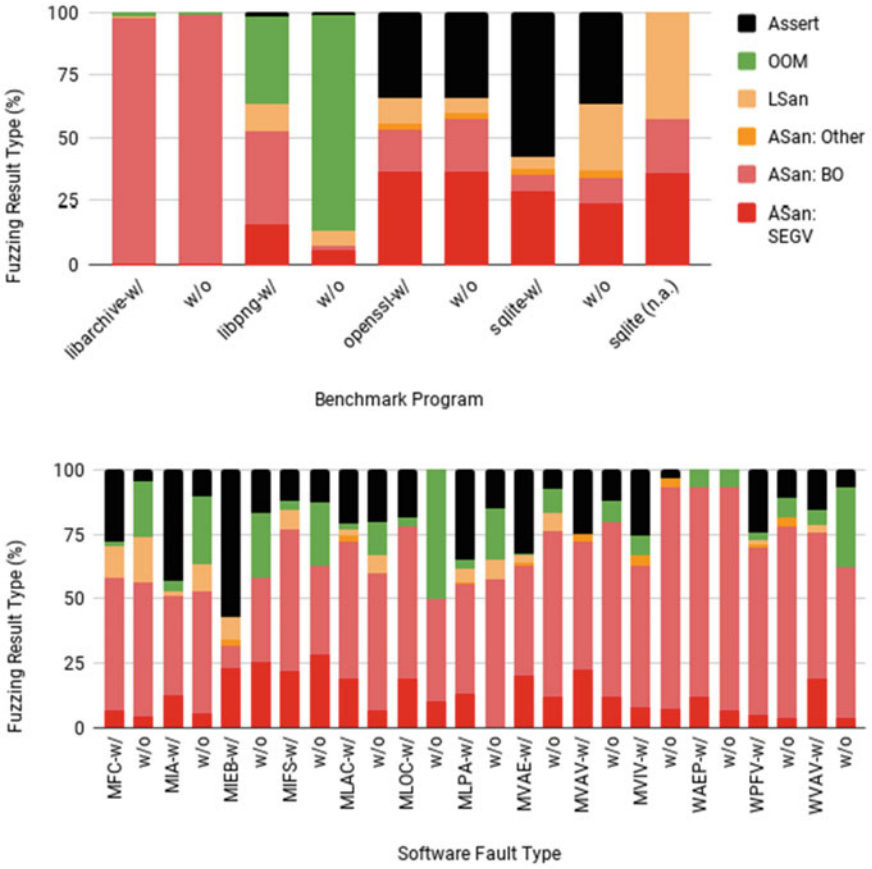


Fig. 6 Fuzzing failure type versus benchmark program or fault type (with and without the seeds and dictionaries) where n.a. means a version compiled with no built-in error detectors

the built-in error detectors but detected by the dynamic analyzers can be considered as realistic vulnerabilities. We note that it is also possible that some built-in detectors can detect such errors later if the sanitizers were not enabled. However, the chance of such cases is typically low (i.e., considering the well-known error latency distribution data [55]), and an attacker can successfully exploit such vulnerabilities and change the program control flow in order to not execute the built-in error detectors.

An injected software fault caught by a sanitizer (e.g., ASan and LSan failure types) typically reveals part of the error propagation path of at least one potential security attack. At a minimum, it emulates a key milestone or symptom of a potential security attack (e.g., a buffer overflow or memory corruption). While the real locations of any future vulnerabilities would be different from where the generated software faults were injected into, it is possible that when a software defect is activated in some specific ways, it can exercise part of the error propagation paths revealed

in experiments using the presented methodology. Repeating the experiments for many software fault samples thus increases the chances of emulating many realistic vulnerabilities. Thus, there is a value in finding software faults that cause such fuzzing failures and then hardening the target programs so as to preemptively check the identified error propagation paths.

Variations Between Fault Types. Figure 6 (bottom) shows the fuzzing failure type distribution as a function of the fault type. In general, using the seeds and dictionaries increased the Assert fuzzing failure type ratios much more than not using them. That holds in all fault types, except for the MIFS fault type where the ratio dropped negligibly from 12.5 to 12.3%. Another observation is that there was no prominent patterns other than the fact that most of the fault types can emulate and discover various failure scenarios and types.

6.3 Detection Latency of Injected Faults

This experiment measures the *fuzzing failure detection latency* that is defined in this study as the number of input data sets tested as part of the fuzzing before it found a fuzzing failure. Figure 7 shows the latencies as a function of the fault type (upper figure) and the fuzzing failure type (lower figure). The data was collected when the seeds and dictionaries are unused. Specific failure types took longer (e.g., all the failures in ASan: SBO, ASan: GBO, ASan: Free non-allocated, ASan: Memcpy Param, and Assert typically required 50k–100k input sets) than the rest. In general, using the seeds and dictionaries reduces the detection latency because the searching becomes more effective than without using them.

Similarly, the upper figure identifies specific fault types that were generally harder to discover during the fuzzing. For example, the MLOC fault type took longer to discover than the MLAC fault type because missing an OR clause is typically less severe than missing an AND clause (in terms of the probability of changing the execution behavior of a program). Without the seeds and dictionaries, the WVAV fault type probabilistically took shorter to discover than the MVAV and MVIV fault types because wrong value assignments generally have a more severe impact than missing value assignments or initializations.

Such knowledge can be used for various purposes. For example, it can be used to estimate the difficulty of finding a specific type of software bugs (e.g., leading to a specific failure type). By comparing the measured latency distribution and the reference latency distribution of the same kinds of software faults, one can decide to conduct more extensive fuzzing studies in order to find much deeper software defects than what were previously found in the field or during testing and fuzzing until the measured distribution becomes statistically equivalent to the reference distribution.

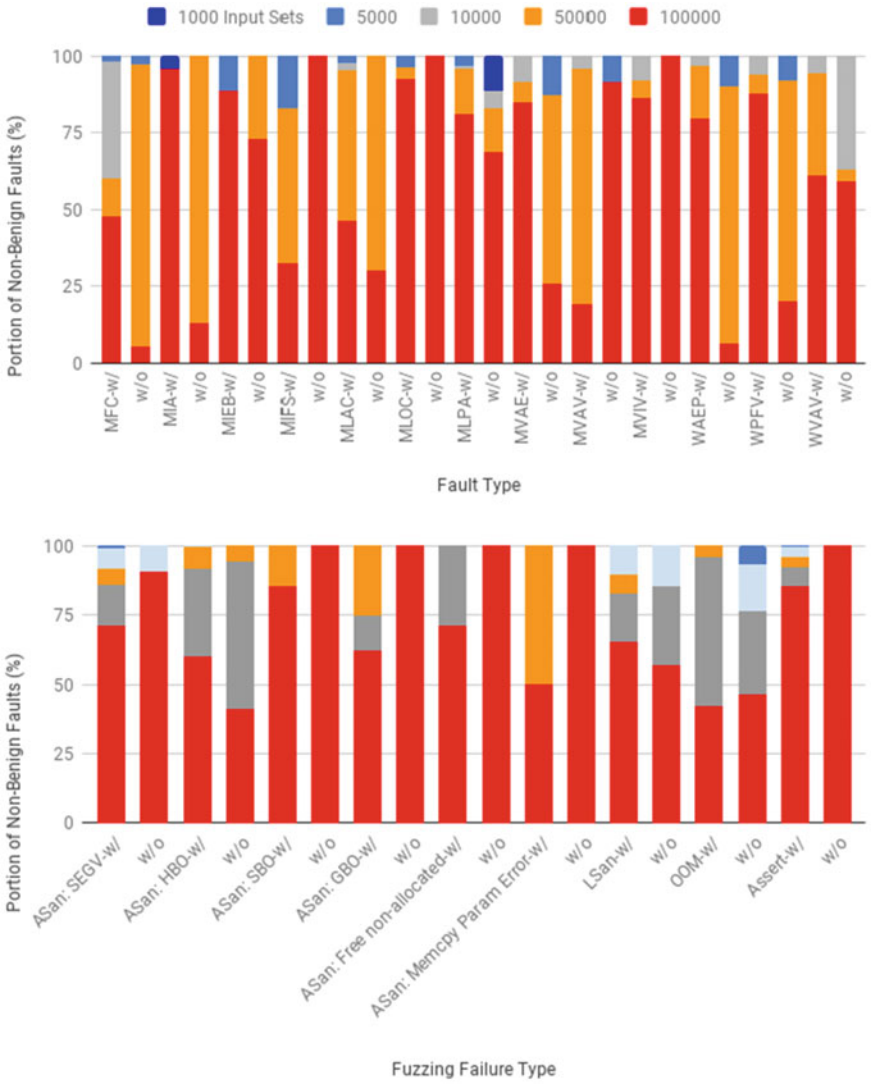


Fig. 7 Portion of injected, non-benign faults detected after exercising a specific number of input data sets (legend)

6.4 Evaluation of Fault Selection Algorithm

This experiment evaluates the efficiency of the presented fault selection algorithm. For this experiment, *sqlite* was used that had on average only 14.8% (or ~1.6%) fault detection ratio with (or without) using the seeds and dictionaries by its fuzzing when the random fault sampling technique was used (in Sects. 5 and 6.1). Table 5 shows the

Table 5 Injected faults caught by fuzzing when the presented fault selection is used with (w/) and without (w/o) using the seeds and dictionaries (for *sqlite*)

Fault type	Sample Count		Detected Fault ratio	
	(w/)	(w/o)	(w/)	(%)
MFC	46	47	32.6 ± 13.5	23.4 ± 12.1
MIA	23	23	43.5 ± 20.3	30.4 ± 18.8
MIEB	14	14	21.4 ± 21.5	7.1 ± 13.5
MIFS	33	34	27.3 ± 15.2	17.6 ± 12.8
MLAC	12	12	58.3 ± 27.9	41.7 ± 27.9
MLOC	12	12	16.7 ± 21.1	16.7 ± 21.1
MLPA	116	116	52.6 ± 9.1	42.2 ± 9.0
MVAE	29	29	41.4 ± 17.9	20.7 ± 14.7
MVAV	19	19	94.7 ± 10.0	10.5 ± 13.8
MVIV	5	5	40.0 ± 42.9	40.0 ± 42.9
WAEP	8	8	37.5 ± 33.5	37.5 ± 33.5
WPFV	35	35	54.3 ± 16.5	28.6 ± 15.0
WVAV	20	21	50.0 ± 21.9	33.4 ± 20.2
Total	372	375	46.0 ± 5.1	29.6 ± 5.1

result with the 95% confidence intervals. On average, 46% (or 29.6%) of the software faults selected by the presented locality-based algorithm led to fuzzing failures with (or without) using the seeds and dictionaries, which is ~3.1–18.7 times higher than the random fault sampling result. Such high fault activation or manifestation ratios are highly effective in reducing the time required to identify critical error propagation paths and security hardening targets because the cost of fuzzing is generally high.

7 Discussion

This section discusses the implications for software security hardening. The error propagation paths identified as part of the fore described experiments can be used to guide the security hardening process of a target native program. As a case study, let us select the fuzzing failures that were uncaught by the built-in error detectors, and analyze their error propagation paths. All those examples can be classified into the following four common cases. Figure 8 exemplifies the four cases where a software fault is injected into *Function* 2. Failures can be seen when the program executes *Function* 1, 2, 3, or 4 where *Function* 1 directly calls *Function* 2, which then calls *Function* 3.

Postcondition Checks for Libraries. If a software fault is injected into *Function* 2, its caller, *Function* 1, can experience a failure. This is typically due to a corrupted

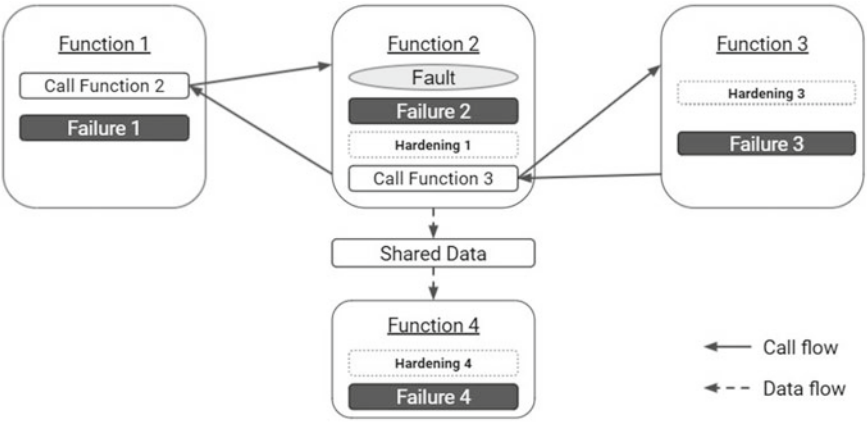


Fig. 8 The four identified cases of secure software hardening

return value of *Function* 2 whose return values include call-by-pointer or call-by-reference arguments of *Function* 2. In that case, adding postcondition checks to *Function* 2 (see Hardening 1 in Fig. 8) can detect the induced error and gracefully handle the error (e.g., fail-stop). Without placing such postcondition checks in *Function* 2, the software fault can cause a memory corruption or other equivalent errors which can become a strong attack vector if the attacker can control specific properties associated with the error (e.g., its location and occurrence time).

Backtracking Control and Data Flows. In Fig. 8, the injected fault can cause a failure at *Function* 2. While it is possible to add assertions to ensure the integrity of data around the corrupted state, it would be also debatable whether adding such assertions makes sense in practice. Typical such discussions would consider various factors such as the considered software fault or attack models and the required level of security in their target programs. *(a) Attack model.* The original intent of injecting software faults was to capture part of the critical error propagation paths. Although the triggering part is undecided, we note that some attack models can directly emulate the impact of an injected fault (e.g., as a register or memory value corruption at a right moment). For example, rowhammer attacks [5, 56] can corrupt a critical state of an object instance of a target software system although that relies on rowhammer vulnerabilities in the underlying kernel or middleware, and thus cannot easily target an arbitrary code location of a native program even when the hardware has such defects.

One of the practical security attack approaches is to find the backward dataflow from the corrupted state to an input value of a target program. There are many techniques (e.g., based on constraint solver generating concrete input values for a given call flow path [57]) that can assist such backward dataflow tracking process. Using the states corrupted as a result of the activation of an identified software fault (e.g., a fault leading to a buffer overflow without being detected by the tests and built-in error detectors) can reduce the search space of such automatic test input generation

techniques (e.g., [58]) and consequently improve their scalability when the techniques are practically used. If one can successfully identify such backward dataflow to an input value, the recommendation is to add sanity checks for the identified input value or to add checks to other derived values used before the input value propagates towards the identified corrupted state within the target program.

Precondition Checks for Libraries. Another case is when the injected software fault causes a failure at *Function* 3 (see Fig. 8). This is typically due to passing of a corrupted argument value from *Function* 2 to *Function* 3 and thus can be detected by adding some precondition checks to *Function* 3.

An example in the experiment was WAEP fault #18 of *openssl*. The following code fragment shows that fault which changes the line −8 to +8 to corrupt a pointer argument value. When *do_ssl3_write*() called *memcpy*() using as alias of the corrupted pointer, *memcpy*() caused a stack buffer overflow that was caught by the address sanitizer.

```
 1: int ssl3_write_bytes(SSL *s, ...,
 2:     const void *buf_, ...) {
 3:   const unsigned char *buf = buf_;
 4:   ...
 5:   for (;;) {
 6:     ...
 7:     i = do_ssl3_write(s, type,
-8:         &(buf[tot]), nw, 0);
+8:         &(buf), nw, 0);
```

This can be detected by adding precondition checks to *do_ssl3_write*() so as to ensure that all buffers passed to it are pointing to valid memory objects. Many similar cases were observed where classifying the allocated memory objects by their type and keeping a list of allocated objects of each type help build such sanity checks in an effective way.

```
 1: void ll_append_head(CIPHER_ORDER**
 2:     head, CIPHER_ORDER* curr,
 3:     CIPHER_ORDER** tail) {
 4:   ...
-5:   if (curr->prev != NULL)
-6:     curr->prev->next = curr->next;
 7:   (*head)->prev = curr;
 8:   ...
```

Sanity Checks for In-Memory Data Structure. In *openssl*, MIFS fault #18 injected into *ssl/ssl_ciph.c* file removed the *if*-statement at line 5 and 6. Later, when *ssl_cipher_collect_aliases*() called by *ssl_create_cipher_list*() was run, it

caused a heap buffer overflow failure that was detected by the address sanitizer. We note that *ssl_create_cipher_list*() can call *ssl_cipher_apply_rule*(), which can then call *ll_append_head*(). However, there is no legitimate, direct call flow between where the fault is injected, *ll_append_head*(), and where the failure is detected, *ssl_cipher_collect_aliases*(). Thus the error propagated via the doubly- linked list data structure. Clearly, adding a sanity check for that data structure can detect such errors in advance before the error propagates to manifest as a heap buffer overflow failure.

Similar to the used *sqlite* benchmark program, in other systems (such as file system and database) fail-stop or fail-fast property is critical for their operational missions. Part of the critical error propagation paths identified as a result of using the presented methodology can be thus used to assess such fail-stop property of a target program under various kinds of software faults [45] and to identify hardening targets. In that case, hardening gains more practical sense than the other use cases because the product owners want to continue running the applications even if there is a detected error. Thus, the applicability of the error propagation analysis explained in this case study depends also on the requirements of a target program [51].

8 Conclusion

Common security assessment approaches tried to measure or find as many vulnerabilities as possible from a target program. Such approaches are costly and their efficiency depends on the amount of effort put into finding critical vulnerabilities. As the frequency of updating software becomes higher and higher, such previous assessment approaches become less and less practical and can mainly be kept as a retrospective measurement technique. In order to overcome such technical hurdle, this study presented a novel methodology that can proactively assess the security defense of native programs by extracting the relevant conditional probabilities. The measured conditional probabilities show: (1) the capability of fuzzing in finding exploitable software faults in the injected classical software faults and (2) the capabilities of built-in security protection mechanisms and tests of a target program against the exploitable software faults. This, for example, helps us understand what portion of non-benign software faults can be caught by the built-in protection mechanisms of a target program and the error detection mechanisms of the underlying software and hardware platforms. By combining the derived conditional probabilities with actual measurement data (e.g., the number of recently found and fixed vulnerabilities), one can further estimate how many non-benign software faults are still hidden in a target program.[8]

[8] This work is rooted in the fault injection methodology and demonstrates a new application area of fault injection in software security evaluation. Since the author joined Google, there have been other works done to improve the dependability and security of mobile cloud computing applications.

References

1. Chen S, Xu J, Kalbarczyk Z, Iyer R, Whisnant K (2004) Modeling and evaluating the security threats of transient errors in firewall software. Perform Eval 56(1):53–72
2. Nakka N, Kalbarczyk Z, Iyer R, Xu J (2004) An architectural framework for providing reliability and security support. In: Proceedings of the IEEE/IFIP international conference on dependable systems and networks (DSN), pp 585–594
3. Pham C, Estrada Z, Cao P, Kalbarczyk Z, Iyer RK (2014) Reliability and security monitoring of virtual machines using hardware architectural invariants. In: Proceedings of the IEEE/IFIP international conference on dependable systems and networks (DSN), pp 13–24
4. Sanders WH (2014) Quantitative security metrics: Unattainable holy grail or a vital breakthrough within our reach? IEEE Secur Priv 12(2):67–69
5. Yim KS (2016) The rowhammer attack injection methodology. In: 2016 IEEE 35th symposium on reliable distributed systems (SRDS), pp 1–10
6. Iyer R, Nakka N, Gu W, Kalbarczyk Z (2010) Fault injection. In: Encyclopedia of software engineering, pp 287–299
7. Pendleton M, Garcia-Lebron R, Cho J-H, Xu S (2017) A survey on systems security metrics. ACM Comput Surv 49(4):1–35
8. Arlat J, Aguera M, Amat L, Crouzet Y, Fabre JC, Laprie JC, Martins E, Powell D (1990) Fault injection for dependability validation: a methodology and some applications. IEEE Trans Softw Eng 16(2):166–182
9. Madeira H, Silva JG (1994) Experimental evaluation of the fail-silent behavior in computers without error masking. In: Proceedings of IEEE 24th international symposium on fault-tolerant computing, pp 350–359
10. Madeira H, Rela M, Moreira F, Silva JG (1994) Rifle: a general purpose pin-level fault injector. In: Echtle K, Hammer D, Powell D (eds) Dependable computing—EDCC-1. Springer, Berlin, pp 197–216d
11. Karlsson J, Liden P, Dahlgren P, Johansson R, Gunneflo U (1994) Using heavy-ion radiation to validate fault-handling mechanisms. IEEE Micro 14(1):8–23
12. Ando H, Kan R, Tosaka Y, Takahisa K, Hatanaka K (2008) Validation of hardware error recovery mechanisms for the sparc64 v microprocessor. In: 2008 IEEE international conference on dependable systems and networks with FTCS and DCC (DSN), pp 62–69
13. Govindavajhala S, Appel AW (2003) Using memory errors to attack a virtual machine. In: 2003 symposium on security and privacy, pp 154–165
14. Miremadi G, Harlsson J, Gunneflo U, Torin J (1992) Two software techniques for on-line error detection. In: [1992] digest of papers. FTCS-22: the twenty-second international symposium on fault-tolerant computing, pp 328–335
15. Choi GS, Iyer RK (1992) Focus: an experimental environment for fault sensitivity analysis. IEEE Trans Comput 41(12):1515–1526
16. Jenn E, Arlat J, Rimen M, Ohlsson J, Karlsson J (1994) Fault injection into vhdl models: the mefisto tool. In: Proceedings of IEEE 24th international symposium on fault-tolerant computing, pp 66–75
17. Segall Z, Vrsalovic D, Siewiorek D, Yaskin D, Kownacki J, Barton J, Dancey R, Robinson A, Lin T (1988) Fiat-fault injection based automated testing environment. In: [1988] The eighteenth international symposium on fault-tolerant computing. Digest of papers, pp 102–107
18. Yim KS, Pham C, Saleheen M, Kalbarczyk Z, Iyer R (2011) Hauberk: Lightweight silent data corruption error detector for gpgpu. In: Proceedings of the IEEE international parallel distributed processing symposium (IPDPS), pp 287–300
19. Stott DT, Floering B, Burke D, Kalbarczpk Z, Iyer RK (2000) Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors. In: Proceedings IEEE international computer performance and dependability symposium. IPDS 2000, pp 91–100

Interested readers are referred to [59] for big data service monitoring, [60] for big data software release, [5] for cloud virtualization platform security, and [61] Android platform ecosystem security.

20. Carreira J, Madeira H, Silva JG (1998) Xception: a technique for the experimental evaluation of dependability in modern computers. IEEE Trans Softw Eng 24(2):125–136
21. Arlat J, Fabre JC, Rodriguez M (2002) Dependability of cots microkernel-based systems. IEEE Trans Comput 51(2):138–163
22. Kanawati GA, Kanawati NA, Abraham JA (1995) Ferrari: a flexible software-based fault and error injection system. IEEE Trans Comput 44(2):248–260
23. Han S, Shin KG, Rosenberg HA (1995) Doctor: an integrated software fault injection environment for distributed real-time systems. In: Proceedings of 1995 IEEE international computer performance and dependability symposium, pp 204–213
24. Pattabiraman K, Nakka NM, Kalbarczyk ZT, Iyer RK (2013) Symplfied: symbolic program-level fault injection and error detection framework. IEEE Trans Comput 62(11):2292–2307
25. Jacques-Silva G, Kalbarczyk Z, Gedik B, Andrade H, Wu K-L, Iyer RK (2011) Modeling stream processing applications for dependability evaluation. In: 2011 IEEE/IFIP 41st international conference on dependable systems networks (DSN), pp 430–441
26. Kao WI, Iyer RK, Tang D (1993) Fine: a fault injection and monitoring environment for tracing the Unix system behavior under faults. IEEE Trans Softw Eng 19(11):1105–1118
27. Kao W-L, Iyer RK (1994) Define: a distributed fault injection and monitoring environment. In: Proceedings of IEEE workshop on fault-tolerant parallel and distributed systems, pp 252–259
28. Tsai TK, Iyer RK, Jewitt D (1996) An approach towards benchmarking of fault-tolerant commercial systems. In: Proceedings of annual symposium on fault tolerant computing, pp 314–323
29. Chillarege R, Bhandari IS, Chaar JK, Halliday MJ, Moebus DS, Ray BK, Wong MY (1992) Orthogonal defect classification-a concept for in-process measurements. IEEE Trans Softw Eng 18(11):943–956
30. Sullivan M, Chillarege R (1991) Software defects and their impact on system availability-a study of field failures in operating systems. In: [1991] digest of papers. Fault-tolerant computing: the twenty-first international symposium, pp 2–9
31. Thakur A, Iyer RK, Young L, Lee I (1995) Analysis of failures in the tandem nonstop-ux operating system. In: Proceedings of the sixth international symposium on software reliability engineering, pp 40–50
32. Herder JN, Bos H, Gras B, Homburg P, Tanenbaum AS (2009) Fault isolation for device drivers. IEEE/IFIP international conference on dependable systems networks 2009:33–42
33. Herder JN, Bos H, Gras B, Homburg P, Tanenbaum AS (2007) Failure resilience for device drivers. In: Proceedings of the IEEE/IFIP international conference on dependable systems and networks (DSN), pp 41–50
34. Moraes R, Barbosa R, Durães J, Mendes N, Martins E, Madeira H (2006) Injection of faults at component interfaces and inside the component code: are they equivalent? In: Proceedings of the European dependable computing conference, pp 53–64
35. Johansson A, Suri N, Murphy B (2007) On the selection of error model(s) for os robustness evaluation. In: 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07), pp 502–511
36. Winter S, Sarbu C, Suri N, Murphy B (2011) The impact of fault models on software robustness evaluations. In: 2011 33rd international conference on software engineering (ICSE), pp 51–60
37. Durães J, Madeira H (2002) Emulation of software faults by educated mutations at machine-code level. In: Proceedings of 13th international symposium on software reliability engineering, pp 329–340
38. Durães JA, Madeira HS (2006) Emulation of software faults: a field data study and a practical approach. IEEE Trans Softw Eng 32(11):849–867
39. Durães J, Madeira H (2003) Definition of software fault emulation operators: a field data study. In: Proceedings of 2003 international conference on dependable systems and networks, pp 105–114
40. Natella R, Cotroneo D, Duraes JA, Madeira HS (2013) On fault representativeness of software fault injection. IEEE Trans Softw Eng 39(1):80–96

41. Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. IEEE Trans Softw Eng 37(5):649–678
42. Hudak JJ, Suh BH, Siewiorek DP, Segall Z (1993) Evaluation and comparison of fault-tolerant software techniques. IEEE Trans Reliab 42(2):190–204
43. Bondavalli A, Chiaradonna S, Cotroneo D, Romano L (2004) Effective fault treatment for improving the dependability of cots and legacy-based applications. IEEE Trans Dependable Secure Comput 1(4):223–237
44. Ng WT, Chen PM (2001) The design and verification of the rio file cache. IEEE Trans Comput 50(4):322–337
45. Chandra S, Chen PM (1998) How fail-stop are faulty programs? In: Digest of papers. Twenty-eighth annual international symposium on fault-tolerant computing (Cat. No.98CB36224), pp 240–249
46. Swift MM, Bershad BN, Levy HM (2003) Improving the reliability of commodity operating systems. In: Proceedings of the nineteenth ACM symposium on operating systems principles, ser. SOSP '03. ACM, New York, pp 207–222
47. Swift MM, Annamalai M, Bershad BN, Levy HM (2006) Recovering device drivers. ACM Trans Comput Syst 24(4):333–360
48. Neves N, Antunes J, Correia M, Verissimo P, Neves R (2006) Using attack injection to discover new vulnerabilities. In: International conference on dependable systems and networks (DSN'06), pp 457–466
49. Antunes J, Neves N, Correia M, Verissimo P, Neves R (2010) Vulnerability discovery with attack injection. IEEE Trans Softw Eng 36(3):357–370
50. Fonseca J, Vieira M, Madeira H (2014) Evaluation of web security mechanisms using vulnerability attack injection. IEEE Trans Dependable Secure Comput 11(5):440–453
51. Kanoun K, Spainhower L (2008) Dependability benchmarking for computer systems. Wiley, IEEE Computer Society Pr
52. Tsai TK, Hsueh M-C, Zhao H, Kalbarczyk Z, Iyer RK (1999) Stress-based and path-based fault injection. IEEE Trans Comput 48(11):1183–1201
53. Serebryany K, Bruening D, Potapenko A, Vyukov D (2012) Addresssanitizer: a fast address sanity checker. In: 2012 USENIX annual technical conference (USENIX ATC 12). USENIX Association, Boston, pp 309–318. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/serebryany
54. Kim S, Zimmermann T, Jr EJW, Zeller A (2007) Predicting faults from cached history. In: Proceedings of the international conference on software engineering (ICSE), pp 489–498
55. Yim KS, Kalbarczyk ZT, Iyer RK (2009) Quantitative analysis of long-latency failures in system software. In: 2009 15th IEEE Pacific rim international symposium on dependable computing, pp 23–30
56. Razavi K, Gras B, Bosman E, Preneel B, Giuffrida C, Bos H (2016) Flip feng shui: Hammering a needle in the software stack. In: 25th USENIX security symposium (USENIX Security 16). USENIX Association, Austin, pp 1–18
57. Cadar C, Ganesh V, Pawlowski PM, Dill DL, Engler DR (2008) Exe: automatically generating inputs of death. ACM Trans Inf Syst Secur 12(2):10:1–10:38
58. Cadar C, Dunbar D, Engler D (2008) Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 8th USENIX conference on operating systems design and implementation, ser. OSDI'08. USENIX Association, Berkeley, pp 209–224
59. Yim KS (2016) Evaluation metrics of service-level reliability monitoring rules of a big data service. In: 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pp 376–387
60. Yim KS (2014) Norming to performing: Failure analysis and deployment automation of big data software developed by highly iterative models. In: 2014 IEEE 25th international symposium on software reliability engineering, pp 144–155
61. Yim KS, Malchev I, Hsieh A, Burke D (2019) Treble: fast software updates by creating an equilibrium in an active software ecosystem of globally distributed stakeholders. ACM Trans Embed Comput Syst 18(5s). [Online]. Available: https://doi.org/10.1145/3358237

Multi-layered Monitoring for Virtual Machines

Cuong Pham

Abstract This chapter describes monitoring methods to achieve both security and reliability in virtualized computer systems. We show how to perform continuous monitoring and leverage information across different layers of a virtualized computer system to detect malicious attacks and accidental failures.

1 Motivation

When a system is deployed at scale, the efficient automation of monitoring is key to achieving resilience against accidental failures and malicious attacks. This chapter specifically focuses on monitoring virtualized computer systems, which is an enabling technology of modern data centers.

Why monitoring? Computer systems fail regardless of how carefully they are constructed. A failure is either a reliability incident or a security incident. While reliability incidents are primarily caused by the increasing complexity of computer systems, security threats increase as data stored and processed by computers carry greater value.

It is a well-established design principle to treat reliability and security incidents as the norm, rather than the exception [1]. A system operates under the assumption that it can accidentally fail or be attacked at any point in time. Therefore, to produce steady and useful progress, the system needs to be monitored so that adverse incidents are detected and mitigated as quickly as possible. This is the principle that embraces high-fidelity monitoring as essential to achieve resiliency in computer systems.

Our research shares the same core proposition with this design principle: using monitoring as the main vehicle to cope with attacks and failures. We focus on the design and construction of efficient monitoring methods that can capture high-fidelity views of target systems.

C. Pham (✉)
2 Nguyen Van Tuong street, district 7, Ho Chi Minh city, Vietnam
e-mail: phammanhcuong@gmail.com

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_6

Why virtualized computer systems? Virtualization is the means to enable sharing and to achieve high utilization in modern data centers. In 2012, 51% of ×86 servers were virtualized, a 13% increase from 2011 [2]. In addition to virtualized servers being more prevalent than non-virtualized ones, the density of VMs on each server is also increasing [3].

The primary driving force of this trend is cloud computing, which leverages virtualization on commodity hardware as the core technology to facilitate sharing. Not unlike other types of utilities, cloud computing benefits from the economies of sharing and scaling. This is because sharing greatly decreases the cost of computing resources, which in turn attracts more users and providers to join the flow.

Given the abundance of VMs, an improvement in the security and reliability of this technology will have a large impact.

2 Target System Model

In this chapter, the target of monitoring is a virtualized system as depicted in Fig. 1. The bottom layers, including Hardware, Firmware/Bios, and Hypervisor/OS, constitute the host machine. The layers on the top, including Application and OS, constitute the virtual machines. The host machine can accommodate multiple VMs running at the same time. From user perspectives, VMs operate independently of each other.

We use the term VM monitoring to indicate any monitoring method that has the protection target (or target for short) in a layer of the virtualization software stack, including software running on a VM and the hypervisor. When the context is unclear,

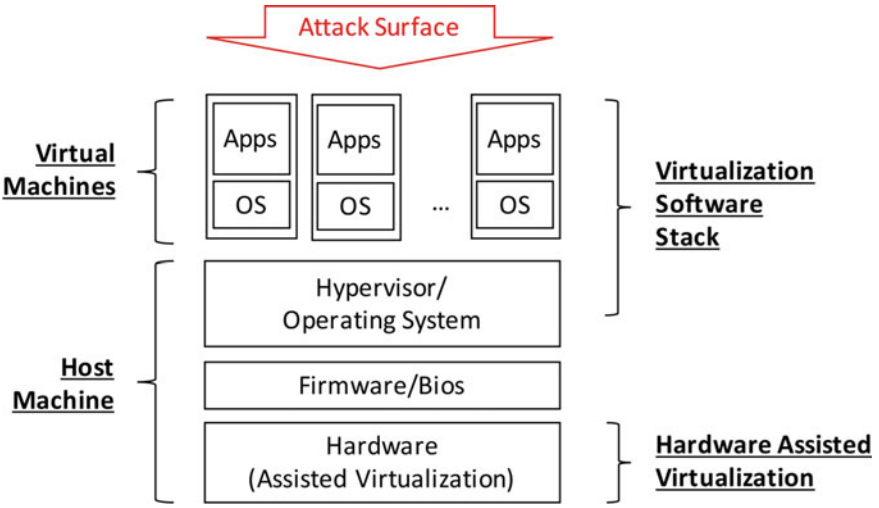


Fig. 1 A typical virtualized computer system. The virtualization software stack is the target of our monitoring

we use a more descriptive term to indicate the target of monitoring. Specifically, we use guest operating system monitoring, or guest application monitoring to indicate that monitoring targets are an operating system (OS) and applications running inside VMs, respectively. Similarly, we use hypervisor monitoring to indicate that the hypervisor is the target of monitoring. In addition, we use out-of-VM monitoring to indicate monitoring techniques deployed outside of target VMs to monitor software running inside VMs (e.g., monitoring is done from the hypervisor or from other VMs).

In the designs of our monitoring, we assume that Hardware Assisted Virtualization (HAV), such as Intel VT-x [4] or AMD-V [5]), is an integral component of the system, and is utilized by hypervisors to implement virtualization. At the moment, all server-grade $\times 86$ processors on the market support HAV. Furthermore, all popular hypervisor implementations, such as the VMWare hypervisor family, KVM [6], XEN [7], and Virtual Box, can utilize HAV to execute VMs.

With regard to security monitoring, our threat model assumes that VM share the attack surface of the target virtualized system. This assumption is derived from the model of data centers that rely on virtualization to serve users and process workloads. Infrastructure as a Service (IaaS) in cloud computing is a typical example of this model. In such a system, a user can execute arbitrary software, from user applications to their own OSs, inside VMs. Meanwhile, they do not have direct access to the host machine, except via the VM-hypervisor interface provided by HAV. Furthermore, we explicitly trust the underlying hardware. We also do not consider physical tampering and inside attackers (e.g., malicious administrators who already have remote access to the host machine).

In this threat model, we consider two broad scenarios: attacking a VM and attacking a hypervisor. The first scenario refers to attacks that aim at compromising software running inside a VM. Since in a typical data center setup most VMs must expose some remote access via the Internet to be used, they are constantly at risk of being targeted by attackers. The second scenario assumes the attacker has full access to a VM and exploits the VM-hypervisor interface to launch attacks against the underlying hypervisor (and other co-located VMs). For example, a public IaaS cloud allows any user to launch their own VMs at a very small cost. Those VMs can be used as an attack entry point to the hypervisor. Or a successful attack described in the previous scenario may grant the attacker administrative access to the exploited VM, which in turn can lead to an attack against the hypervisor.

3 Limitations of State-of-the-Art VM Monitoring

Despite significant research effort that has been invested, state-of-the-art VM monitoring techniques still experience some fundamental limitations that dwarf their practicality. Those are limitations that leave critical gaps for failures and attacks to escape detection. Here we present limitations in regard to security and reliability monitoring.

3.1 Polling-and-Scanning Monitoring Paradigm

Most VM monitoring techniques, e.g., [8–12], follow the polling-and-scanning paradigm. In this paradigm, monitoring is done by scanning the target system at a specific polling interval. This paradigm is also known as *passive monitoring* [13].

There are two major limitations of the polling-and-scanning method. First, it leaves vulnerable time gaps between consecutive polling intervals. During those temporal gaps, a transient attack, which completely removes its footprint after completing, cannot be detected. We have demonstrated in [14, 15] that transient attacks can be crafted to evade VM monitors with a high chance of success. Next, this monitoring method can only scan the static state of the target system, e.g., the state that is stored in SRAM or persistent storage. What it misses is operational data about the activities of the target system, which is necessary to enforce many security and reliability monitoring policies.

3.2 Untrustworthy Input

The goal of monitoring is to capture and present a trusted view of target systems. This view is used at a later phase in a system's operational pipeline, e.g., enforcing a security or reliability policy. Thus, the input of monitoring must be carefully selected to faithfully represent the target system. This requirement is particularly imperative in the context of security monitoring, because attackers always proactively seek opportunities like this, which let them manipulate input to falsify monitoring views.

However, many out-of-VM monitoring techniques [8–12] fail to satisfy this requirement, as they rely on untrustworthy input. These monitoring techniques exclusively rely on data structures maintained by software inside a target VM to derive views of the VM itself. It has been demonstrated that if the guest software is compromised, those data structures can be manipulated by attackers to circumvent such out-of-VM monitors [16, 17].

3.3 Inflexible Monitor Placement

Target systems and attacks are both moving targets. For example, the target system can be reconfigured or updated, or a new vulnerability or bug can be discovered. Many of these events require a corresponding update in the monitoring system. In addition, attacks are often carried out in multiple stages [18], with each stage requiring a different set of monitors to fully cover the trace of the attack.

For these reasons, monitoring systems need to be made ready for changes. Moreover, changes in a monitoring system should not be a source of downtime to target

systems. This is however not the case for existing VM monitoring techniques, which require monitoring setup and configuration as a part of the target system boot process.

3.4 Incompatible Reliability and Security Monitoring

Reliability and security tend to be treated separately because they appear orthogonal: reliability focuses on accidental failures, security on intentional attacks. Because of the apparent dissimilarity between the two, tools to detect and recover from the different classes of failures and attacks are usually designed and implemented differently. So, integrating support for reliability and security in a single framework is a significant challenge.

Current VM monitoring techniques are no exception. While there is a substantial body of VM monitoring research dedicated to security monitoring, and some work dedicated to reliability, we are not aware of any previous effort toward combining these two subjects of monitoring.

The above four identified issues in VM monitoring hinder its adoption in production systems. Our research aims at (i) raising the awareness of those issues via demonstrations of real attacks and failures, and (ii) exploring new monitoring paradigms and methods that can resolve all of the four issues.

4 HyperTap: Virtual Machine Monitoring Using Hardware Architectural Invariants

Reliability and security (RnS) are two essential aspects of modern highly connected computing systems. Traditionally, reliability and security tend to be treated separately because of their orthogonal nature: while reliability deals with accidental failures, security copes with intentional attacks against a system. As a result, mechanisms/algorithms addressing the two problems are designed independently, and it is difficult to integrate them under a common monitoring framework.

In this section, we identify the commonalities between reliability and security monitoring to guide the development of suitable frameworks for combining both uses of monitoring.

We apply our observations in the design and implementation of the HyperTap framework for virtualization environments.

4.1 Monitoring Principles

A monitoring process can be divided into two tightly coupled phases: logging and auditing [41]. In the logging phase, relevant system events (e.g., a system call) and state (e.g., system call parameters) are captured. In the auditing phase, these events and states are analyzed, based on a set of policies that classify the state of the system, e.g., normal or faulty. Based on that model, we observe that although Reliability and security monitors may apply different policies during the auditing phase, they can utilize the same event- and state-logging capability. This observation suggests that the logging phases of multiple reliability and security monitors need to be combined into a common framework. Unification of logging phases brings further benefits, namely, it avoids potential conflict between different monitors that track the same event or state, and reduces the overall performance overhead of monitoring.

4.1.1 Unified Logging

It is not uncommon for co-deployed logging mechanisms to conflict. For instance, two monitors relying on a certain counter that only allows exclusive access cannot use it simultaneously. A concrete example would be to deploy both the failure detection technique proposed in [43] and the malware detection technique proposed in [44] in the same system, as they both use hardware performance counters. In addition, one monitor may become a source of noise for other monitors. For example, intrusive logging could generate an excessive number of events.

The problem can be solved by unifying logging for co-located monitors. Unified logging is responsible for (i) retrieving common target system events and states, and then (ii) streaming them in a timely manner to customizable auditors, which enforce RnS policies.

Aside from avoiding potential conflicts, the combination of logging phases yields additional benefits. It can reduce the overall performance overhead of combined monitors. To ensure the consistency of captured states and events, logging is often a blocking operation. Once the event and state have been logged, an audit can be performed in parallel with execution of the target system. Therefore, combining blocking logging phases boosts performance, even in cases where the captured states differ. Furthermore, this approach inherits other benefits of the well-known divide-and-conquer strategy: it allows one to focus on hardening the core logging engine, and enables incremental development and deployment of auditing policies.

4.1.2 Achieving Isolation via Architectural Invariants

An OS invariant is a property defined and enforced by the design and implementation of a specific OS, so that the software stack above it, e.g., user programs and device drivers, can operate correctly. In the context of VMI, OS invariants allow the internal

state of a VM to be monitored from the outside by decoding the VM's memory [8–12]. No user inside a VM can interfere with the execution of outside monitoring tools. However, monitoring tools still share input, e.g., a VMs' memory, with the other software inside VMs. Therefore, those monitoring tools are vulnerable to attacks at the guest system level, as demonstrated in [16, 17, 45].

An architectural invariant is a property defined and enforced by the hardware architecture, so that the entire software stack, e.g., hypervisors, OSes, and user applications, can operate correctly. For example, the ×86 architecture requires that the CR3 and TR registers always point to the running process's Page Directory Base Address (PDBA) and Task State Segment (TSS), respectively. Hardware invariants and HAV features have been studied in the context of security monitoring [28] and offline malware analysis [33].

We find that architectural invariants, particularly the ones defined by HAV, provide an outside view with desirable features for VM reliability and security monitoring. The behaviors enforced by HAV involve primitive building blocks of essential OS operations, such as context switches, privilege level (or ring) transfers, and interrupt delivery. Furthermore, strong isolation between VMs and the physical hardware ensures the integrity of architectural invariants against attacks inside VMs. Software inside VMs cannot tamper with the hardware as it can with the OS. In this study, we explore the full potential of HAV for online enforcement of RnS policies.

However, relying solely on architectural invariants and ignoring OS invariants would widen the semantic gap separating the target VM and the hypervisor. The reason is that many OS concepts, such as user management (e.g., processes owned by different users), are not defined at the architectural level. In this study, we propose to use architectural invariants as the root of trust when deriving OS state. For example, the thread info data structure in the Linux kernel containing thread-level information can be derived from the TSS data structure, a data structure defined by the ×86 architecture.

In order to circumvent the OS state derivation, an attack would need to change the layout of OS-defined data structures (e.g., by adding fields to an existing structure that point to tainted data). Changing data structure layout, as opposed to changing values, is difficult for attackers, because (i) it involves significant changes to the kernel code that references the altered fields, and (ii) it would need to relocate all relevant kernel data objects. Not only are those attacks difficult to perform on-the-fly, but since malware always tries to minimize its footprint, our approach significantly impedes would-be attackers.

4.1.3 Robust Active Monitoring

Passive monitoring is suitable for persistent failures and attacks, because it assumes the corrupted or compromised state remains in the system sufficiently longer than the polling interval. That assumption does not hold in many RnS problems. For example, the majority of crash and hang failures in Linux systems have short failure latencies (the time for faults to manifest into failures) [46]. An unnecessarily long

detection latency, e.g., caused by polling monitoring, would result in subsequent failure propagation or inefficient recovery (e.g., multiple roll-backs).

As we demonstrate in Sect. 4.3.2, a transient attack can be combined with other techniques to create a stealthy attack that can defeat passive monitoring. Active monitoring, or event-driven monitoring, on the other hand, possesses many attractive features. Since it is event-driven, there is no time dependence that can be exploited. Furthermore, active monitoring can capture system activities in addition to the system state, which passive monitoring provides. System activities are the operations that transition a system from one state to another. Invoking a system call is an example of a system activity. In many cases, information about system activities is crucial to enforcing RnS policies.

Active monitoring is not foolproof, as it can suffer from event bypass attacks. If an attack can prevent or avoid generation of events that trigger logging, it can bypass the monitor. To make active monitoring robust, we propose to use hardware invariants, specifically the VM Exit feature provided by HAV, to generate events.

4.2 Framework and Implementation

4.2.1 Scope and Assumptions

HyperTap integrates with existing hypervisors to safeguard VMs against failures and attacks. It aims to make this protection transparent to VMs by utilizing existing hardware features. Thus, HyperTap does not require modification of either the existing hardware or the guest OS's software stack.

HyperTap's implementation assumes that the underlying hardware and hypervisor are trusted. Although extra validation and protection for the hardware and hypervisor could address concerns about the robustness of different hypervisors against failures and attacks, these issues are addressed by the proposed monitors in hShield (Sect. 6).

4.2.2 Monitoring Workflow

Figure 2 depicts the overall workflow of HyperTap. The left side of the figure illustrates how the shared event logging mechanism works and the right side describes the auditing phase.

HyperTap utilizes HAV to intercept the desired guest OS operations through VM Exit events generated by corresponding hardware operations. Since the HAV VM Exit mechanism is not designed to intercept all desired operations, e.g., system calls,

HyperTap supports a wide range of events, from coarse-grained events, such as process context switches, to finer-grained events, such as system calls, and very fine-grained events, such as instruction execution and memory accesses. That variable granularity ensures that HyperTap can be adopted for a broad range of RnS policies.

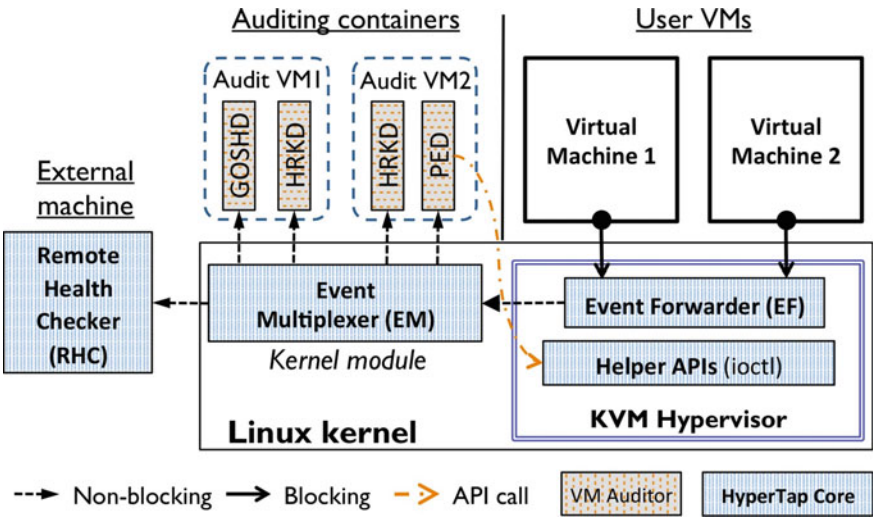


Fig. 2 Implementation of HyperTap in the KVM hypervisor. The hypervisor is modified to forward VM Exit events to the Event Multiplexer (EM), which is implemented as a separate kernel module. The EM forwards events to registered auditors running as user processes inside auditing containers. The Remote Health Checker (RHC) monitors the hypervisor's liveness

HyperTap delivers captured events to registered auditors, which implement specific monitoring policies. An auditor starts by registering for a set of events needed to enforce its policy. Upon the arrival of each event, the auditor analyzes the state information associated with the event. Auditors are associated with VMs and each VM can have multiple auditors.

HyperTap also provides an interface that allows auditors to control target VMs. For example, the auditing phase is non-blocking by default, but an auditor may pause its target VM during analysis in order to stop the VM during an attack, or roll-back the VM when it detects a non-recoverable failure.

4.2.3 Implementation

This subsection presents the integration of HyperTap with KVM [6], hypervisor built with HAV as a Linux kernel module. Figure 2 depicts the deployment of HyperTap's components.

HyperTap's unified logging channel is implemented through two components: an Event Forwarder (EF) and an Event Multiplexer (EM). The EF is integrated into the KVM module, and forwards VM Exit events and relevant guest hardware state to the EM. By default, events are sent non-blocking to minimize overhead. The EM, which is implemented as another Linux kernel module in the host OS, buffers input events from the EF and delivers them to the appropriate auditors.

The EM is also responsible for sampling VM Exit events that are sent to a Remote Health Checker (RHC) running in a separate machine. The RHC server acts as a heartbeat server to measure the intervals between received events. If no events are received after a certain amount of time, it raises an alert about the liveness of the monitoring system.

Auditors are implemented as user processes inside auditing containers 4 running on the host OS. Compared to the dedicated auditing VM used in previous work [11, 11], this approach offers multiple benefits. First, it provides lightweight attack and failure isolation among different VMs' auditors, and between auditors and the host OS. Second, it simplifies implementation and reduces the performance overhead of event delivery from the EM module. Finally, it allows the integration of auditors into existing systems, since containers are robust and compatible with most current Linux distributions.

We needed to add less than 100 lines of code to KVM to implement the EF component and export Helper APIs.

4.3 Performance Evaluation

We conducted experiments to measure the performance overhead of individual HyperTap auditors as well as the combined overhead of running multiple auditors. We measured the runtime of the UnixBench 3 performance benchmark when (i) each auditor was enabled, and (ii) all three auditors were enabled. The target VM was a SUSE 11 Linux VM with 2 vCPUs and 1GiB of RAM. The host computer ran SUSE 11 Linux and the KVM hypervisor, with an 8 core Intel i5 3.07 GHz processor and 8 GiB of RAM.

The results were illustrated in Fig. 3. The baseline is the execution time when running the workloads in the VM without HyperTap integrated, and the reported numbers are the average of five runs of the workloads.

In most cases, the performance overhead of running all three auditors simultaneously was (i) only slightly higher than that of running the slowest auditor, HT-Ninja, individually, and (ii) substantially lower than the summation of the individual overheads of all auditors. That result demonstrates the benefits of HyperTap's unified logging mechanism.

For the Disk I/O and CPU intensive workloads, all three auditors together produced less than 5% and 2% performance losses, respectively. The Disk I/O intensive workloads appear to have incurred more overhead than CPU intensive workloads because they generated more VM Exit events, at which point some monitoring code was triggered.

For the context switching and system call micro-benchmarks, all three auditors together induced about 10% (or less) and 19% performance losses, respectively. It is important to note that those micro-benchmarks were designed to measure the performance of individual specific operations without any useful processing; they do not necessarily represent the performance overhead of general applications. The

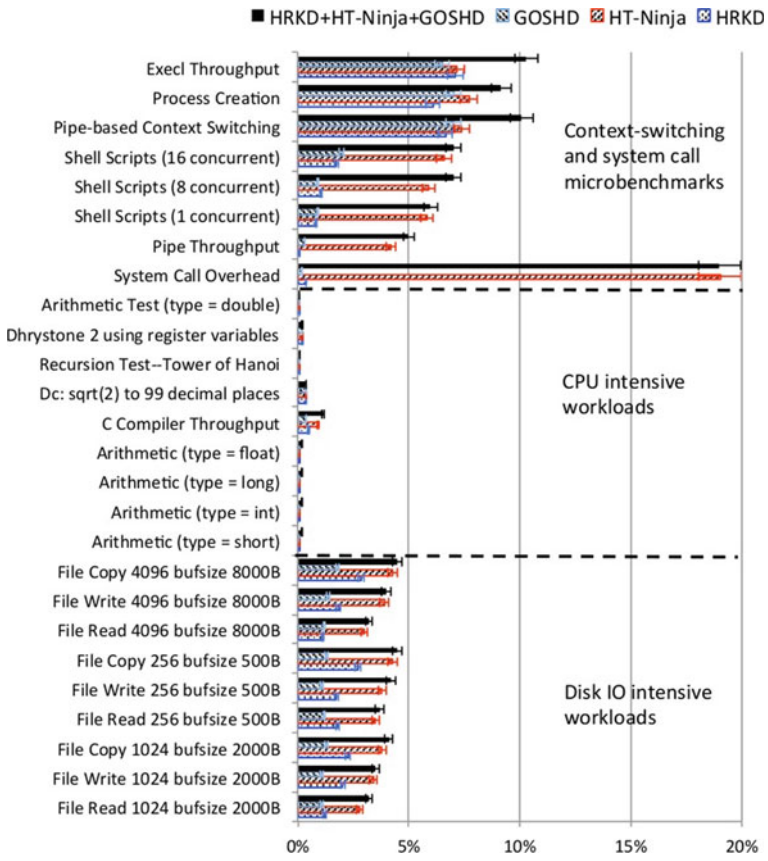


Fig. 3 Measured performance overhead of HyperTap sample monitors. The workloads are run with three different configurations: (1) both HRKD and HT-Ninja, (2) only HT-Ninja, and (3) only HRKD. Error bars indicate one standard deviation

relatively high overhead was caused by the HyperTap routines enabled for logging those benchmarked operations. Since only HT-Ninja needs to log system calls, it was the primary source of the overhead in the system call micro-benchmark case.

5 Hprobes: Dynamic Virtual Machine Monitoring Using Hypervisor Probes

This section introduces Hprobe, a framework that allows one to dynamically monitor applications and operating systems inside a VM. The Hprobe framework does not require any changes to the guest OS, which avoids the tight coupling of monitoring with its target.

Furthermore, the monitors can be customized and enabled/disabled while the VM is running.

5.1 Introduction

The HyperTap framework introduced in the previous chapters provides an efficient and hard-to-bypass event-driven monitoring mechanism. The key design of HyperTap is the reliance on a fixed set of hardware architectural invariants to capture guest OS's activities. While we have shown that this monitoring capability is effective to support an important set of reliability and security monitoring policies (see Chap. 4 for examples of the evaluated policies), there are still many cases in which monitors requires a more flexible means to place monitoring points, or hooks, to capture specific guest OS and applications' operational activities.

One class of active monitoring systems is a hook based system, where the monitor places hooks inside the target application or OS [13]. A hook is a mechanism used to generate an event when the target executes a particular instruction. When the target's execution reaches the hook, control is transferred to the monitoring system where it can record the event and/or inspect the system's state. Once the monitor has finished processing the event, it returns control to the target system and execution continues until the next event. Hook based techniques are robust against failures and attacks inside the target when the monitoring system is properly isolated from the target system.

We find *dynamic* hook-based systems attractive for dependability monitoring as they can be easily adapted: once the hook delivery mechanism is functional, implementing a new monitor involves adding a hook location and deciding how to process the event. In this case, dynamic refers to the ability to add and remove hooks without disrupting the control flow of the target. This is particularly important in real-world use, where monitoring needs to be configured for multiple applications and operational environments. In addition to supporting a variety of environments, monitoring must also be responsive to changes in those environments.

In this section, we present the Hprobe framework, a dynamic hook-based VM reliability and security monitoring solution. The key contributions of the Hprobe framework are that it: is loosely coupled from the target VM, can inspect both the OS and user applications, and it supports runtime insertion/removal of hooks. All of these aspects result in a VM monitoring solution that is suitable for running on an actual production system. We have built a prototype implementation using Hardware-Assisted Virtualization that is integrated with the KVM hypervisor [6]. From our experiments, the overhead for an individual probe (the time between hook invocation and when control is returned to the VM) is 2.6 μs on a modern server-class CPU. To demonstrate monitoring using the Hprobe framework, we have constructed an emergency security vulnerability detector, a heartbeat detector, and an infinite loop detector. While our prototype framework shares some similarities and builds on previous monitoring systems, these detectors could not have been implemented on

any existing platform. All of these detectors were tested using real applications and exhibit low overhead ($\leq$5).

5.2 *Design*

5.2.1 Hook-Based Monitoring

An illustration of a hook-based monitoring system adapted from the formal model presented in Lares [13] is shown in Fig. 4. Hook based monitoring involves a monitor taking control of the target after the target reaches a hook. In the case of hypervisor-based VM monitoring, the target is a virtual machine and the monitor can run in either the hypervisor [10], in a separate security VM [13], or in the same VM [30]. Regardless of the separation mechanism used, one must ensure that the monitor is resilient to tampering from within the target VM and the monitor has access to all relevant states of that VM (e.g., hardware, memory, etc.). Furthermore, a VM monitoring system should be able to trigger on the execution of any instruction, be it in the guest OS or in an application.

If a monitoring system can capture all relevant events, it also follows that the monitoring system should be dynamic. This is important in the fast-changing landscape of IT security and reliability. As new vulnerabilities and bugs are discovered, one will inevitably need to account for them.

The value of a static monitoring system decreases drastically over time unless periodic software updates are issued. However, in many VM monitoring solutions [8, 13, 14, 30], such software updates would require a hypervisor reboot or at the very least a guest OS reboot. These reboots result in system downtime whenever the monitor needs to be adapted. In many production systems, this additional downtime is unacceptable, particularly when the schedule is unpredictable (e.g., security vulnerabilities). Dynamic monitors can also provide performance improvement over

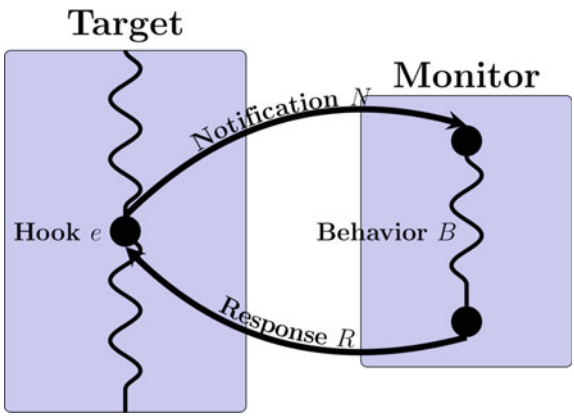


Fig. 4 Hook-based monitoring. A hook triggers based on event e and control is transferred to the monitor through notification N. The monitor processes e with a behavior B and returns control to the target with a response R

statically configured monitoring: one can monitor only events of interest vs. a general class of events (e.g., a single system call versus all system calls). Furthermore, it is possible to construct dynamic detectors that change during execution (e.g., a hook can be used to add or remove other hooks). Static monitoring systems also present a subtle design flaw: a configuration change in the monitoring system can affect the control flow of the target system (e.g., by requiring a restart).

In line with dynamism and loose coupling with the target system, the detector must also be simple in its implementation. If a system is overly complex and difficult to extend, the value of that system is drastically reduced as much effort needs to be expended to use that system. In fact, such a system will simply not be used. DNSSEC[1] and SELinux[2] can serve as instructive examples: while they provide valuable security features (e.g., authentication and access control), both of these systems were released around the year 2000 and to this day are still disabled in many environments. Furthermore, a simpler implementation should yield a smaller attack surface [58].

5.2.2 Design Principles

In light of the observation made in the previous section, we set the following design principles for a dynamic VM active monitoring system:

- **Protection**: Monitoring should be impervious to attacks (e.g., hook circumvention) inside the VM. The authors of Lares [13] outline a formal model with potential attacks and security requirements for a hook-based monitoring system. Those requirements using the notation in Fig. 5 are: the notification N should only be triggered on legitimate events, the state of the target should not change during monitoring, an attacker cannot modify the behavior B of the monitor, and the response R cannot be avoided by the target.
- **Simplicity**: The monitoring system should be simple to implement and extend. In order to ease adoption and support cloud environments, it should not require any modification of the guest OS.
- **Dynamism**: The monitoring system should be loosely coupled with the target. The target itself should be protected from changes in the monitoring system: reconfiguration can be expected to affect execution time, but it should not disrupt the control flow of the target (e.g., require a reboot or application restart). Furthermore, it should be possible to insert the hooks into both the target OS and its applications.
- **Performance**: The monitoring system should have acceptable overhead for use in a production system.

We use these requirements as a guide to design a hook-based hypervisor monitoring framework that we call hypervisor probes or hprobes. The hypervisor provides

[1] https://tools.ietf.org/html/rfc2535.

[2] https://www.nsa.gov/publicinfo/pressroom/2001/se-linux.shtml.

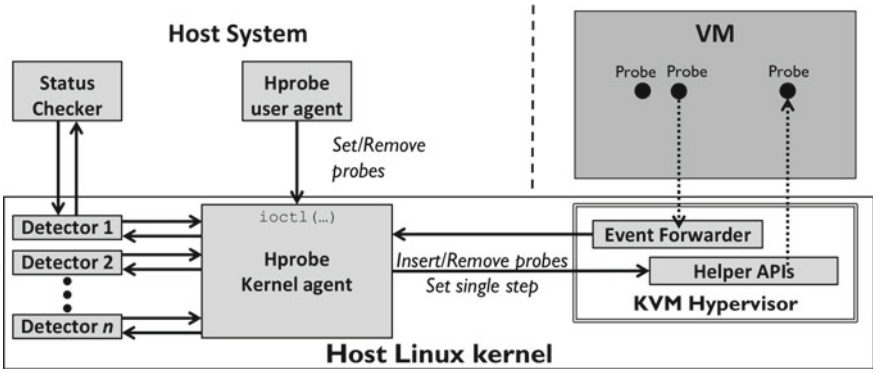


Fig. 5 Hprobes integrated with the KVM hypervisor. The Event Forwarder has been added to KVM and communicates with a separate kernel agent through Helper APIs. Detectors can either be implemented as kernel modules in the Host OS or in user space by communicating with the kernel agent through ioctl functions

a convenient interface for isolating monitoring from the VM while maintaining full access to the target VM. The proposed framework allows one to insert and remove hooks into arbitrary locations inside the guest's memory (i.e., both the guest OS and user applications) at runtime. To demonstrate the effectiveness of our framework, we built a prototype and three monitors. Two of the monitors implement reliability techniques, and the third illustrates the simplicity of using hprobes to rapidly produce a monitor that protects against a security vulnerability.

5.3 Prototype Implementation

5.3.1 Review Debugging with Software Interrupt Int3

The $\times 86$ architecture offers multiple methods for inserting breakpoints, which are used in our prototype framework. We focus on the int3 instruction as it is flexible and is not limited in the number of breakpoints that can be set. The int3 instruction is a single byte opcode ($0 \times cc$) that raises a breakpoint exception (#BP). A debugger uses OS provided functionality (e.g., a system call like ptrace() [59] in Linux) to control and inspect the process being debugged. In order to insert a breakpoint, a debugger overwrites the instruction at the desired location with int3, and then saves the original instruction. When the breakpoint is hit and the #BP exception is generated, the OS catches the exception and notifies the debugger. At this point, the debugger has control of the process and can inspect the process's memory or control its execution, e.g., by single-stepping over subsequent instructions.

5.3.2 Integration with KVM

The hprobe prototype was inspired by the Linux kernel profiling feature kprobes [60], which has been used for real-time system analysis [61]. The operating principle behind our prototype is to use VM Exits to trap the VM's execution and transfer control to monitoring functionality in the hypervisor. This implementation leverages Hardware-Assisted Virtualization (HAV), and the prototype framework is built on the KVM hypervisor [6]. The prototype's architecture is shown in Fig. 6. The modifications to KVM itself make up the Event Forwarder, which is a set of callbacks inserted into KVM's VM Exit handlers. The Event Forwarder communicates with a separate hprobe kernel agent using Helper APIs. The hprobe kernel agent is a loadable kernel module that is the workhorse of the framework. The kernel agent provides an interface to detectors for inserting and removing probes. This interface is accessible by kernel modules through a kernel API in the host OS (which is also the hypervisor since KVM itself is a kernel module) or by user programs via an ioctl interface.

The execution of an hprobe based detector is illustrated in Figs. 6 and 7. A probe is added by rewriting the instruction in memory at the target address with int3, saving the original instruction, and adding the target address to a doubly-linked list of active probes. This process happens at runtime and requires no application or guest OS restart. As explained in Sect. 5.3.1, the int3 instruction generates an exception when executed. With HAV properly configured, this exception generates a VM Exit event,

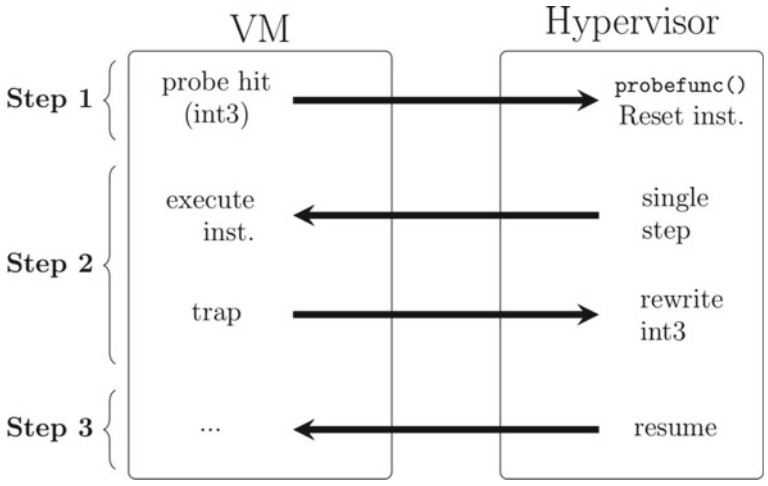


Fig. 6 A probe hit in the hprobe protoype. Right-facing arrows are VM Exits and left-facing arrows are VM Entries. When int3 is executed, the hypervisor takes control. The hypervisor optionally executes a probe handler (probefunc()) and places the CPU into single-step mode. It then executes the original instruction and does a VM Entry to resume the VM. After the guest executes the original instruction, it traps back into the hypervisor and the hypervisor will write the int3 before allowing the VM to continue as usual

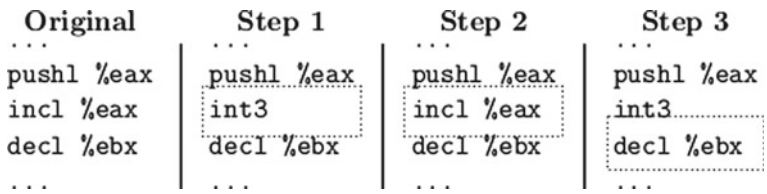


Fig. 7 Assembly pseudocode demonstrating what an hprobe looks like in the VM's memory before adding a probe (left frame) and during a probe hit (right three frames). The dashed box indicates the VM's current instruction

at which point the hypervisor intervenes (Step 1). The hypervisor uses the Event Forwarder to pass the exception to the hprobe kernel agent, which traverses the list of active probes and verifies that the int3 was generated by an hprobe. If so, the hprobe kernel agent reports the event and optionally calls an hprobe handler function that can be associated with the probe. If the exception does not belong to an hprobe (e.g., it was generated by running gdb or kprobes inside the VM), the int3 is passed back to KVM to be handled as usual. Each hprobe handler performs a user-defined monitoring function and runs in the Host OS. When the handler returns (a deferred work mechanism can also be used to support non-blocking probes, if desired), the hypervisor replaces the int3 instruction with the original opcode and puts the CPU in single-step mode. Once the original instruction executes, a single-step (#DB) exception is generated, causing another VM Exit event [4] (Step 2). At this point, the hprobe kernel agent rewrites the int3, performs a VM Entry, and the VM resumes its execution (Step 3). This single-step and instruction rewrite process ensures that the probe is always caught. If one wishes to protect the probes from being overwritten by the guest, the page containing the probe can be write-protected. Although this prototype was implemented using KVM, the concept will extend to any hypervisor that can trap on similar exceptions. Note that instead of int3, we could use any other instruction that generates VM Exits (e.g., hypercall, illegal instruction, etc.). We chose int3 since it is well supported and has a single-byte opcode.

5.3.3 Building Detectors

As mentioned in the previous section, hprobes can be controlled via an ioctl interface or a kernel API. Both interfaces distinguish between probes that are inserted into guest kernel space and guest user space. That is because while the OS always maps the kernel space pages at the same address for all virtual address spaces, each user program has its own set of pages. User space probes require the Page Directory Base Address (from the CR3 register on $\times 86$) to translate a guest virtual address into a guest physical address. Once we know the guest physical address, we can overwrite the instruction at that address and insert probes into the address space of a particular process. However, the mapping of an OS-level construct like a running process to hardware paging structures is not readily available from the hypervisor

due to the semantic gap between the VM and the hypervisor. Therefore, we use libVMI to obtain the value of the CR3 register corresponding to the target process's virtual address space [62]. This allows us to translate the virtual address of a probe location (which can be obtained from dynamic/static analysis, or by inspecting the application's symbol table) to a guest physical address that can be used to add a probe.

If one wishes to insert a probe into a user application, however, there exists another challenge. Unlike the guest OS, the pages of a running application's code may not be resident in memory at all times. That is, during an application's lifetime, some of its code may reside on disk. When execution reaches a page that is not resident, the OS will bring that page into memory. This means that the hypervisor may not be able to insert probes directly into all locations of the program at all times (i.e., it would have to wait for the OS to bring certain pages into memory). This situation arises particularly during application startup. In this case, the OS uses a demand paging mechanism in which the pages belonging to the application reside on disk until the application attempts to access one of those pages. Therefore, if the page containing the target location for a probe has not yet been accessed, a translation for guest physical address to guest virtual address will not exist. In order to support probes for user programs, this situation must be resolved so that the hprobe framework can guarantee that once a probe has been added through the APIs, it will get called on the next invocation of the instruction at the probe's desired location.

One approach to solving the problem of having target code paged out is to wait until the OS naturally brings the necessary page into memory. As mentioned in Sect. 2.2, recent versions of $\times 86$ Hardware Assisted Virtualization (HAV) use two-dimensional page tables, and do not require VM Exits for all page table updates. Therefore, in order to trap a page table update when using EPT, one must remove access permissions from EPT entries to induce an EPT VIOLATION VM Exit event. In this case, we remove write permissions from the guest physical page corresponding to the guest page table entry that refers to the guest virtual page for the intended probe location. We remind the reader that in this case the page itself is not yet present in the guest OS, and therefore a translation from guest virtual address to guest physical address does not exist in the guest OS paging structures. When an EPT violation corresponding to our protected guest page table entry occurs (indicating that the page containing the probe location is now in memory), we put the CPU into single-step mode. After the instruction writing to the guest page table executes, we can insert the probe by performing the usual translations and traversing the guest paging structures. This process of using page protection to insert probes into non-resident locations is described in Fig. 8. Note that we could improve performance slightly by avoiding the single-step and decoding the trapped instruction that caused the EPT VIOLATION. In practice, however, this paged-out situation only occurs once during the lifetime of the program (unless a page is swapped out, in which case disk latency would dominate VM Exit latency) and the performance gain would be negligible.

Oftentimes when monitoring, it is necessary to not only be aware of events in the VM (e.g., an instruction at a particular address was executed), but also the state of the VM (e.g., registers, flags, etc.). When inserting an hprobe from within the hypervisor

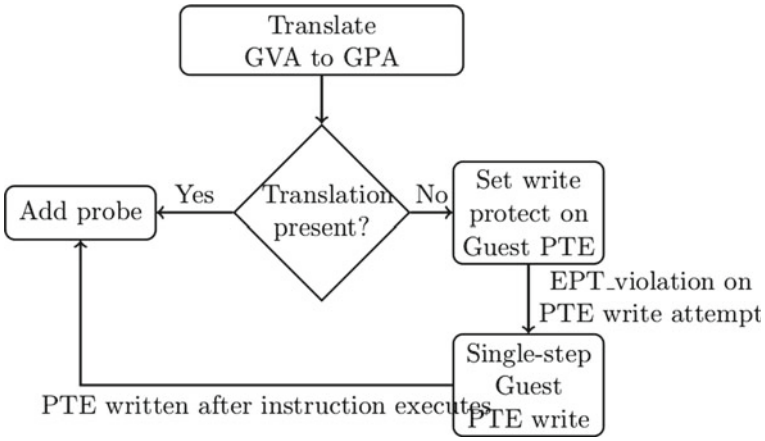


Fig. 8 How a user space probe is added. A guest virtual address (GVA) for the probe's location must be translated into a guest physical address (GPA). If the translation fails because the page is not present, we write protect the EPT page containing the guest page table entry (PTE) for that GVA. When the guest OS attempts to update the guest page table, the hprobe kernel agent is notified via an EPT violation and sets single step mode. After the single-step, the translation succeeds, and the probe is added

(i.e., using a kernel module in the Host OS), the hprobe kernel agent passes a pointer to a structure containing vCPU state to the hprobe handler. These privileged probe handlers can use this structure to decode additional information or possibly modify the state of the VM to mitigate a failure or vulnerability.

5.3.4 Discussion

Our use of int3 to generate an exception utilizes hardware enforcement of event generation: there is no dependence on any functionality inside the guest OS. This allows the hprobe hooking mechanism to be used on any guest OS supported by the hypervisor. Since the majority of the work is done outside of the hypervisor modifications (i.e., all of the heavy lifting is done inside of the kernel agent), the system can be ported to other hypervisors that support trapping on int3.

When reflecting on the requirements set forth in Sect. 5.2, we observe that the hprobe framework satisfies those requirements:

- **Protection**: By using an out-of-VM approach that is enforced by HAV, our hooks cannot be circumvented. Furthermore, we can use memory protection in the hypervisor to prevent probes from being modified (or hide them by read protecting them).
- **Simplicity**: Modifications to introduce the Event Forwarder and Helper APIs to KVM add only 117 source-lines-of-code (SLOC) and the kernel agent is 703

SLOC. The simple API allows monitors to be developed quickly and most detectors can be based on a common template (e.g., build one detector by reusing a majority of the code from a previous one). As an anecdotal example, most of the example detectors presented in Sect. 5.4 required only two hours of programming to be fully functional. Hprobes can be used on an unmodified guest OS.

- **Dynamism**: Our API allows for the insertion and removal of probes at runtime without disrupting the control flow of the target VM. Furthermore, unique to hook-based VM monitoring systems, we support application level monitoring through user space probes.
- **Performance**: While we require multiple VM Exits, we find that for our test applications and use cases, the performance is acceptable and worth the value added in the previous two dimensions.

This prototype satisfies the protection requirements adapted from Lares [13] in Sect. 5.2.2. The notification N is only delivered if events occur legitimately (spurious int3 s are ignored by the kernel agent). The context information of the event (the VM's state at event e) cannot be modified during hprobe processing since the hypervisor is in control. The security application (e.g., a probefunc()) runs inside the hypervisor and therefore, its behavior B cannot be altered by the VM. Additionally, the effects of any response R from the hypervisor are enforced since the hypervisor has full control over the target VM. Since hprobes configure VM Exits to occur on int3, one could imagine a Denial-of-Service (DOS) attack based on causing VM Exits using spurious int3 instructions. We note that hprobes do not present a new DOS threat and that if an attacker were interested in such an attack, he or she can perform it using existing functionality (e.g., using the vmcall instruction).

While using the hprobe framework does require modifications to the hypervisor, these modifications are small and robust across multiple versions of KVM and the Linux kernel. During the course of this project, we used the diff-match-patch libraries 3 to migrate the Event Forwarder and Helper APIs between KVM versions. We have tested hprobes on OpenSUSE 11.2, CENTOS7, Gentoo with kernel version 3.18.7, Ubuntu 12.04, and Ubuntu 14.04. The hprobe kernel agent is written to be version agnostic (e.g., with #ifdef macros for kernel version specific constructs like unlocked_ioctl).

5.3.5 Limitations

This prototype is useful for a large class of monitoring use cases, however it does have a few limitations. Namely,

- Hprobes only trigger on instruction execution. If one is interested in monitoring data access events (e.g., trigger every time a particular address is read from/written to), hprobes do not provide a clean way to do so. One would need to place a probe at every instruction that modifies the data (potentially every instruction that modifies any data if addresses are affected by user input). More cleanly, one could use an hprobe at the beginning and end of a critical section to turn on and

off page protection for data relevant to that critical section, capturing the events in a manner similar to livewire [8], but with the flexibility of hprobes. We are considering this in future work.
- Hprobes leverage VM Exits, resulting in non-optimal performance. This tradeoff is worth the simpler, more robust implementation with its trust rooted in HAV.
- Probes cannot be fully hidden from the VM. Even with clever EPT tricks to hide the existence of a probe when reading from its location, a timing side channel would still exist since an attacker could observe that the probed instruction takes longer than expected to complete.

6 hShield: Monitoring Hypervisor Integrity

6.1 Introduction

HyperTap and HProbes, introduced in the previous chapters, rely on the trustworthy of the underlying hypervisor to deploy their monitoring mechanisms. In this chapter, we turn the table around and validate this assumption. Particularly, we investigated VM-escape attacks, which are attacks that compromise hypervisor executions via the VM-hypervisor interface provided by Hardware Assisted Virtualization (HAV). Based on the analysis of this threat model, we introduce a new monitoring technique that detects VM-escape attacks.

In a virtualized system, the hypervisor is a single-point-of-failure. It is the centralized component that manages interactions between VMs and the underlying physical resources, such as computing, networking, and storage. Most components in hypervisor are granted high-privilege to permit access to the shared resources. If one of those components is compromised, the entire virtualized system, including physical resources and other co-located VMs, is potentially compromised as well. When an attack works on one instance of hypervisor, the attack might be extended to affect other instances, which have the same version as the exploited hypervisor.

In order to detect VM-escape attacks, we introduce a monitoring framework called hShield. The core of hShield is the incorporation of an efficient Control-Flow Integrity (CFI) enforcement method, which is specifically designed based on our analysis of HAV-based hypervisors. In addition, our CFI method addresses two fundamental limitations of state-of-the-art CFI techniques [76, 77], namely imprecise Control-Flow Graph (CFG) construction and the overhead of runtime CFI enforcement.

The design of hShield aims to provide the following features to hypervisor security monitoring:

- **Resistance to VM escape attacks that subvirt the control-flow of the hypervisor**. Many of the attacks in this class can be classified into a zero-day attack—attackers exploit an undiscovered vulnerability in the implementation of a hypervisor, which allows them to execute malicious codes together with the normal execution of the hypervisor. hShield aims at detecting this class of attacks when they are being executed without knowing the vulnerability in advance.
- **Negligible performance penalty in attack-free executions**. Similar to HyperTap and HProbes, hShield employs the principle of event-driven monitoring, which is effective in detecting both transient and persistent attacks. Additionally, we analyzed the hypervisor execution model to extract events that hShield can efficiently monitor without incurring noticeable performance overhead when the system is in an attack-free state.

In order to evaluate hShield, we compared the result of our CFI technique with that of BinCFI [77], a state-of-the-art CFI implementation. Our experiments show that the CFG constructed using our method is more precise, thus, more secure in terms of CFI enforcement. More specifically, we showed that the approximation of BinCFI's static analysis leaves dangerous paths in CFGs that can be exploited by attacks to perform a VM-escape. In addition, we showed that hShield can detect a real VM-escape attack that we crafted from a published vulnerability.

6.2 Assumptions and Threat Model

6.2.1 Assumptions

Our design targets at hypervisors that utilize Hardware Virtualization (e.g., Intel VT-x and AMD SVM) to manage VMs' executions. We make the following assumptions about the system.

The underlying hardware virtualization is implemented correctly, meaning that the only way to change from the VM privilege into the hypervisor privilege is to going through the VM-exit interface, as described in Sect. 2.2. We do not handle attacks that exploit hardware vulnerabilities.

The target host system is secured from physical tampering (e.g., secured in a server room) and there is no insider-attacker (e.g., malicious administrators who already have remote access to the host system).

The host system itself has limited direct open access from the outside world. Preventing misuse of administrative credentials, e.g., through social engineering methods to illegally obtain an administrative credential and use it against the host system, is out of the scope of this work.

The target host system is equipped with a trusted boot technology, such as Trusted Platform Module (TPM) [78], or Intel Trusted eXecution Technology (TXT) [79], which ensures the integrity of the host system, including the hypervisor, at load-time. Note that, we focus on ensuring the integrity of the hypervisor at runtime, given the

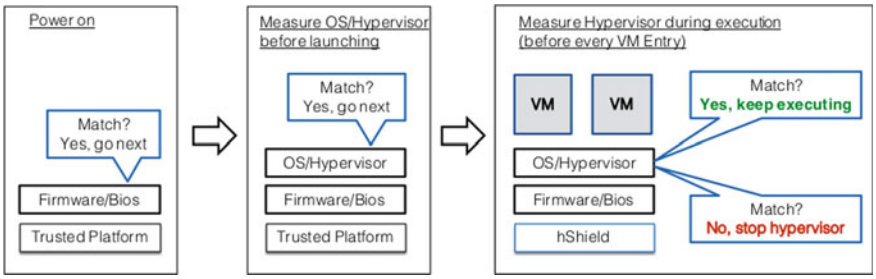


Fig. 9 hShield protects hypervisor during execution. It assumes the integrity of the platform is guaranteed at load-time by a Trusted Platform, such as TPM or Intel

integrity at load-time is guaranteed. Figure 9 shows how hShield works in tandem with trusted platform technologies.

6.2.2 Threat Model

Virtualization creates an isolated environment for each VM, so that multiple VMs can share common physical resources. The isolation is enforced so that a VM cannot access resources of the host system, or other co-located VMs.

The primary threat model that we consider is classified as VM escape attacks. A VM escape attack is an attack that breaks the isolation wall created by hypervisor to allow programs running inside a VM to violate the integrity (i.e., alter the execution) of the hypervisor. In particular, an attacker originally has full control over a VM. During the execution of the VM, the attacker is able to exploit unknown or unpatched vulnerabilities of the hypervisor software in an attempt to compromise the hypervisor. The exploit allows the attacker to redirect control flow to execute malicious code. The malicious code can be either injected by the attacker or salvaged from existing code, e.g., through a return-oriented attack. The malicious code is executed at the privilege of the hypervisor, thus it has permissions to interfere and/or access secrets stored in the hypervisor and other co-located VMs. This is a powerful class of attack. Figure 10 demonstrates the VM escape attack via VM-exit interface.

The assumption about attackers having full control over a VM is based on practical settings of virtualized computing platforms. In a public IaaS environment, such as Amazon AWS EC2, Microsoft Azure, or IBM SmartCloud, users can create a VM to run custom software with very small cost. In other virtualized environments, in which users have no direct access to a VM, attackers may gain access to a VM through exploiting vulnerabilities in the VM's software (e.g., database or web service). Once having full control over a VM, an attacker can use the VM as an entry point to start attacking the underlying hypervisor.

We further breakdown VM escape attacks into transient attacks and permanent attacks. Transient attacks are attacks that occur stealthy fast in order to bypass periodic integrity measurements [80]. Meanwhile, permanent attacks once performed

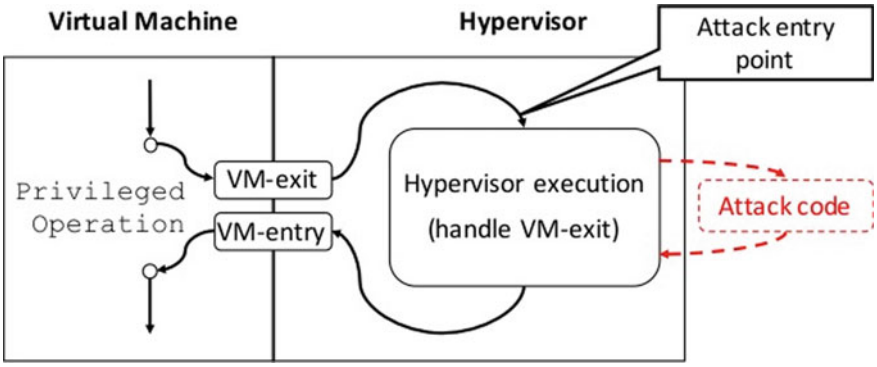


Fig. 10 Illustration of a VM escape attack in a hardware virtualization-based hypervisor. The attack entry point is the interface the hypervisor created to handle VM-exit events. The attack diverts the execution of the hypervisor (represented by the red box) from the normal execution

stay persistently in the target system. Majority of integrity measurement techniques are designed to cope with persistent attacks, leaving a gap for transient attacks to exploit [80]. Previous work [14, 15] has demonstrated the high effectiveness of transient attacks against periodic, or polling-based, monitoring. Our threat model includes both transient and permanent VM escape attacks.

6.3 hShield Approach Overview

This section describes the approach of our system, called hShield, to achieve the goals established in the previous section.

6.3.1 Limitations of Existing Control Flow Integrity Monitoring

CFI enforcement [76] is a common method used to prevent attacks relying on subverting executions of target systems (e.g., via exploiting buffer overflow vulnerabilities). In this method, valid execution paths of a program are represented as a Control-Flow Graph (CFG). The CFI runtime enforcement ensures that the target program must follow a valid path in a predetermined CFG.

A CFG is a directed graph, in which a node represents a basic block[3] in the program, and a directed edge represents a transfer in the control-flow (e.g., a jump, call, or return instruction) from a source node, where the transfer is invoked, to the target node, where the transfer lands at. Figure 9 is an example of a CFG.

[3] A basic block a consecutive sequence of instruction with no jump target except the entry and no jump source except the exit.

Runtime enforcing CFI aims at protecting target programs against unknown attacks based on the validity of CFG. A predetermined CFG is essentially a white-list of valid execution paths that are allowed to be executed. Hence, this white-list-based monitoring approach can detect attacks that divert the target program to execute an invalid path according to the determined CFG. As opposed to a black-list-based monitoring approach which can only detect previously identified attacks.

The first challenge of CFI enforcement is to obtain a precise CFG of the target program. The existing approach to CFG construction is to use static analysis [76, 77]—analyzing the source code or binary of target programs. However, static analysis cannot determine indirect control flow transfer—the control-flow targets that are computed at runtime, e.g., function pointers or return addresses. In order to address this limitation, current CFI techniques employ approximations to statically determine such dynamic targets [77].

This imprecision is a potential source for attack to by-pass CFI security runtime enforcement. For example, an attacker can use a jump-to-libc attack to invoke functions that are dynamically-incorrect, but statically-approximated.

The second challenge of CFI enforcement is to minimize the runtime overhead caused by runtime validation. The approach used by state-of-the-art CFI techniques is to perform target validation, e.g., validate whether the current jump follows a valid edge in the CFG, at the end of every basic block. The main challenge of this approach is to keep the performance overhead of the validation small due to the high frequency of basic block jumps.

6.3.2 hShield CFG Construction

hShield addresses the approximated CFG issue mentioned above by combining static analysis and profiling to construct a CFG. More specifically, we use static analysis to construct an initial CFG, which contains basic blocks (nodes in the CFG) and direct jumps (edges in the CFG), extracted from the target program binary. To derive indirect control flow information, we analyze the profiled traces of the target program execution under a set of representative workloads.

A trace records sequences of basic blocks visited during the execution of the target program. The order of basic blocks in a trace can be used to construct a CFG. For instance, two consecutive basic blocks B1 and B2 in a trace indicates that there is an edge from node B1 to node B2 in the CFG. A CFG constructed based on profiled traces contains both direct and indirect control flow information. However, the constructed CFG may not cover all possible valid paths that the target program may execute. The path coverage of the CFG is determined by the workloads used to execute and record the traces of the target program. All the collected traces are used to construct a CFG.

The initial CFG constructed using static analysis is merged with the CFG constructed based on profiled execution traces to produce a single CFG. That CFG contains both direct and indirect control flow information. This approach combines the advantages of both methods: static analysis can extract direct control flows,

and execution traces contain indirect control flows which can only be accurately determined at runtime.

For the purpose of detecting VM escape attacks, the constructed CFG of a hypervisor needs to cover all valid execution paths from a VM Exit to the corresponding VM Entry. According to our threat model, this is the only attack vector that an attacker inside a VM can penetrate the hypervisor.

Figures 11 and 12 show the result of the CFG construction for the KVM-QEMU hypervisor. Figure 11 indicates that IO INSTRUCTIONs are the most frequent type of VM Exits: 82% of VM Exits triggered during the execution of a VM under CentOS booting and the set of utilities in the UnixBench benchmark are IO-related events.

Figure 12 shows the detailed CFG construction results for QEMU using various types of workloads. In a KVM-QEMU hypervisor, all IO-related VM Exits are handled by QEMU, thus the collected events presented in the graph are IO-related events. The CFG was incrementally constructed using the traces collected by executing the workloads in order listed in the x-axis.

Each of the workloads was run three times. The graph shows that neither new nodes nor edges were discovered after the PostMark benchmark, meaning that the CFG constructed by a subset of benchmarks is able to cover all paths to execute all the selected benchmarks.

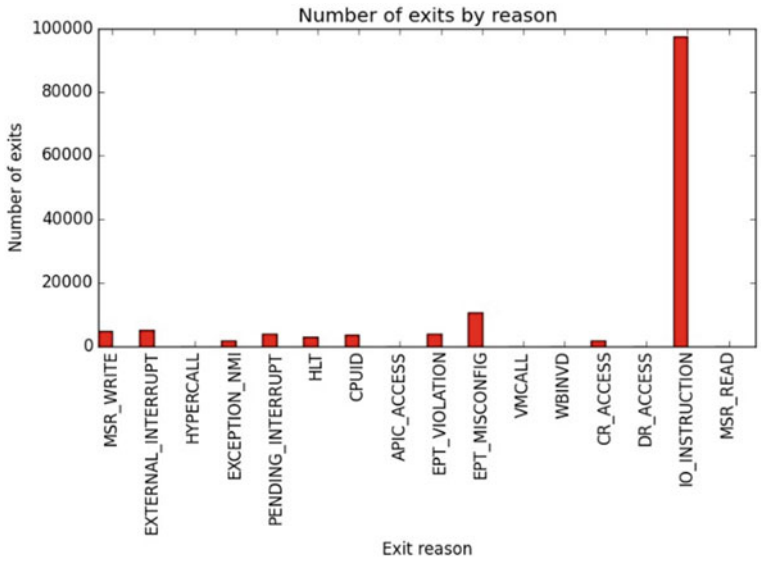


Fig. 11 The distribution of VM Exit reasons profiled during the execution of a VM under CentOS Linux booting and UnixBench workloads

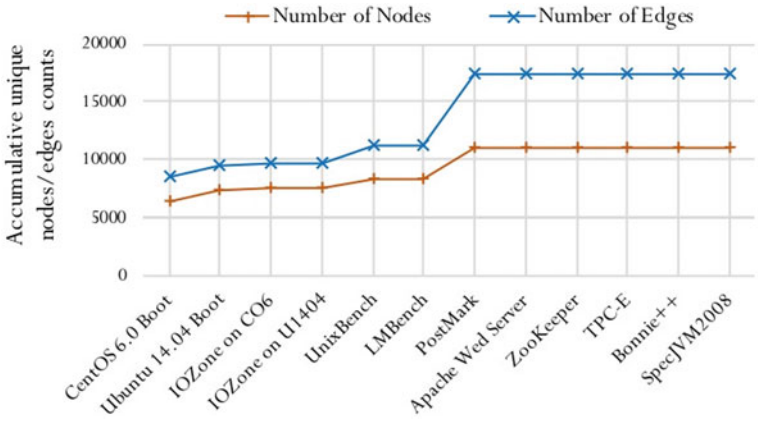


Fig. 12 Profiling QEMU (IO and MMIO exits only) under different VM workloads

6.3.3 hShield Runtime Enforcement

hShield proposes a novel technique to improve the performance overhead of CFI runtime enforcement. This technique is particularly designed for the HAV-based hypervisor execution model. Existing CFI enforcement performs validation at every control flow transfer. This validation is the major source of performance degradation occurring while executing protected programs. hShield's solution to this issue is to reduce the validation frequency by delaying it until a VM Entry is about to execute. Per our measurement, on average the frequency of executing a VM Entry is three orders of magnitude smaller than the frequency of a control flow transfer in the KVM-QEMU hypervisor.

hShield implements a hardware counter to compute a hashed value of hypervisor execution on-the-fly. Figure 13 describes how a hash is computed for each VM Exit handling. At the end of a VM Exit handling, triggered by a VM Entry event, hShield compared the computed hash against a pre-constructed HashSet. The pre-constructed HashSet represents the constructed CFG of the hypervisor. In other

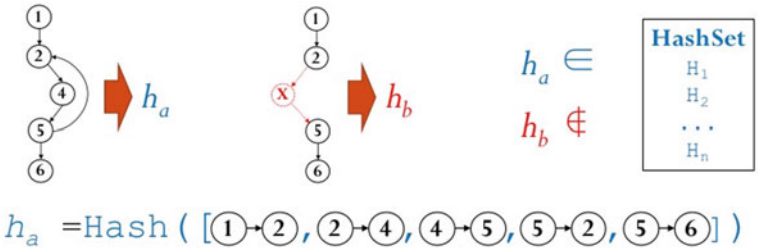


Fig. 13 HashSet construction

words, the HashSet is a white-list of valid hypervisor execution paths. If an execution path is not listed in this white-list, hShield flags it as an offended execution.

This approach of delaying CFI validation to the end of each VM Exit handling makes an important trade-off comparing the existing CFI enforcement: reducing performance overhead with the cost of longer detection latency. Since current techniques check for CFI at every control flow transfer, a CFI violation can be detected right before the execution of a malicious code. In hShield, the detection happens at the end of the violated VM Exit handling.

Section 6.4 details the hash function that hShield uses, and Sect. 6.5 describes the architectural support to hShield.

6.4 Execution Hashing

The function of execution hashing is to map an arbitrarily long execution pattern input to a fixed length output hash value. An execution pattern is a stream of machine instructions executed by the processor.

6.4.1 Requirements

The hash function needs to be collision resistant. This property is to ensure that it is computationally infeasible to find a collision—an outside execution pattern that has the same hash as one of the white-listing members. Most standard cryptographic functions, such as MD5 or the SHA family, have this property.

The hardware implementation poses several extra constraints. First, the function needs to be interactive, that is a hash can be continuously evaluated at runtime as input instructions coming, instead of storing the whole history of instructions and perform calculation at the end.

In addition, the hash function needs to facilitate the implementation of loop rerolling. hShield's loop rerolling involves frequent comparisons of basic blocks. Thus, hashing individual basic blocks should be an intermediate operation of the entire hashing scheme. Furthermore, loop rerolling requires re-evaluation of the final hashing output at runtime. For example, the hashing output changes when a loop iteration is removed. The ability to efficiently re-evaluate outputs at runtime is a necessity to enable hShield to cope with various issues, such as ones caused by hardware speculative executions. With speculative execution, a conditional branch may be predictively evaluated in advance, and unrolled and re-executed if the prediction was wrong.

6.4.2 Incremental Collision-Free Hashing

The hashing function we select is a variation of the MuHASH function in the family of incremental collision-free hashing functions proposed in [81]. The key property which makes this family of hashing functions suitable to our usage is incremental. This property allows a hash value to be updated when a portion of the input is changed without caching or re-computing the value from scratch. We leverage this feature to facilitate loop rerolling implementation and cope with speculative execution.

This family of hashing functions splits hashing into two phases: randomize and combine. Each input is broken into a sequence of blocks, and each block is randomized independently using a standard hashing function (e.g., a SHA function). The output of randomization is combined using an inexpensive commutative operation, e.g., modular multiplication in the case of MuHASH. Thanks to the communicative property of the combining operation, a hashed value can be updated by re-evaluating the randomized value of the modified input block.

Besides incrementality, MuHASH offers other properties that is suitable to hShield requirements:

- **Collision-resistance**: Based on an assumed-perfect standard hashing function (e.g., a SHA function), the security strength—the hardness of finding a collision—of the MuHASH is proven to be equivalent to the hardness of the discrete logarithm problem [81].
- **Parallel construction**: The randomization phase can be performed in parallel for each block. Note that property is stronger than interactive construction. We leverage this property to perform randomization per basic block with a small memory footprint.
- **Efficiency**: The construction uses only standard hashing function and inexpensive modular operation (as opposed to using exponentiation). The efficiency of this hashing function family is the same as using a standard hashing function on the entire input [81].

6.4.3 Runtime Construction

Essentially, the counter operates as a hash function f:

$$f : \text{Exe} \times \text{Salts} \rightarrow \text{Range}$$

The hash function f maps from the space of finite variable-length instruction streams Exe and a space of salt values $Salts$ to the space of fixed length output value $Range$.

An execution $E \in Exe$ is a finite length stream of *basic block* $B_1 B_2 ... B_n$, each basic block is a sequence of instructions $I_1 I_2 ... I_m$. Each instruction I_i is a valid $\times$ 86 instruction represented in its binary form.

A salt $salt \in Salts$ is a unique value for each system, thus it individualizes each system's counter table. A salt value is generated for a counter table when the profiling

mode is executed. Note that for a salt to be effective, it does not need to be random. Thanks to the uniqueness property of salts, the work of crafting exploit code must be redone for each every system.

A hashing session starts on a VM-exit event, and ends on the corresponding VM-entry event. The continuous construction of the hash function during a session is as follows:

- Step 1: Session starts with resetting basic block counter to i = 1:
- Step 2: For each incoming basic block B_i, concatenate a 32-bit binary encoding $<i>$ of the basic block counter, and the salt value:

$$B_i' = \langle i \rangle \cdot \langle \text{salt} \rangle \cdot B_i$$

- Step 3: (Randomization) Compute a hash value for the incoming basic block:

$$h_i = \text{shal}\left(B_i'\right)$$

- Step 4: (Combination) Combine h i using a combining operation current hash value of the execution chunk:

$$f_i = \begin{cases} h_1, & i = 1 \\ f_{i-1} \odot h_i, & i > 1 \end{cases}$$

  As recommended by [81], we use the arithmetic operation multiplication modulo for combining operator to achieve collision-resistance.
- Step 5: Continue going back to step 2 until the session is ended.

Assuming that there are n basic blocks in the evaluated execution chunk E, the final construction can be summarized in Fig. 14, and as the equation follows:

$$f(E, salt) = \odot_{i=1}^{n} \text{sha1}(\langle i \rangle \cdot \langle \text{ salt } \rangle \cdot B_i)$$

6.5 hShield Architectural Design

hShield is a security assisted hardware extension to the existing HAV to perform whitelist-based continuous monitoring of hypervisor executions. This section describes an example architectural design of hShield (Fig. 15).

6.5.1 hShield Components

Each physical host is equipped with one hShield unit. An hShield unit consists of multiple per-core hShield Counters and one per-host hShield Auditor. Each hShield

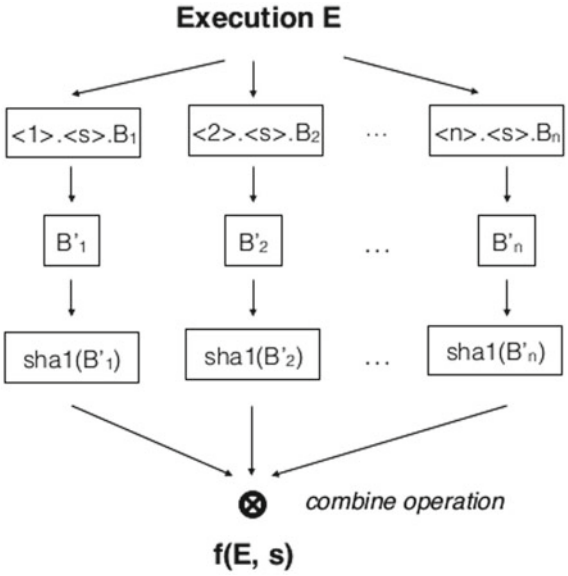


Fig. 14 The construction of the incremental hashing function

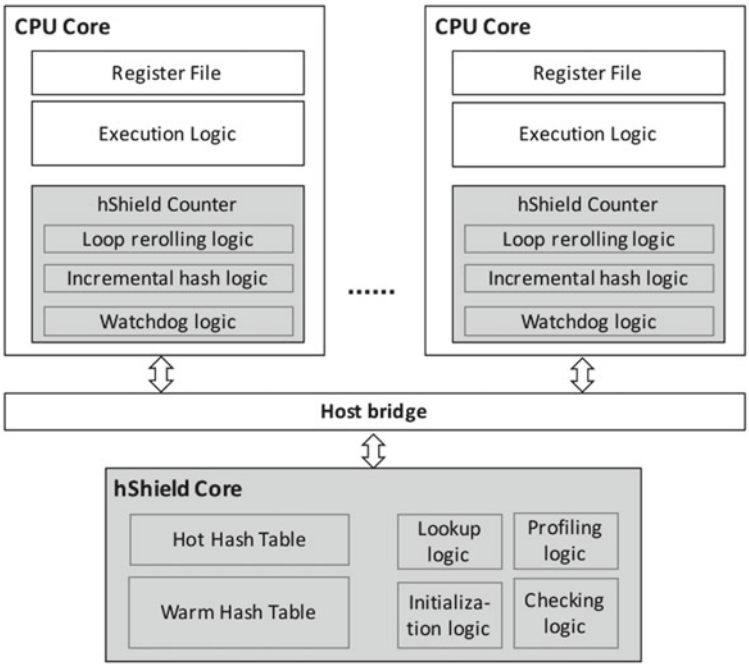


Fig. 15 hShield architecture. Each CPU core has its own hShield counter to measure hypervisor execution at runtime. After a measurement is complete, the result is sent to the hShield core, which is a dedicated core per host system, to verify the measurement.

counter is built-in into a processor core, called the counter's host core. Each counter independently carries out the measurement of VM-exit handler executing on its host core. At the end of each measurement, the result, i.e., the hash represents the VM-exit handler execution, is sent to the auditor for whitelist member checking. The hShield auditor, implemented as a dedicated co-processor in this design, is responsible for securely loading and storing the whitelist, and efficiently executing whitelist updating and membership checking. Figure 15 illustrates this architecture.

hShield is designed to facilitate both whitelist construction and runtime checking. hShield auditor has two operational modes: profiling and checking. The profiling mode is used to support whitelist construction. In this mode, the auditor records hashes sent by counters to its hash tables. Meanwhile, the checking mode is used to validate hypervisor's executions during regular runs (e.g., with arbitrary clients' VMs). In this mode, the auditor validates an execution by comparing the hash sent by a counter against in the whitelist loaded in its hash tables.

hShield architectural design follows the separation of concerns principle. After being the initialized by the centralized auditor, the operation of each counter are independent from each other, and also independent from the auditor. An hShield counter operates the same way whether the auditor is in the profiling or checking mode. The only component that stores the whitelist is the auditor. During runtime, there is only one type of unidirectional interaction between a counter and the auditor, which is sending-receiving a hash. There is no other interface that can leak information about the whitelist from the auditor to any of the processing cores.

Table 1 shows the interface of hShield Counters and hShield Auditor via the commands they process. The next subsections describe in details hShield counters and auditors.

6.5.2 hShield Counters

Figure 16 depicts the finite state machine (FSM) of an hShield counter's operation. Each node of the FSM represents an operational state of a counter, and each edge represents an event that triggers a state transition. Note that the FSM can be terminated when it is in any state, and the "End" state is not shown in the figure for readability purposes. Besides the "End" state, an hShield counter can be in one of the following operational states:

"Init": At boot time, all hShield counters are initialized by the hShield auditor. Particularly, the hShield auditor instructs each of the hShield counters to load two common salt and proof values. When the initialization is done, represented by the "Done initialization" edge, the hShield counter transits to the "Ready" state.

"Ready": When an hShield counter is in this state, the processor is executing either in the guest mode (i.e., a VM is executing), or other tasks that do not belong to the hypervisor. Upon a "VM-exit" event, the counter transits to the "Reset counter" state. Meanwhile, upon an event that indicates "Hypervisor resumed" (e.g., a task switch event that the to-be-executed task belongs to the hypervisor), the counter transits to the "Reload counter" state.

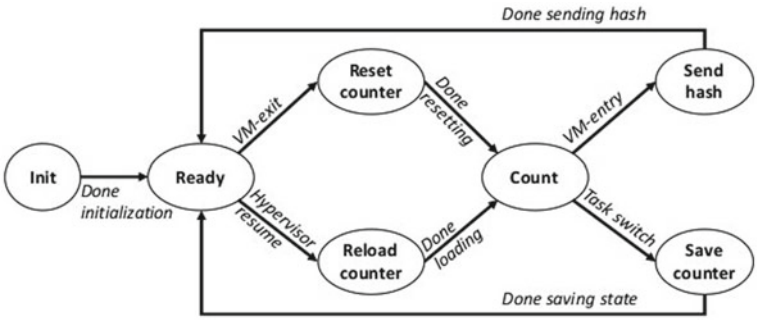


Fig. 16 Finite state machine of an hShield counter operation. A node is a state of the counter, an edge is an event that triggers a state transition. All state can transit to the "End" state, which is not shown in this figure

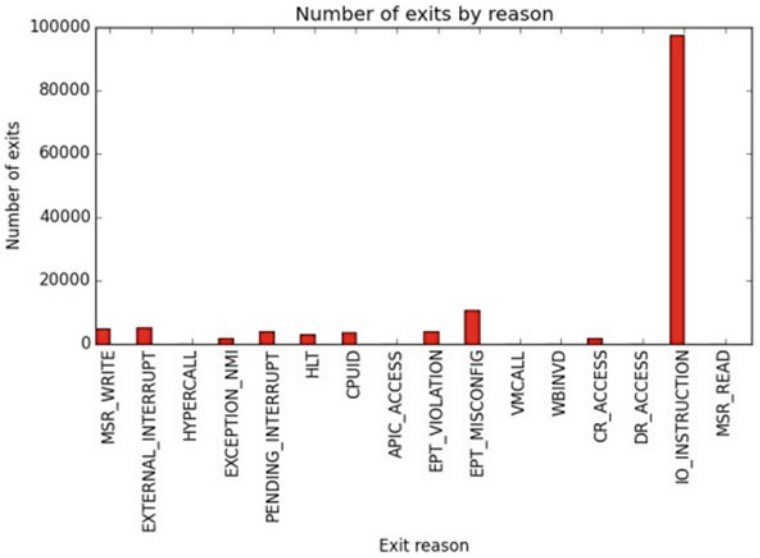


Fig. 17 The number of unique paths per each type of exit

"Reset counter": An hShield counter in this state is to respond to a VM-exit event issued by its host core. In this state, the counter resets all its internal state, e.g., the basic block counter, to get ready for a new hashing session. Upon completing the resetting, the counter transits to the "Count" state.

"Reload counter": In this state, the hShield counter loads an on-going hashing session context from memory to its internal state. The counter only loads the context which was properly signed using its hShield Proof. Upon completing the loading, the counter transits to the "Counter" state.

"Count": When an hShield counter is in this state, the host core is executing a hypervisor task that handles a VM-exit. In this state, the counter executes a hashing session, which implements the execution inference techniques and incremental hashing scheme. In the event of a task switching, the counter suspends the on-going hashing session, and then moves to the "Save counter" state. In the event of an VM-entry, which signifies the end of the on-going hashing session, the counter compute the final hash of the hypervisor execution, and then transits to the "Send Hash" state.

"Save counter": In this state, the hShield counters save the context of the on-going hashing session to main memory. The saved data is signed with the hShield Proof to prevent tampering. Upon completing the saving, the counter transits back to the "Ready" state.

"Send Hash": This state marks the end of a hashing session by sending its result to the hShield auditor. Upon completing the sending, the counter transits back to the "Ready" state.

6.5.3 hShield Auditor

An hShield Auditor is a centralized component that manages the whitelist for a host system. An hShield Auditor operates in either of the two modes: profiling and checking. Setting which mode hShield Auditor operates on is done through the BIOS.

Profiling Mode

The profiling mode is used to facilitate the construction of the target hypervisor whitelist. This mode is also considered the unsafe mode of hShield Auditor, because its whitelist can be read and updated. Thus, the profiling mode must be run in a strictly controlled environment with known-good VM workloads. In this mode, an hShield Auditor performs the following tasks.

At boot time, the following tasks are performed in a sequence:

1. Generates a new *salt* value.
2. Generates a new *proof* value.
3. Broadcasts the `HS_COUNTER_INIT` command together with the salt and proof values to all hShield Counters in the host to trigger their initialization process.

During runtime, the following tasks are performed in response to specific events:

- Upon receiving a hash from a Counter, the Auditor updates its hashing tables.
- Upon receiving a `HS_WL_COUNT` instruction, the Auditor returns the number of whitelist members.
- Upon receiving a `HS_WL_READ` instruction, the Auditor returns the hash corresponding to the specified whitelist member.

- Upon receiving a `HS_SALT_READ` instruction, the Auditor returns the value of the generated salt.

The `HS_WL_READ` and `HS_SALT_READ` instructions are used at the end of the profiling process to fetch the whitelist from the hShield Auditor to persist to the host's storage.

Checking Mode

The checking mode is used for runtime monitoring of the target hypervisor, given that the whitelist has been properly constructed. In this mode, an hShield Auditor performs the following tasks.

At boot time, after the integrity of the host system is verified, e.g., by TPM and Intel TXT, the following task are performed in a sequence:

1. Load the whitelist and salt from the host persistent storage.
2. Generates a new proof value.
3. Broadcasts the `HS_COUNTER_INIT` command together with the salt and proof values to all hShield Counters in the host to trigger their initialization process.

During runtime:

- Upon receiving a hash from a Counter, the Auditor verifies the membership of the hash.

Regardless of the hShield Auditor's operational mode, the operation of the hShield Counters in the same host is not affected: upon each VM-entry, the corresponding hShield Counter sends a hash to the centralized Auditor.

Hash Tables

Hash tables are hShield Counters internal storage to keep the whitelist. An hShield Counter contains two hash tables: hot and warm. The two tables function in the same way, except the following differences: The hot table's size is smaller than the warm table's; the hot table stores the top popular whitelist members, in the meanwhile, the warm table stores the less popular whitelist members; a membership check operation is performed in the hot table first, it there is no hit in the hot table, the operation is then performed on the warm table.

The hot-warm hash table design is to take advantage of the observed distribution of the frequency of the hit rate of whitelist members. The hot table is smaller, but stores the most frequently hit whitelist members.

7 Conclusion

This chapter proposes three new continuous monitoring methods that address both VM attack and hypervisor attack scenarios mentioned in Sect. 2. Figure 18 summarizes our contributions organized in relation to the layers in the target system (y-axis) and system operational phase (x-axis).

7.1 Continuous Monitoring of Guest OS and Applications

For monitoring software running inside VMs, we introduce HyperTap and Hprobes, which are out-of-VM monitoring frameworks that facilitate detection of security and reliability incidents occurring inside a VM. These two frameworks can work in tandem to provide desirable monitoring features. HyperTap primarily focuses on monitoring the guest OS, while Hprobes adds guest application monitoring capability. On the one hand, HyperTap relies on fixed and well-defined hardware invariants to achieve robust and strong isolation with target VMs; on the other hand, Hprobes provides a mechanism for dynamic and flexible deployment of monitoring in the target VMs.

Both HyperTap and Hprobes employ the event-driven monitoring paradigm, which allows monitors to reactively respond to events of interest. In contrast to polling-and-scanning, event-driven monitoring exposes no temporal gap for failures and attacks to exploit. In addition, the event-driven monitoring mechanisms employed by these frameworks can capture target VMs' operational activities at

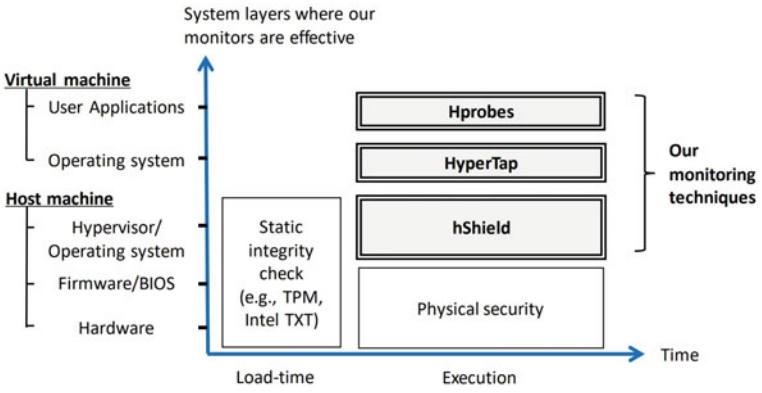


Fig. 18 An illustration of our techniques to monitor a virtualized system at runtime (e.g., during execution). The y-axis represents the system layers from hardware at the bottom to user applications in a VM at the top. The techniques are positioned at the layers where they provide monitoring: Hprobe monitors the VM's user applications, HyperTap monitors the VM's operating system, and hShield monitors the hypervisor

various granularities, e.g., system call invocations and process/task-switching events. This provides a basis of support for a broad range of security and reliability enforcement policies.

To demonstrate the capabilities of HyperTap and Hprobes in supporting security and reliability monitoring, we introduced a set of low-cost and high-coverage monitors:

HyperTap Guest OS Hang Detection (GOSHD). GOSHD detected 99.8% of injected hang failures in a guest OS. GOSHD is also able to identify partial hangs, a new failure mode in multi-processor systems.

HyperTap Hidden-Rootkit Detection (HRKD). Rootkits are malicious computer programs that hide other programs from system administrators and security-monitoring tools. HRKD guarantees discovery of hidden processes and threads regardless of their hiding techniques.

We verify the claim by testing HRKD against nine real-world rootkits in both Linux and Windows environments, with various types of hiding mechanisms.

HyperTap Privilege Escalation Detection (PED). In a privilege escalation attack, a process gains higher privileges than originally assigned to it in order to obtain unauthorized access to system resources. We demonstrate that PED can detect this class of attacks, including attacks that successfully bypassed Ninja [19], a real-world monitor, by exploiting temporal gaps created by polling-and-scanning monitoring.

Hprobes Emergency Exploit Detectors (EED). Often, a security vulnerability is discovered. After the vulnerability is made public, a patch takes time to be developed and must be put through a QA cycle. During this time, the target system is at risk of being attacked at the known vulnerability. We show that Hprobes can solve this practical problem by developing EED, a class of detectors that can prevent the exploitation of newly discovered vulnerabilities without patching the target system.

Hprobes Application Heartbeat Detector (AHD). One of the most basic reliability techniques used to monitor computing system liveness is a heartbeat detector. Using Hprobes, we constructed AHD, a monitor that directly measures the application's execution. That is, since probes are triggered by the application execution itself, they can be viewed as a mechanism for direct validation that the application is functioning correctly.

Hprobes Infinite Loop Detector (ILD). Infinite loops are a common failure that can cause process hangs. We demonstrated ILD, a monitor that uses Hprobes dynamic hook placement mechanism to measure the worst case execution time (WCET) [20] of a loop. The measure WCET is used to effectively detect infinite loops.

Table 1 hShield counter and auditor commands

Command	Callee[a]	Caller[b]	Mode[c]	Parameters	Return
`HS_COUNTER_INIT`	Counter	Auditor		()	`void`
`HS_WL_COUNT`	Auditor	Software	Profiling	()	Number of members
`HS_WL_READ`	Auditor	Software	Profiling	(`s`, `e`)	Whitelist members indexed from `s` to `e`
`HS_SALT_READ`	Auditor	Software	Profiling	()	*salt*
`HS_HASH`	Auditor	Counter	Profiling/checking	*hash*	`void`

[a]Callee is the either a Counter or Auditor, which processes the commands
[b]Caller is the component that can invoke the command. When a caller is "Software", that means this command is an instruction available for a software to use
[c]Mode is applicable for Auditor (as a callee) only. Mode specifies in which Auditor's mode ("Profiling", "Checking", or both) the command is available

7.2 Continuous Monitoring of Hypervisor

HyperTap and Hprobes rely on the trustworthiness of the underlying hypervisor to deploy their monitoring mechanisms. We demonstrate that this assumption can be violated by VM-escape attacks, which are attacks that compromise hypervisor executions via VM-exits, the VM-hypervisor interface provided by HAV. Based on the analysis of this threat model, we introduce hShield, which implements a novel Control-Flow Integrity (CFI) enforcement method to detect VM-escape attacks.

hShield continuously measures the CFI of every VM-exit handler, the basic block of hypervisor execution that handles VMs' privilege operations. The measurement is compared against a preconstructed Control-Flow Graph (CFG) to validate whether a valid path is executed. In hShield, a CFG is constructed using dynamic analysis, as opposed to the static analysis used by state-of-the-art techniques, to enhance the precision. We show that attacks can exploit the approximation of static analysis in building CFG to execute insecure paths, while our precise CFG cannot be exploited in this way.

In addition to demonstrating the strength of the constructed CFG, we show that our prototype of hShield is able to detect attacks crafted using a high-profile vulnerability in QEMU [21].

References

1. Ghemawat S, Gobioff H, Leung S-T (2003) The google file system. ACM SIGOPS Oper Syst Rev 37:29–43. ACM

2. 451 Research (2013) Theinfopro servers and virtualization study. https://451research.com/the infopro-commentator/servers-and-virtualization
3. Al Gillen, Eastwood M, Feng I, Stolarski K, Scaramella J, Chen G (2013) Worldwide virtual machine 2013–2017 forecast: virtualization buildout continues strong. IDC report
4. Intel Corporation (2014) Intel R 64 and IA-32 architectures software developer's manual volume 3 (3A, 3B & 3C): system programming guide, September 2014
5. Advanced Micro Devices Inc (2013) AMD64 architecture programmer's manual volume 2: system programming, May 2013
6. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) KVM: the Linux virtual machine monitor. In: Proceedings of the Linux symposium, vol 1, pp 225–230
7. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. ACM SIGOPS Oper Syst Rev 37:164–177. ACM
8. Garfinkel T, Rosenblum M (2003) A virtual machine introspection based architecture for intrusion detection. In: Proceedings of network and distributed systems security symposium, pp 191–206
9. Jiang X, Wang X, Xu D (2010) Stealthy malware detection and monitoring through VMM-based out-of-the-box semantic view reconstruction, vol 13, March 2010. ACM, New York, NY, USA, pp 12:1–12:28. https://doi.org/10.1145/1698750.1698752.
10. Payne BD, de Carbone MDP, Lee W (2007) Secure and flexible monitoring of virtual machines. In: Twenty-third annual computer security applications conference (ACSAC). IEEE, pp 385–397
11. Dolan-Gavitt B, Leek T, Zhivich M, Giffin J, Lee W (2011) Virtuoso: narrowing the semantic gap in virtual machine introspection. In: 2011 IEEE symposium on security and privacy (SP). IEEE, pp 297–312
12. Hofmann S, Dunn AM, Kim S, Roy I, Witchel E (2011) Ensuring operating system kernel integrity with osck. In: Proceedings of the sixteenth international conference on architectural support for programming languages and operating systems, ASPLOS XVI. ACM, New York, NY, USA, pp 279–290. ISBN 978-1-4503-0266-1. https://doi.org/10.1145/1950365.1950398.
13. Payne B, Carbone M, Sharif M, Lee W (2008) Lares: an architecture for secure active monitoring using virtualization. In: 2008 IEEE symposium on security and privacy (SP). IEEE, pp 233–247
14. Pham C, Estrada Z, Cao P, Kalbarczyk Z, Iyer RK (2014) Reliability and security monitoring of virtual machines using hardware architectural invariants. In: 2014 44th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 13–24, June 2014. https://doi.org/10.1109/DSN.2014.19
15. Wang G, Estrada ZJ, Pham C, Kalbarczyk Z, Iyer RK (2015) Hypervisor introspection: a technique for evading passive virtual machine monitoring. In: 9th USENIX workshop on offensive technologies (WOOT 15), Washington, D.C., August 2015. USENIX Association. https://www.usenix.org/conference/woot15/workshop-program/presentation/wang
16. Bahram S, Jiang X, Wang Z, Grace M, Li J, Srinivasan D, Rhee J, Xu D (2010) DKSM: subverting virtual machine introspection for fun and profit. In: 29th IEEE symposium on reliable distributed systems, pp 82–91
17. Hund R, Holz T, Freiling FC (2009) Return-oriented rootkits: bypassing kernel code integrity protection mechanisms. In: Proceedings of the 18th USENIX security symposium, pp 383–398
18. Cao P, Badger E, Kalbarczyk Z, Iyer R, Slagell A (2015) Preemptive intrusion detection: theoretical framework and real-world measurements. In: Proceedings of the 2015 symposium and Bootcamp on the science of security, p 5
19. Flo TR (2005) Ninja: privilege escalation detection system for GNU/Linux. Ubuntu Manual, http://manpages.ubuntu.com/manpages/lucid/man8/ninja.8.html
20. Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T et al (2008) The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans Embedded Comput Syst (TECS) 7(3):36
21. NIST (2015) Vulnerability summary for cve-2015-3456. Online. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-3456

22. Garfinkel S (1999) Architects of the information society: 35 years of the Laboratory for Computer Science at MIT. Mit Press
23. Spiceworks (2014) Start of SMB it report. Spiceworks report. http://www.spiceworks.com/marketing/state-of-smb-it
24. Bartels A, Rymer JR, Staten J, Kark K, Clark J, Whittaker D (2014) The public cloud market is now in hypergrowth: sizing the public cloud market, 2014 to 2020. Forrester report. https://www.forrester.com/The+Public+Cloud+Market+Is+Now+In+Hypergrowth/fulltext/-/E-RES113365?intcmp=blog:forrlink
25. Popek GJ, Goldberg RP (1973) Formal requirements for virtualizable third generation architectures, p 121. https://doi.org/10.1145/800009.808061
26. Bhatia N (2009) Performance evaluation of Intel ept hardware assist. VMware, Inc
27. Fu Y, Lin Z (2012) Space traveling across vm: automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In: 2012 IEEE symposium on security and privacy (SP). IEEE, pp 586–600
28. Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH (2006) Antfarm: tracking processes in a virtual machine environment. In: Proceedings of the USENIX annual technical conference, pp 1–14
29. Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH (2008) Vmm-based hidden process detection and identification using lycosid. In: Proceedings of the Fourth ACM SIGPLAN/SIGOPS international conference on virtual execution environments, VEE '08. ACM, New York, NY, USA, pp 91–100. ISBN 978-1-59593-796-4. https://doi.org/10.1145/1346256.1346269
30. Sharif MI, Lee W, Cui W, Lanzi A (2009) Secure in-vm monitoring using hardware virtualization. In: Proceedings of the 16th ACM conference on computer and communications security, CCS '09. ACM, New York, NY, USA, pp 477–487. ISBN 978-1-60558-894-0. https://doi.org/10.1145/1653662.1653720.
31. Liu Q, Weng C, Li M, Luo Y (2010) An in-vm measuring framework for increasing virtual machine security in clouds. IEEE Sec Privacy 8(6):56–62
32. Dolan-Gavitt B, Leek T, Hodosh J, Lee W (2013) Tappan zee (north) bridge: mining memory accesses for introspection. In: Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security, CCS '13. ACM, New York, NY, USA, pp 839–850. ISBN 978-1-4503-2477-9. https://doi.org/10.1145/2508859.2516697
33. Dinaburg A, Royal P, Sharif M, Lee W (2008) Ether: malware analysis via hardware virtualization extensions. In: Proceedings of the 15th ACM conference on computer and communications security, CCS '08. ACM, New York, NY, USA, pp 51–62. ISBN 978-1-59593-810-7. https://doi.org/10.1145/1455770.1455779
34. Pfoh J, Schneider C, Eckert C (2011) Nitro: hardware-based system call tracing for virtual machines. In: Advances in information and computer security. Springer, pp 96–112
35. Liu Y, Xia Y, Guan H, Zang B, Chen H (2014) Concurrent and consistent virtual machine introspection with hardware transactional memory. In: 2014 IEEE 20th international symposium on high performance computer architecture (HPCA), February 2014, pp 416–427. https://doi.org/10.1109/HPCA.2014.6835951
36. Estrada ZJ, Pham C, Deng F, Yan L, Kalbarczyk Z, Iyer RK (2015) Dynamic vm dependability monitoring using hypervisor probes. In: European dependable computing conference (EDCC)
37. Petroni Jr NL, Hicks M (2007) Automated detection of persistent kernel control-flow attacks. In: Proceedings of the 14th ACM conference on computer and communications security, CCS '07. ACM, New York, NY, USA, pp 103–115. ISBN 978-1-59593-703-2. https://doi.org/10.1145/1315245.1315260
38. Nergal (2001) The advanced return-into-lib(c) exploits: Pax case study. Phrack #58, Article 4. http://www.phrack.org/issues.html?issue=58&id=4
39. Zhang F, Leach K, Sun K, Stavrou A (2013) Spectre: a dependable introspection framework via system management mode. In: Proceedings of the 43rd annual IEEE/IFIP international conference on dependable systems and networks (DSN'13), June 2013
40. Pelleg D, Ben-Yehuda M, Harper R, Spainhower L, Adeshiyan T (2008) Vigilant–out-of-band detection of failures in virtual machines. Oper Syst Rev 42(1):26

41. Bishop M (1989) A model of security monitoring. In: Fifth annual computer security applications conference. IEEE, pp 46–52
42. Moon H, Lee H, Lee J, Kim K, Paek Y, Kang BB (2012) Vigilare: toward snoop-based kernel integrity monitor. In: Proceedings of the 2012 ACM conference on computer and communications security, CCS '12. ACM, New York, NY, USA, pp 28–37. ISBN 978-1-4503-1651-4. https://doi.org/10.1145/2382196.2382202
43. Wang L, Kalbarczyk Z, Gu W, Iyer RK (2006) An os-level framework for providing application-aware reliability. In: PRDC'06. 12th Pacific Rim international symposium on dependable computing. IEEE, pp 55–62
44. Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S (2013) On the feasibility of online malware detection with performance counters. SIGARCH Comput Archit News 41(3):559–570. ISSN 0163-5964. https://doi.org/10.1145/2508148.2485970
45. Rhee J, Riley R, Xu D, Jiang X (2009) Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring. In: International conference on availability, reliability and security (ARES). IEEE, pp 74–81
46. Yim KS, Kalbarczyk ZT, Iyer RK (2009) Quantitative analysis of long-latency failures in system software. In: PRDC'09. 15th IEEE Pacific Rim international symposium on dependable computing. IEEE, pp 23–30
47. Cotroneo D, Natella R, Russo S (2009) Assessment and improvement of hang detection in the linux operating system. In: SRDS'09. 28th IEEE international symposium on reliable distributed systems. IEEE, pp 288–294
48. Butler J, Hoglund G (2004) Vice–catch the hookers. Black Hat USA, p 61
49. Devik Sd. (2001) Linux on-the-fly kernel patching without LKM. Phrack Magazine #58, Article 7. http://www.phrack.org/issues.html?id=7&issue=58
50. Ormandy T (2010) The GNU C library dynamic linker expands $ORIGIN in setuid library search path. http://seclists.org/fulldisclosure/2010/Oct/257. [Online]. Accessed 29-April-2013
51. SecurityFocus (2013) Linux kernel cve-2013-1763 local privilege escalation vulnerability. http://www.securityfocus.com/bid/58137/info. [Online]. Accessed 29-April-2013
52. Jana S, Shmatikov V (2012) Memento: learning secrets from process footprints. In: 2012 IEEE symposium on security and privacy (SP), pp 143–157. https://doi.org/10.1109/SP.2012.19
53. Garfinkel T (2003) Traps and pitfalls: practical problems in system call interposition based security tools. In: Proceedings of the network and distributed systems security symposium, vol 33
54. Provos N (2003) Improving host security with system call policies. In: Proceedings of the 12th USENIX security symposium, vol 1. Washington, DC, p 10
55. Kosoresow AP, Hofmeyer SA (1997) Intrusion detection via system call traces. IEEE Softw 14(5):35–42
56. Criswell J, Geoffray N, Adve VS (2009) Memory safety for low-level software/hardware interactions. In: USENIX security symposium, pp 83–100
57. Criswell J, Lenharth A, Dhurjati D, Adve V (2007) Secure virtual architecture: a safe execution environment for commodity operating systems. In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP '07. ACM, New York, NY, USA, pp 351–366. ISBN 978-1-59593-591-5. https://doi.org/10.1145/1294261.1294295
58. Manadhata PK, Wing JM (2011) An attack surface metric. IEEE Trans Softw Eng 37(3):371–386
59. Padala P (2002) Playing with ptrace, part 1. Linux J (103). http://www.linuxjournal.com/article/6100
60. Krishnakumar R (2005) Kernel korner: kprobes-a kernel debugger. Linux J 2005(133):11
61. Feng W, Vishwanath V, Leigh J, Gardner M (2007) High-fidelity monitoring in virtual computing environments. In: Proceedings of the international conference on the virtual computing initiative
62. Payne BD (2012) Simplifying virtual machine introspection using libvmi. Sandia report
63. NIST (2008) Vulnerability summary for cve-2008-0600. Online. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-0600

64. Corbet J (2008) vmsplice(): the making of a local root exploit. Online. http://lwn.net/Articles/268783/
65. Arnold J, Kaashoek MF (2009) Ksplice: automatic rebootless kernel updates. In: Proceedings of the 4th ACM European conference on computer systems. ACM, pp 187–198
66. Vaughan-Nichols SJ (2015) No reboot patching comes to linux 4.0. Online. http://www.zdnet.com/article/no-reboot-patching-comes-to-linux-4-0/
67. Bovet DP, Cesati M (2005) Understanding the Linux kernel. O'Reilly Media, Inc
68. Spinellis D (1994) Trace: a tool for logging operating system call transactions. ACM SIGOPS Oper Syst Rev 28(4):56–63
69. Gilbert MJ, Shumway J (2009) Probing quantum coherent states in bilayer graphene. J Comput Electron 8(2):51–59
70. Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C (2007) The daikon system for dynamic detection of likely invariants. Sci Comput Program 69(1):35–45
71. Pattabiraman K, Saggese GP, Chen D, Kalbarczyk Z, Iyer R (2011) Automated derivation of application-specific error detectors using dynamic analysis. IEEE Trans Depend Sec Comput 8(5):640–655
72. Carbin M, Misailovic S, Kling M, Rinard MC (2011) Detecting and escaping infinite loops with jolt. In: ECOOP 2011–object-oriented programming. Springer, pp 609–633
73. Agesen O, Mattson J, Rugina R, Sheldon J (2012) Software techniques for avoiding hardware virtualization exits. In: USENIX annual technical conference, pp 373–385
74. Wagner J, Kuznetsov V, Candea G, Kinder J (2015) High system-code security with low overhead. In: 36th IEEE symposium on security and privacy, number EPFL-CONF-205055
75. Larson SM, Snow CD, Shirts M et al (2022) Folding@home and genome@home: using distributed computing to tackle previously intractable problems in computational biology
76. Abadi M, Budiu M, Erlingsson U, Ligatti J. Control-flow in-tegrity principles, implementations, and applications. ACM Trans Inf Syst Secur 13(1):4:1–4:40, November 2009. ISSN 1094-9224. https://doi.org/10.1145/1609956.1609960
77. Zhang M, Sekar R (2013) Control flow integrity for cots binaries. Presented as part of the 22nd USENIX security symposium (USENIX Security 13), Washington, D.C. USENIX, pp 337–352. . ISBN 978-1-931971-03-4. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/Zhang
78. Trusted Computing Group (2015) Trusted computing group: trusted platform module. http://www.trustedcomputinggroup.org/developers/trusted_platform_module
79. Intel Corporation (2015) Trusted compute pools with intel(r) trusted execution technology. http://www.intel.com/txt
80. Azab AM, Ning P, Wang Z, Jiang Z, Zhang X, Skalsky NC (2010) Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM conference on computer and communications security, CCS '10. ACM, New York, NY, USA, pp 38–49. ISBN 978-1-4503-0245-6. https://doi.org/10.1145/1866307.1866313
81. Bellare M, Micciancio D (1997) A new paradigm for collision-free hashing: incrementality at reduced cost. In: Advances in cryptology—EUROCRYPT'97. Springer, pp 163–192
82. Weaver VM, Terpstra D, Moore S (2013) Non-determinism and overcount on modern hardware performance counter implementations. In: 2013 IEEE international symposium on performance analysis of systems and software (ISPASS), pp 215–224, April 2013. https://doi.org/10.1109/ISPASS.2013.6557172

Security for Software on Tiny Devices

Saurabh Bagchi

1 Introduction

At over 9 billion embedded processors in use today, the number of embedded devices has surpassed the number of humans. With the rise of the "Internet of Things" (IoT), the number of embedded devices, their complexity, and their connectivity is exploding. These smart *"things"* include fitness trackers, smart light bulbs, smart thermostats, Amazon's Dash Button, utility smart meters, smart locks, and smart TVs. Microcontrollers executing bare-metal software have been embedded deeply into larger systems. These embedded microcontrollers are often overlooked but they control vital components of our systems, e.g., network cards, wireless controllers, hard drive controllers, SD memory cards, or near field communication in cellphones. Many of these devices (or components in devices) are low cost with software running directly on the hardware, known as "bare-metal systems." In such systems, the application runs as privileged low-level software with direct access to all processor registers, the entire available memory, and all peripherals. This is in contrast to systems with an operating system that provides isolation and manages access to security-sensitive resources. These bare-metal systems must satisfy strict execution timing guarantees, while running on constrained hardware platforms with power and dollar constraints.

Society relies on these systems to provide secure and reliable computation, communication, and data storage. Yet, they are built with security paradigms that have been obsolete for several decades. Embedded systems are generally deployed without any active defenses or mitigations and do not follow common design criteria to enforce least privileges and restricted access. Defenses that are well known for

This article summarizes work done jointly with Mathias Payer (ETH Zurich) and also reflects further discussions with Stephen Checkoway (Oberlin College) and Mathias.

S. Bagchi (✉)
Purdue University, West Lafayette, Indiana, USA
e-mail: sbagchi@purdue.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_7

desktops to protect against code injection, control-flow hijacking, or data corruption attacks are missing on embedded systems.

1.1 Ravi's Contributions on This Topic

Ravi has made some fundamental contributions on this topic. These have come more recently in the problem areas of security for teleoperated surgical robots [8, 11] and customized malware (and its countermeasure) for cyber-physical systems [10], with expanded focus on autonomous vehicular systems [4, 17] and the smart grid [13, 19]. We have learned of some design elements from these works and the community has adopted many principles and techniques for rigorous evaluation from them.

1.2 Why can't We "just" Adopt Defenses from the Server World to the Embedded World?

Protecting embedded devices in the presence of vulnerabilities poses unique challenges that are fundamentally different from desktop or server systems. Therefore, simply porting existing defenses is not an option. First, embedded systems often run directly on the hardware without an intermediate operating system or virtualization layer. The program itself is responsible for mediating access to all resources, including security-critical ones, among all the tasks. Second, due to the lack of a Memory Management Unit (MMU), embedded systems have a single flat address space where all memory locations (e.g., the locations of I/O ports) are static. Third, embedded systems are custom tailored to a specific purpose. Each type of system may have a specific hardware configuration where some I/O ports are security sensitive while others are not. Orthogonally, note that desktop defenses are incomplete and cannot defend against all code reuse attacks or information leaks, as shown through any recent attack that bypasses all existing defenses [23, 25]. Leveraging buffer overflows, use-after-free bugs, integer overflow, or type confusion vulnerabilities, adversaries can leak information and compromise software running on desktop systems despite all currently used defenses.

Protecting software against control-flow hijacking, code reuse attacks, and information disclosure is challenging for embedded systems. However, the embedded system environment also provides some unique opportunities that enable strong, novel defenses. First, whole program analysis on these systems is feasible. Due to cost, power, and environmental constraints, the software code base running on these systems is usually kept small. Best coding practices result in limited stack depth, restricted use of indirect control-flow, limited use of recursion, and fixed memory allocation. These bound the exploration space so that static analysis can be applied to entire programs. Second, the source code of each component is generally available

to the developers as all components are developed by the same company. Even when libraries are used, all code is compiled to a monolithic binary and combined through Link Time Optimization (LTO). Third, both the software running on an embedded system and the underlying hardware are single purpose, further simplifying the analysis. The software running on embedded systems has limited functionality, often with a single purpose—compared to desktop systems with hundreds of parallel processes. Similarly for the hardware, each hardware unit is dedicated to a single executing process, whereas on desktop systems, the device is shared among multiple processes. The combination of these opportunities enables us to scale static and dynamic analysis techniques to full embedded systems and to devise strong protection mechanisms that respect the above-mentioned domain-specific constraints.

Our solution approach: RESIN. In our prior and ongoing work, we seek to solve the problem of protecting embedded systems against a wide variety of attacks, without the need to rearchitect the entire application. Our approach has 3 interdependent high-level tasks. We show a schematic of our overall system in Fig. 1. The parts where the user/developer need to provide input are shown in the salmon colored boxes according to the legend.

1. **Task I: Guided IoT exploration**. In this task, we develop targeted static analysis to identify the control and the data flow in the program. This is augmented with an

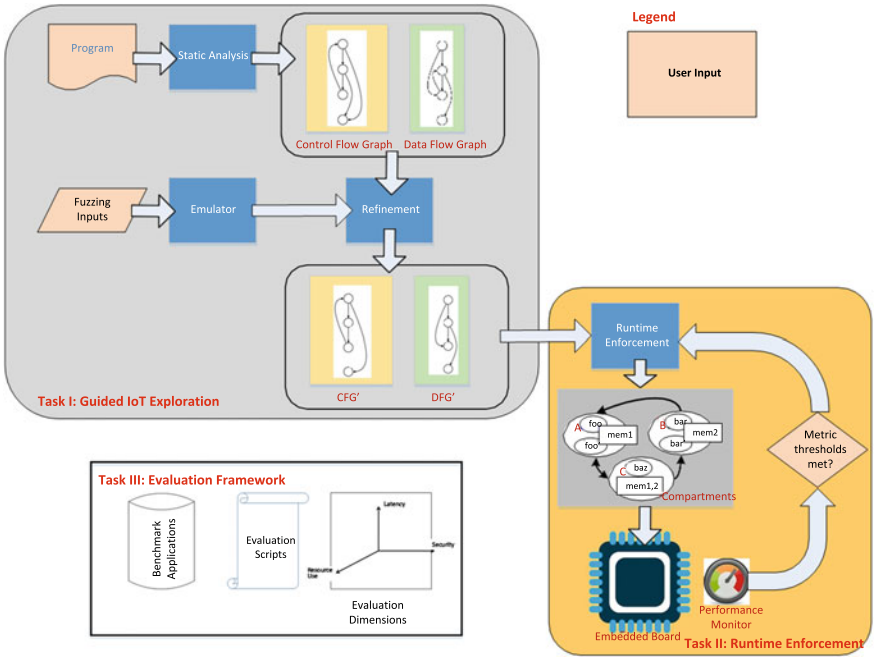


Fig. 1 Overview of the main components and their placement in the overall system RESIN

IoT emulator which emulates the runtime behavior under fuzzed and controlled inputs to discover information that is outside the scope of static analysis.
2. **Task II: Runtime enforcement**. Here we develop runtime enforcement techniques for enforcing the principle of least-privilege execution, which is considered standard security practice, but is absent in embedded system execution. We will operate with feedback provided by the constraints of the hardware and the performance impact due to the isolation of multiple compartments of code and data.
3. **Task III: Evaluation framework**. We develop a rich set of bare-metal system applications to stress different functionalities in representative use cases. We develop these benchmarks for a set of embedded boards that provide different hardware capabilities (e.g., access to different sensors) and develop scripts for evaluating different aspects of security and performance. The security and performance metrics combine the domain-agnostic as well as the domain-specific ones. The latter includes an understanding that performance needs to be deterministic for our target domain.

Target domains. We demonstrate the benefits of RESIN through realistic applications developed in five security-critical, target domains on real hardware.

1. **Smart homes**. Devices such as the Amazon Dash button, smart light bulbs, smart door locks, and per-room temperature sensors increase convenience in an modern home but, in the hands of an adversary, can result in safety and privacy hazards.
2. **Wearables**. We are increasingly tracking different aspects of our lives through heart rate monitors, activity trackers, smart shoes, or smart watches. These devices have access to highly personal data and may need to communicate urgent and critical health indicators.
3. **Smart cities**. Modern cities are increasingly connected with smart, battery-powered sensors placed in side walks and streets to detect pedestrians, bikes, and cars. These devices are low-powered, embedded, run real time, and communicate wirelessly. Protecting these devices is crucial for roadside safety.
4. **Connected transportation and infrastructure**. A modern car contains dozens of safety–critical, connected embedded devices that communicate over shared buses as well as over a variety of wireless interfaces including Bluetooth and 4G LTE. Vehicle-to-vehicle and vehicle-to-infrastructure communications are starting to be built and deployed.
5. **Industrial control systems**. Physical processes in industrial settings are computer-controlled. These safety–critical systems can be exploited to cause great harm.

Our work focuses on the sort of low-powered, embedded devices that are ubiquitous in these domains.

2 Background and Related Work

2.1 Embedded System Development

An embedded system is often meant to perform a dedicated function in contrast to a general purpose computer. Frequently this is a component within some larger system. In a vehicle, for example, there are multiple dedicated embedded computing systems, e.g., to control the anti-lock brakes, to monitor and control the vehicle's emissions, or to display information on the dashboard. The number of embedded systems has risen rapidly and today, less than 2 percent of microcontrollers manufactured are used in general-purpose computers [5]. An important sub-class of embedded systems have real-time requirements and therefore security mechanisms proposed by us or others, cannot afford to perturb the timing of the software to any significant extent. Importantly, for such systems, it is crucial to guarantee the timing properties and thus our security mechanisms must also minimize the variance in the timing that they introduce.

Certain common hardware constraints on embedded development are: (i) processing power—these devices are typically 16-or 32-bit, running at up to a few hundred MHz-s, driven by requirement of low dollar cost and low power consumption (say, 1 mW/MIPS to 10 mW/MIPS); (ii) memory—the RAM is typically up to a few 100 s of kBs and the flash which contains the program is typically up to a few MBs; (iii) slow buses—such as SPI or I^2C, which are relatively slow compared to the processor. Overall, our target class, the bare-metal embedded system, is a highly cost-conscious segment of the market. There are typically one or more peripherals attached to an embedded board. These provide functionality such as USB or sensing and they are typically accessed through a Hardware Abstraction Layer (HAL), which eases the programming by abstracting away low-level control signals and other details needed to access the peripherals. For specificity, let us consider one device that fits within our target domain. Figure 2 shows ARM's memory model for the ARMv7-M

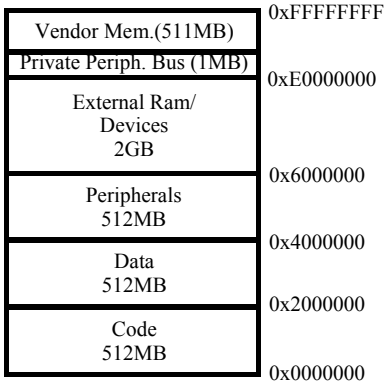


Fig. 2 ARM's memory model for ARMv7-M devices

architecture. It breaks a 32-bit (4 GB) memory space into several different regions. It is a memory-mapped architecture, meaning that all I/O is directly mapped into its memory space (peripherals, and external devices). While the architecture reserves large amounts of space for each area, actual devices only use a small portion of it. For example, the Cortex-M4 (STM32F479I) device we use in our evaluation has 2 MB of flash in the code area, 384 kB of RAM, and uses only a small portion of the peripheral space.

Embedded software development has traditionally been done mostly in C, with some limited use of assembly code. Low-level coding often requires close interaction with the hardware platform, even though the HAL does abstract away the lowest levels of hardware detail. Because of the hardware resource constraints, embedded software often is very compact, at the cost of readability and generalizability (to different hardware platforms). Software is usually written assuming any memory or peripheral can be accessed any time and from any region of the code. Further, the preferred mode of programming is event-driven programming, whereby the software reacts to external stimuli such as a sensor providing a value sensed from the physical world after running it through its analog-to-digital converter. There are expectations from embedded software that it will run unattended for long periods of time (say, months) and the end user will have limited ability (if at all) of programming the system "in the field." The vendor of the embedded board usually provides the compiler and linker tool chain to convert the C program to executable code. There has been robust development of LLVM-based toolchains for various target embedded platforms. We will leverage this trend by building our toolchain on top of LLVM as additional passes or modifications to existing passes.

Certain software design patterns frequently occur in embedded software. First, the software statically allocates all the memory that it will require, rather than relying on dynamically allocating memory. The second is the careful and parsimonious use of memory, such as fitting multiple, possibly unconnected, variables into a single register. Third, debugging tools in embedded development are more limited. At the powerful end of the spectrum is a JTAG-based debugger, often called an in-circuit emulator (ICE). In fact, in a 2015 survey of embedded developers, debugging was found to be the single greatest challenge [30].

A commonly found piece of hardware in our target class of devices is the *Memory Protection Unit (MPU)*. It enables setting privileges on regions of memory, which control read, write, and execute permissions for both privileged and unprivileged execution modes. On the ARMv7-M architecture for example, the MPU can define up to eight regions, numbered 0–7. Each region is defined by setting a starting address, size, and permissions. We assume, at the high end, machines with an MPU but without a *Memory Management Unit (MMU)*, such as Cortex ARM M0 to M4. These machines generally run in a 32-bit address space. Machines with an MMU are out of scope. The cost for adding an MMU to embedded systems is seen as prohibitive and does not fit current software design patterns where code runs bare-metal or with only a thin operating system layer. At the low end, we assume 8051-style or Atmel AVR-style Harvard-like machines without MPU or MMU. These machines generally

run with a 16-bit address space, sometimes with multiple 16-bit address spaces, e.g., for ROM and RAM.

2.2 Threat Model

IoT devices are heavily connected and susceptible to different forms of attacks. For the research proposed here, we assume that the software running on the devices contains software flaws (bugs) that are reachable through potentially adversary-controlled outside input. Input, including malicious input, to the device can be local or remote. Local input is any input that requires close physical proximity to the IoT device such as a connection through local I/O, a serial port, or near field communication such as Bluetooth or ZigBee. An example of local input is data access through a diagnostic port. Remote input is something that can be sent over a network interface, such as WiFi. All IoT devices communicate with the cloud in some form. Remote input requires an internet connection. If the IoT device runs any services, any internet device can connect to those services. An example would be a listening telnet server.

Any data attack where an adversary connects to the device, intercepts a connection from the device, or uses an I/O port to communicate with the device is in scope. We assume that the software running on the IoT device has flaws that are reachable through adversary-controlled input. Our threat model includes both data confidentiality and data integrity attacks during the runtime of the device. Verifying the integrity of the device at boot time is out of scope. Cryptographic attacks that break encrypted communication with the cloud are out of scope but implementation bugs in cryptographic protocols remain in scope. Physical attacks—for example, flashing a new firmware onto the device—are out of scope.

2.3 Lack of Defenses on Embedded Systems

Due to hardware resource and development constraints, current IoT systems lack any mitigations against memory safety violations that people have become accustomed to for desktop and server systems. Full scale operating systems leverage virtual memory to isolate processes from each other. Best programming practices ensure that each process runs with least privileges and only communicates with other processes through a well defined API. Operating systems restrict access based on fine-grained permissions and capabilities along access control lists. Inside the process, the MMU ensures the separation between code and data (DEP) and allows segments to be placed at random locations whenever a process is started (ASLR). Additional mitigations such as stack canaries or control-flow integrity (CFI) [7, 29] may be added on a per-process basis as part of the compiler toolchain.

Compared to full hardware support and regular operating systems, embedded systems are much more constrained. First, the hardware is highly constrained and

storage or memory overhead are hard to justify due to the additional cost. The embedded devices we target also do not have an MMU and, at best, use an MPU to overlay privileges on a flat physical address space. The lack of an MMU makes defenses such as ASLR impossible as they require a virtual address space. Second, embedded operating systems generally do not enforce isolation between the operating system kernel and the individual processes or even among processes—the memory structure is comparable to a set of threads that run in the same address space together with privileged software. The lack of separation between privileged code (kernel code) and unprivileged code (the applications) prohibits defenses such as DEP as all privileged memory and peripherals are directly reachable from unprivileged code. Third, the rigid compiler toolchain and development environment with lack of good debugging facilities hinders compiler innovation and prohibits the use of modern compiler-based defenses such as stack canaries or CFI. While these mitigations would have to be adapted for embedded systems to fit their unique constraints, there is no fundamental reason why they should not be used.

3 Guided IoT Exploration

IoT software is fundamentally different from desktop or server software. Both desktops and servers run multiple applications at different privileges (e.g., different users, separation between kernel and user-space, or virtualization). Low-end IoT devices are highly resource constrained. IoT devices have limited CPU, memory, power, and communication abilities and software is generally highly adapted to these devices. Due to this customization, existing software analysis techniques do not apply to IoT systems. In this task we develop static and dynamic analysis techniques that infer information about IoT applications. This information is then leveraged in task II to enforce strong security policies such as per-task compartmentalization, targeted memory safety to protect against control-flow hijacking and data-only attacks, and event-aware state protection. Note that all policies are geared towards the special circumstances IoT systems run in.

Advantages of IoT software are that application source code is generally available, the amount of code is manageable, and frameworks such as the ARM Mbed IoT platform [2] generalize common tasks such as access to peripherals. Unfortunately, these advantages are offset by several challenges to security in low-end embedded systems: (i) many embedded systems engineers have little or no security experience resulting in code that does not follow security best practices, (ii) programmers face the complexity of cross cutting concerns across all layers of the stack: from low-end I/O and pin management to high-end application concerns such as communicating with a backend server in the cloud, (iii) applications are developed in an ad hoc manner on stale tool chains (i.e., the compiler is rarely updated due to the complexity of setting up a cross-compilation tool chain), and (iv) lack of defense mechanisms, mitigations, and analysis methods (e.g., static analysis or fuzzing) that are used ubiquitously on desktop and server software.

Preliminary work. We have broad experience in protecting different forms of embedded systems and in developing sanitizers and mitigations to find a wide variety of vulnerabilities. Our most recent work to protect bare-metal embedded devices is EPOXY [12]. EPOXY is an LLVM-based mitigation that enforces a light privilege overlay, dropping privileges for all instructions and selectively raising privileges for a few privileged operations such as writing to I/O registers. Based on this privilege overlay, we enforce data execution prevention to prevent code injection, a safe stack [18] to protect against return oriented programming, and diversification to protect against data-only attacks.

Earlier, we have developed a more holistic defensive approach that enforces full memory safety for tiny embedded systems through nesCheck [20]. This work leverages a CCured-like [21] pointer analysis that classifies pointers as safe, sequence, or dynamic, allowing different instrumentation depending on the type of the pointer. The overheads for nesCheck are higher than for an EPOXY-based approach. Both EPOXY and nesCheck protect different classes of embedded systems against wide types of attack vectors at low overhead, adhering to performance and power constraints of embedded devices.

Orthogonally, we have developed a wide set of sanitizers that enforce security policies for regular software systems such as Desktops or servers. We have worked on Control-Flow Integrity [7, 9, 14, 22, 29], a mitigation that protects against control-flow hijacking by checking that the target of control-flows observed at runtime belongs to the set of valid targets. As an extension to CFI, we have explored a mechanism that keeps state for variadic function calls [6], allowing us to make the relationship between caller and callee explicit and to check argument types whenever they are used. The arguments of variadic functions (e.g., printf) depend on an implicit contract between caller and callee and cannot be checked statically by the compiler. Our mechanism enforces a dynamic runtime integrity check to ensure that the arguments pushed by the caller are correctly used by the callee.

Type confusion [15, 16] is another attack vector that enables memory corruption as a secondary effect. Type confusion abuses differences between object sizes to compromise systems. Our mechanisms track type information of all live objects to ensure type integrity for all type conversions and type checks.

Approach. To address the lack of defenses, we require detailed information about individual IoT applications to automatically employ defenses given the sparse resources available on IoT platforms. In a first step, we therefore propose novel static and dynamic analysis methods to recover necessary information about the IoT applications. We address the diversity of the IoT environment by developing a hardware abstraction language that encapsulates the differences between individual instruction set architectures, resource configurations, and availability of sensors and actuators in a portable manner. Second, we develop an event-aware static analysis that decodes event loops of embedded devices. As a proof of concept defense, we develop an event-aware version of CFI for IoT devices. Third, we develop an emulator to simulate different IoT configurations and software. IoT applications are often event-based, so we need precise knowledge of different program paths and interactions with the

underlying hardware (such as sensors and actuators), based on a hardware configuration. Indirect control-flow transfers remain challenging for any static analysis due to the aliasing problem. We use emulator-based tracing to handle the limitations of static analysis in handling indirection. We leverage the fact that IoT applications often have constrained control paths that they execute. This will allow the community to test their IoT software for security weaknesses, allowing precise bug discovery and targeted patches for vulnerable software.

4 Runtime Enforcement Techniques

In Task I, we create accurate control and data flow graphs using both static analysis and fuzzed data inputs. In this task, we take this information and automatically, in a policy-driven manner, create containers of code, data and peripherals, which serves as fault containment domains. Here we develop graph theoretic algorithms on the above-mentioned graphs to enforce the principle of least privilege (Task II.1) and then we enforce isolation through compartments, which are realized through available hardware resources, however scarce they may be (Task II.2). We then monitor the execution of the application with the initial degree of compartmentalization and incrementally change it if the performance impact is unacceptable (Task II.3).

Preliminary work. We have used the MPU, commonly available in embedded devices, in pre-liminary work [12] to create a proof-of-concept called EPOXY with simply two privilege levels of software. This provides the foundation on which code integrity, adapted control-flow hijacking defenses, and protections for sensitive I/O can be applied, by building on the "two privilege level" idea. We have evaluated the performance of our combined defense mechanisms for a suite of 75 benchmarks and 3 real-world IoT applications. Our results for the application case studies show that EPOXY has, on average, a 1.8% increase in execution time and a 0.5% increase in energy usage; however, the worst-case execution overheads will make the technique unusable for many applications. There are some specific technical constraints imposed by each generation of MPU, such as, for the MPU on the ARMv7-M architecture, each region must be a power of two in size, greater than 32 and start at a multiple of its size (e.g., if the size is 1 kB then valid starting addresses are multiples of 1 kB). Regions can overlap with the high numbered region's permissions taking effect. The MPU's hardware restrictions significantly constrain the design of compartments. For example, of all the MPU registers available (only 8 to start off with), several are used for enforcing basic protections such as making the code region not writable. The number of compartments available at any point in the execution is restricted to those that are remaining.

The use of MPUs to create isolation boundaries has been proposed and developed by ARM in its mBedOS platform, through a software module called μVisor [3]. Using this, the ARM development environment *allows* a developer, but does not provide any automation support, to create multiple "boxes." Each box gets its own memory region, including stack, and interactions among boxes are monitored and

allowed/disallowed by trusted code called "gateways." The usability challenge with the current concept is daunting. In our work, we fundamentally reduce this usability barrier by providing novel techniques to automatically infer data and control flow, and from that and the policies for security enforcement, automatically create the isolated containers.

4.1 Task II.1: Automatic Least Privilege Separation

In this task, we take the control flow and data flow graph created in Task I and create compartments out of them to achieve the desired goal of least privilege execution. The graph nodes are partitioned into disjoint compartments with the invariant that at any point of time in the execution, only a single code region belonging to the currently active compartment is executing. Further, that code region only has access to the data regions and peripherals that are within that compartment. The graph algorithms will have the goal of creating the appropriate-sized compartments, balancing the needs of the performance overhead and hardware resources used versus the level of privilege separation achieved. To expand on this, if there are fewer compartments, then there is less performance overhead and hardware resources (such as, MPU regions) used, but there is a higher degree of privilege to more code regions, thus reducing security.

We develop the graph algorithms using both static and dynamic information (from Task I). The static information contains the graph structure—the nodes and the edges, while the dynamic information annotates the edges with the frequency and nature of interactions. The latter can include for example the amount of data being passed among the compartments. We design and develop three variants of graph algorithms of progressive complexity. In all of these, we use the insight that the graphs for embedded software are likely to be much smaller than for general-purpose software and are relatively sparse in terms of indegree and outdegree.

1. No code or data motion: This will operate without a feedback loop and create compartments in one shot. Thus, this will not take into account the possibility of moving code or data to create more compact compartments.
2. Automatic code or data motion: This will run in multiple passes (we anticipate 2–3) where each pass will indicate the quality of compartmentalization and this will trigger some movement of code regions or data regions to create more compact compartments. The necessity of multiple passes arises because there is a coupling of the two steps—the creation of the compartments and the layout of code and data in memory.
3. Programmer annotation: This will be driven by an objective function where the amount of exposed code at any point in the execution needs to be minimized subject to the hardware and the performance constraints. The exact performance impact may not be known at the outset and will be fed back as input from Task II.3. If this objective function does not reach a certain specified value, which practically speaking is likely to be specified as an improvement over the

baseline, then the programmer will be requested for annotation about criticality of code regions. Alternately, the criticality can be inferred by doing some form of scalable taint tracking [32] to determine which code regions are more susceptible to unvalidated user input.

4.2 Task II.2: Enforcing Isolation Among Compartments

The goal of this task is to put in place the embedded software to enforce isolation among compartments for control and data. A compartment may access data only within its own compartment or some data that is explicitly marked as shareable. Control flow can go from one compartment to another compartment with the mediation of some privileged code, which will validate that the transition is allowed, as determined by one or more of static analysis, paths learned through fuzzing, or developer annotation. Such privileged code will form part of the trusted computing base for our system RESIN and will thus have to be minimized.

The way we envisage this task working is that the embedded program will be instrumented to trigger the privileged code whenever the code region within compartment A invokes the code region in compartment B. For example, if the code granularity is simply a function, then the call and the return instructions can be instrumented. The privileged code enforces the appropriate check, namely, that a control flow transfer is allowed here. This can be inferred from the static analysis, augmented with the trace-based emulation. If there are further violations detected at runtime, then this will be stored in a trace, for further offline, post-mortem debugging. We expect that such a trace will be highly compressible, drawing from our prior insights from deterministic record and replay in such embedded platforms [28]. The insight here is from the regular pattern of embedded application executions, as introduced in subsection 2.1.

We use MPU permissions to enforce the compartment-specific constraints. An MPU register can designate a contiguous region of memory to be read/write/execute, for privileged or unprivileged code. However, the number of registers is limited (8 in current ARMv7-M architecture, 16 in the next generation). Therefore some compartments have to be merged. This can be achieved by a mix of code and data motion and increasing the range of addresses accessible to some code regions. In general, the more interconnected the CFG and DFG are, the more challenging it will be to move all the relevant code and data regions into the same compartment. Our initial examination of baremetal applications (as in [12, 20]) has shown that the graphs have a bi-modal characteristic—some parts are sparse (where the code accesses a few libraries and no other code region is dependent on it), while some parts are dense (code regions in the hardware abstraction layer which are accessed by multiple higher-level code regions).

A broader design space that we need to consider is isolation versus resource requirements, e.g., separate stack for each compartment versus shared stack. If it is a shared stack, then portions of the stack will have to be protected, such as, only

some parts of the caller's stack should be accessible from the callee. This results in a greater requirement for MPU registers. But if the caller and the callee stacks are kept separate, then this has higher overhead in terms of the memory usage and the runtime overhead of switching between the stacks of the different compartments. Note that mBedOS, the open source embedded operating system from ARM that runs on the Cortex-M microcontroller, requires separate stacks for each compartment ("box" in their terminology).

5 Evaluating Security

Building defenses for embedded systems is only worthwhile if the defenses stop attacks without compromising correctness, performance, or energy usage. In essence, we need an objective method to measure the characteristics of interest before and after applying the defense.

5.1 IoT Metrics

Meaningful metrics are an essential component of any evaluation methodology. We would like to be able to say that Approach A provides more security than Approach B with respect to Attacker C. Unfortunately, good qualitative and quantitative metrics for security have thus-far proved elusive. The difficulty of constructing useful metrics is, in some respects, intrinsic to security. As an illustration of this difficulty, consider hardware-enforced, per-page memory protections with a write-xor-execute (W X) policy where no page of memory can be both writable and executable. Computer systems where this policy is strictly enforced (e.g., in Apple's iOS) appear to be more secure than computer systems without such a policy-enforcement because the policy prevents attackers from injecting and executing new code as well as modifying existing code. More advanced exploitation techniques, such as return-oriented programming, may still allow attackers to exploit vulnerabilities in the system. It is thus difficult to say that W X enforcement leads to increased security compared to no enforcement. In essence, it is difficult to quantify security gains from a defense mechanism, even if that mechanism rules out entire classes of attack techniques.

Despite the difficulty, several approaches to measuring the security of control-flow hijacking defenses have been previously proposed. The first is a tool-based approach wherein a tool like ROPgadget [24] is run over a binary to determine the number of return-oriented programming gadgets existing before and after a defense is applied. The second approach tries to quantify how much an indirect control transfer instruction's target set size has been reduced [31]. Both approaches were the best metrics at the time of their introduction; however, they are flawed and do not provide a sufficient notion of security.

Instead, we will develop qualitative and quantitative metrics for IoT security building on our preliminary work [7]. Qualitatively, we consider the defensive mechanisms' strengths in terms of the classes of attacks mitigated by the mechanism. For example, a defense mechanism can be evaluated in one dimension by considering whether it allows attacker-controlled memory writes to memory-mapped I/O registers or not. Whereas [7] uses the sets of instructions that can be targeted by control-flow instructions, we will use our analyses from Task I to abstract the notion of target sets to sets of input constraints for transferring control to those instructions. Similarly, we will abstract sets of memory locations that can be written by memory-storing instructions to sets of constraints on writing to those locations. Based on these constraints and the privileged and unprivileged compartments as described in Tasks I and II, we will construct quantitative security metrics. The input constraints are a refinement of target sets. As a result, our metrics will be more precise and better capture the security properties of the system under test.

Since introducing security mechanisms invariably involve trade-offs, we will also measure performance (both raw performance as well as any performance variation due to the mechanisms); resource utilization such as memory, flash-storage (e.g., for code size increases), or power; and reliance on hardware capabilities (such as the number of MPU registers required).

Undoubtedly, the impact on security, performance, and resource utilization of some defense mechanisms will be "tunable." For example, using more MPU registers to increase the number of compartments will likely lead to greater security at the expense of runtime and resource use. An important question to answer is how does the mechanism scale? For example, is there a break-down point where small increases in security come with large performance penalties? Similarly, how portable is the mechanism? Does it rely on specialized hardware not present on other embedded systems? Answers to these questions are essential for evaluating defense mechanisms. We can study this tradeoff by varying the resource description in the emulator, described in Task I.2.

5.2 IoT Benchmarks

Workloads for IoT or other embedded devices look very different from workloads for desktop, server, and mobile applications. As a result, existing benchmarks for these domains do not adequately measure IoT systems. For example, the well-known SPEC CPU suite of benchmarks are focused on "measuring and comparing compute intensive performances" [26]. In particular, SPEC CPU is concerned with the performance of a single task consisting of integer or floating point computations. An IoT device, by contrast, may spend most of its time in a low-power mode waiting for an event such as a timer firing or receiving input from a sensor or network. Once the event occurs, the device switches into a higher-power mode and executes a short task, often involving interaction with the physical world by means of attached peripherals, and then returning to the low-power mode.

Requirements. Benchmarks for IoT devices must meet several criteria. First, the applications must be realistic and mimic the application characteristics discussed above. While an individual benchmark need not satisfy *all* characteristics, the set of benchmarks in a suite must cover all characteristics. This ensures security and performance concerns with real applications are also present in the benchmarks. IoT devices are diverse, therefore the benchmarks should also be diverse and cover a range of factors, such as code complexity, types of peripherals used, and being built with or without an OS. Finally, network interactions must be included in the benchmarks.

Second, benchmarks must facilitate repeatable measurements. For IoT applications, the incorporation of peripherals, dependence on physical environment, and external communication make this a challenging criterion to meet. For example, if an application waits for a sensed value to exceed a threshold before sending a communication, the time for one cycle of the application will be highly variable. Similarly, the network characteristics tend to be quite variable and can affect the timing measurements. The IoT devices benchmarks must be designed to both allow external interactions while enabling repeatable measurements.

A third criterion is the measurement of a variety of metrics relevant to IoT applications. These include performance metrics (e.g. total runtime cycles), resource usage metrics (local resources like memory and stable storage, and energy resources), and domain-specific metrics (e.g. fraction of the cycle time the device spends in low-power sleep mode). An important goal of our effort is to enable benchmarking of IoT security solutions and hence the benchmarks must enable measurement of security properties of interest. There are of course several security metrics very specific to the defense mechanism but many measures of general interest can also be identified, such as the fraction of execution cycles with elevated privilege ("root mode") and number of Return-Oriented Programming (ROP) gadgets.

5.3 BenchIoT: Our Contribution

We have developed BenchIoT, a benchmark suite and evaluation framework that fulfills all the above criteria for evaluating IoT devices [1]. Our benchmark suite comprises of five realistic benchmarks, which stress one or more of the three fundamental task characteristics of IoT applications: sense, process, and actuate. They also have the characteristics of IoT applications introduced above. The BenchIoT benchmarks enable deterministic execution of external events and utilize network send and receive. BenchIoT targets 32-bit IoT devices implemented using the popular ARMv7-M architecture. Each BenchIoT benchmark is developed in C/C + + and compiles both for bare-metal IoT devices (i.e. without an OS), and for the ARM Mbed Operating System (Mbed-OS). Our use of the Mbed API (which is orthogonal to the Mbed-OS) enables realistic development of the benchmarks since it comes with important features for IoT devices such an embedded file system.

BenchIoT enables repeatable experiments while including sensor and actuator interactions. It uses a software-based approach to trigger such events. The software-based approach enables us to precisely control when and how the event is delivered to the rest of the software. This approach has been used in the past in embedded systems for achieving repeatability as a means to automated debugging [27, 28]. We can also control for the exact content of the events, which again enables the goal of repeatability of external events such as sensors and actuators without relying on the physical environment.

BenchIoT's evaluation framework enables automatic collection of 14 metrics covering four categories: (1) Security; (2) Performance; (3) Resource usage, and (4) Energy consumption Fig. 5. The evaluation framework is a combination of a runtime library and automated scripts. It is extensible to include additional metrics to fit the use of the developer and can be ported to other applications that use the ARMv7-M architecture. An overview of BenchIoT and the evaluation framework is shown in Fig. 3. The workflow of running any benchmark in BenchIoT is as follows:

(1) The user compiles and statically links the benchmark with a runtime library, which we refer to as the *metric collector library*, to enable collecting the dynamic metrics ❶; (2) The user provides the desired configurations for the evaluation (e.g. number of repetitions, timing of the interrupts, sensor values to use) ❷; (3) To begin the evaluation, the user starts the script that automates the process of running the benchmarks to collect both the dynamic ❸ and static ❹ metrics; (4) Finally, the benchmark script produces a result file for each benchmark with all its measurements ❺.

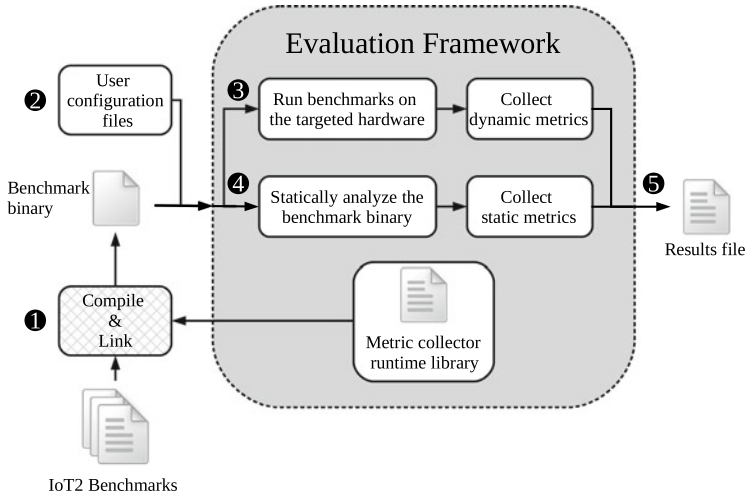


Fig. 3 An overview of the evaluation workflow in BenchIoT. BenchIoT provides five realistic IoT benchmarks spanning one or more of the key functionalities of sense, process, and actuate. BenchIoT measures four types of metrics: security, performance, resource usage, and energy consumption

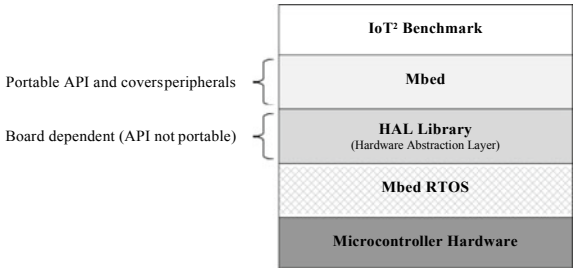


Fig. 4 Illustration of software layers and APIs used in developing BenchIoT benchmarks. BenchIoT provides portable benchmarks by relying on the Mbed platform

To implement the benchmarks and demonstrate rich and complex IoT devices applications, BenchIoT targets 32-bit IoT devices using the ARM Cortex-M (3, 4, 7) μCs, which are based on the ARMv7-M architecture. ARM Cortex-M is the most popular μC for 32-bit μCs with over 70% market share. This enables the benchmarks to be directly applicable to many IoT devices being built today. As shown in Fig. 4, hardware vendors use different HAL APIs depending on the underlying board. Since ARM supplies an ARM Mbed API for the various hardware boards, we rely on that for portability of BenchIoT to all ARMv7-M boards. In addition, for applications requiring an OS, we couple those with Mbed's integrated RTOS—which is referred to as Mbed-OS. Mbed-OS allows additional functionality such as scheduling, and network stack management. To target other μCs, we will have to find a corresponding common layer or build one ourselves—the latter is a significant engineering task and open research challenge due to the underlying differences between architectures.

Benchmark Applications

Table 1 shows the list of BenchIoT benchmarks with the task type and peripherals it is intended to stress. While the bare-metal benchmarks perform the same functionality, their internal implementation is different as they lack OS features and use a different TCP/IP stack. For the bare-metal applications, the TCP/TP stack operates in polling mode and uses a different code base. As a result the runtime of bare-metal and OS benchmarks are different.

6 Conclusion

It is upon us to significantly and promptly improve the security for bare-metal embedded and IoT systems. This has become imperative as they form the fabric, sometime hidden, of many critical systems, ranging from industrial control systems, public use equipment (like elevators and escalators), autonomous transportation facilities, personal IoT devices (smart devices and home assistants), to the innards of high-end computing equipment (like disk drives) or mobile equipment (like baseband processors on mobile phones). A high-level direction that we and other members of the community are pursuing is to restrict the privileges and capabilities of different

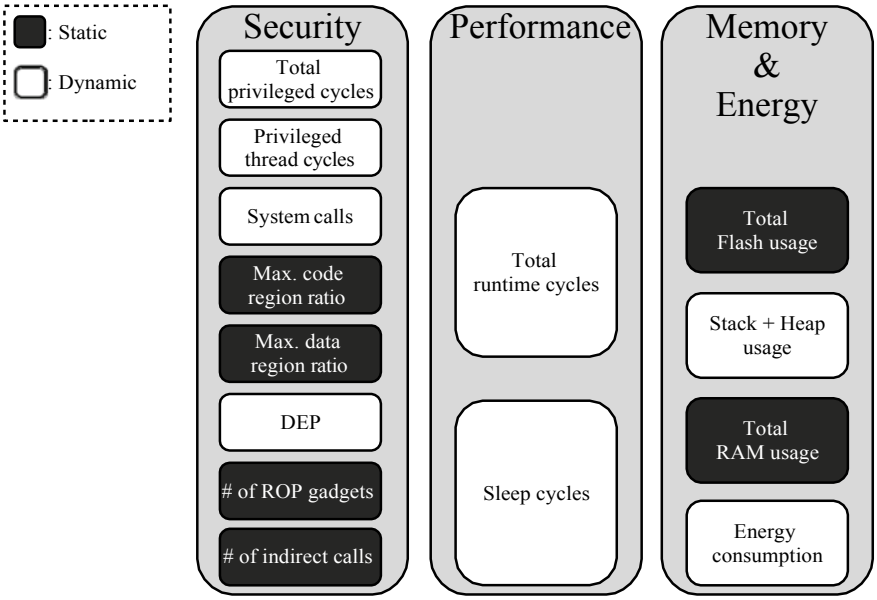


Fig. 5 A summary of the BenchIoT metrics. A white box indicates a dynamic metric, and a black box indicate a static metric

Table 1 A summary of BenchIoT benchmarks and their categorization with respect to task type, and peripherals

Benchmark	Task type			Peripherals
	Sense	Process	Actuate	
Smart light	✓	✓	✓	Low-power timer, GPIO, real-time clock
Smart thermostat	✓	✓	✓	Analog-to-digital converter (ADC), GPIO, μSD card Smart
Lock		✓	✓	Serial (UART/USART), display, μSD card, real-time clock
Firmware updater		✓	✓	Flash in-application programming
Connected display		✓	✓	Display, μSD card

regions of the application to the lowest necessary to perform intended operations. This is ideally done without needing application modification and with limited user annotations, to indicate what denotes security-critical operations, thus easing the application of the solution to legacy embedded applications. In this article, we have identified three interactive thrusts to achieve this solution:

(i) New static and dynamic analyses to identify security and functionality characteristics of each part of the application; **(ii) New runtime techniques** that enforce

the desired security properties while minimizing the performance impact; and **(iii) New security metrics and benchmarks** that accurately measure the security and performance impacts of defense mechanisms for embedded systems.

References

1. Almakhdhub NS, Clements AA, Payer M, Saurabh Bagchi B (2019) A security benchmark for the internet of things. In: 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 234–246
2. ARM (2017) mbed IoT platform. https://www.mbed.com/en/platform/
3. ARM Inc. Mbed uVisor, 2017
4. Banerjee SS, Jha S, Cyriac J, Kalbarczyk ZT, Iyer RK (2018) Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, pp 586–597
5. Barr Michael, Massa A (2006) Programming embedded systems: with C and GNU development tools. O'Reilly Media, Inc.
6. Biswas P, Di Federico A, Carr SA, Rajasekaran P, Volckaert S, Na Y, Franz M, Payer M (2017) Venerable Variadic Vulnerabilities Vanquished. In: SEC: USENIX security symposium
7. Burow N, Carr SA, Nash J, Larsen P, Franz M, Brunthaler S, Payer M (2017) Control-flow integrity: precision, security, and performance. ACM Comput Surv 50(1)
8. Cao PM, Wu Y, Banerjee SS, Azoff J, Withers A, Kalbarczyk ZT, Iyer RK (2019) CAUDIT : continuous auditing of SSH servers to mitigate brute-force attacks. In: 16th USENIX symposium on networked systems design and implementation (NSDI 19), pp 667–682
9. Carlini N, Barresi A, Payer M, Wagner D, Gross TR (2015) Control-flow bending: on the effectiveness of control-flow integrity. In: SEC: USENIX security symposium
10. Chung K, Kalbarczyk ZT, Iyer RK (2019) Availability attacks on computing systems through alteration of environmental control: smart malware approach. In: Proceedings of the 10th ACM/IEEE international conference on cyber-physical systems, pp 1–12
11. Chung K, Li X, Tang P, Zhu Z, Kalbarczyk ZT, Iyer RK, Kesavadas T (2019) Smart malware that uses leaked control data of robotic applications: the case of raven-ii surgical robots. In: 22nd International symposium on research in attacks, intrusions and defenses (RAID 2019), pp 337–351
12. Clements AA, Almakhdub NS, Saab K, Srivastava P, Koo J, Bagchi S, Payer M (2017) Protecting bare-metal embedded systems with privilege overlays. In: IEEE symposium on security and privacy (Oakland), pp 289–303
13. Esiner E, Mashima D, Chen B, Kalbarczyk Z, Nicol D (2019) F-pro: a fast and flexible provenance-aware message authentication scheme for smart grid. In: 2019 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm). IEEE, pp 1–7
14. Ge X, Talele N, Payer M, Jaeger T (2016) Fine-grained control-flow integrity for kernel software. In: EuroSP: IEEE European symposium on security and privacy
15. Haller I, Jeon Y, Peng H, Payer M, Bos H, Giuffrida C, van der Kouwe E (2016) Type sanitizer: practical type confusion detection. In: CCS: ACM conference on computer and communication security
16. Jeon Y, Biswas P, Carr SA, Lee B, Payer M (2017) HexType: efficient detection of type confusion errors for C++. In: CCS: ACM conference on computer and communication security
17. Jha S, Cui S, Banerjee S, Cyriac J, Tsai T, Kalbarczyk Z, Iyer RK (2020) Ml-driven malware that targets av safety. In: 2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, pp 113–124

18. Kuzentsov V, Payer M, Szekeres L, Candea G, Song D, Sekar R, Code pointer integrity. In: OSDI: symp. on operating systems design and implementation, vol 214
19. Lou X, Tran C, Tan R, Yau DKY, Kalbarczyk ZT (2019) Assessing and mitigating impact of time delay attack: a case study for power grid frequency control. In: Proceedings of the 10th ACM/IEEE international conference on cyber-physical systems, pp 207–216
20. Midi D, Payer M, Bertino E (2017) Memory safety for embedded devices with nes check. In: AsiaCCS: ACM symposium on information, computer and communications security
21. Necula GC, Condit J, Harren M, McPeak S, Weimer W (2005) CCured: type-safe retrofitting of legacy code. Trans. Prog. Lang. Syst. 27(3):477–526
22. Payer M, Barresi A, Gross TR (2015) Fine-grained control-flow integrity through binary hardening. In: DIMVA: conference on detection of intrusions and malware and vulnerability assessment
23. Rudd R, Skowyra R, Bigelow D, Dedhia V, Hobson T, Crane S, Liebchen C, Larsen P, Davi L, Franz M, Sadeghi A-R, Okhravi H (2017) Address-oblivious code reuse: on the effectiveness of leakage-resilient diversity. In: Proc Netw Distribut Syst Secur Symp (NDSS 17), pp 1–15, 2017
24. Salwan J (2011) ROPgadget—Gadgets finder and auto-roper
25. Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi A-R, Holz T (2015) Counterfeit object-oriented programming: on the difficulty of preventing code reuse attacks in C++ applications. In: Security and privacy (SP), 2015 IEEE symposium on. IEEE, pp 745–762
26. Standard Performance Evaluation Corporation (2017) SPEC CPU®2017. Online: https://www.spec.org/cpu2017/
27. Tancreti M, Hossain MS, Bagchi S, Raghunathan V (2011) Aveksha: a hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems. In: Proceedings of the 9th ACM conference on embedded networked sensor systems, pp 288–301
28. Tancreti M, Sundaram V, Bagchi S, Eugster P (2015) Tardis: software-only system-level record and replay in wireless sensor networks. In: Proceedings of the 14th international conference on information processing in sensor networks (IPSN). ACM, pp 286–297
29. Tice C, Roeder T, Collingbourne P, Checkoway S, Erlingsson Ú, Lozano L, Pike G (2014) Enforcing forward-edge control-flow integrity in GCC & LLVM. In: Fu K (ed) Proceedings of USENIX Security 2014. USENIX
30. UBM Canon (2015) 2015 embedded markets study
31. Zhang M, Sekar R (2013) Control flow integrity for COTS binaries. In: Proceedings of USENIX security 2013. USENIX
32. Zhu DY, Jung J, Song D, Kohno T, Wetherall D (2011) Tainteraser: protecting sensitive data leaks using application-level taint tracking. ACM SIGOPS Oper Syst Rev 45(1):142–154

Large-Scale Systems and Data Analytics

Introduction: Large-Scale Systems and Data Analytics

Saurabh Bagchi

Dependability has become a critical requisite property of computer systems as society begins to rely on computer systems in ever more ways and in ever expanding domains. Whether it is in conducting our business or in leading our personal lives, we rely on computer systems delivering correct results in a timely manner, even while they execute on a wide variety of hardware platforms and are written in a babel of programming languages. And more often than not, computer systems meet our expectations—large data centers serve up content at blinding speeds, stock trading happens correctly at millisecond latencies, processing on large genomics databases correctly unearth our predispositions to different medical conditions, and our favorite online movie portal goes down only rarely. However, the progress can be made even faster if researchers working on dependability challenges can be exposed to problems through quantitative data. Theories in the labs and small demonstrations in prototypes can be transitioned to the demanding realities of large computer systems if they could validate their inventions with real system usage and failure data. Unfortunately, even though our field has reached a certain stage of maturity, there is an astonishing lack of such publicly available data for researchers.

A comparison may fruitfully be drawn to the wide use of benchmarks and reference data sets in performance analysis of computer systems, such as, those put out by SPEC (Standard Performance Evaluation Corporation) or TPC (Transaction Processing Performance Council). However, the systems and the dependability communities have been sorely lacking in open datasets that can be used to motivate compelling research problems and to validate developed solutions. The five chapters in this section address various aspects of what are the reliability weakspots of large-scale systems and how collection and analytics of data from these systems can be used to mitigate the weakspots.

S. Bagchi (✉)
Purdue University, West Lafayette, Indiana, USA
e-mail: sbagchi@purdue.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_8

1 Rise of Data Analytics for System Dependability

The rise of data analytics has taken the technology world by storm and has brought about tectonic shifts in the technology landscape. The field of system dependability has also benefited from these advances in data analytics. The benefits have been seen in a multitude of ways. A course I teach, and available on a MOOC platform, titled "Big data for reliability and security" explores these developments in detail. In brief, the topics within system dependability that have seen significant impact from data analytics include:

1. *Identification of interactions among system components.* Most large-scale systems are composed out of many components. To understand the reliability weak spots, it is a necessary step to understand (at some level) the interactions among these components. Techniques from topics like causality theory have been used to make advances in this theme.
2. *Predicting failures.* Some component or system failures are predictable, such as, due to memory or other resource leaks. Data analytics techniques have fruitfully been used to build models that can predict if and when such failures will happen. Mitigation actions can then be initiated, either manually or through software means.
3. *Identifying root causes of failures.* This has been a topic of inquiry and advancements for several decades—how to automatically identify the root cause of a failure, which can then lead to replacement or other recovery mechanism. Due to the huge volumes of interactions among the components and complexity of these interactions, it is not possible to manually analyze them. Hence data analytics techniques have been brought to bear on this problem. This has seen some success, albeit in tightly specified computing systems.
4. *Regularization and sanitization of data.* An almost universal feature of data from large-scale systems is that it is noisy and incomplete. In order to extract insights from such data that will improve the dependability, we use big data techniques. This includes processing like regularization, feature engineering, dimensionality reduction, etc.
5. *Data analytics for security.* Some of the earliest compelling applications of big data techniques were security problems. These included applications such as spam detection and credit card fraud detection. Many newer applications of security have led to rapid advancements in data analytics techniques. These include detecting security attacks against data in motion (such as, when data is being exchanged between an IoT device and a server), ensuring the privacy of data on our personal devices (such as, wearables), and security of cross-organizational interactions (such as, service level contracts between multiple corporations).

2 Preview of Articles in This Section

This section brings together a diverse mix of contributions, all with the unifying theme of dealing with large-scale systems and some form of data analytics for dependability.

The article by **Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi** looks at the reliability of Non-volatile Compute-in-Memory (nvCiM) DNN accelerators. Such accelerators offer a great opportunity to break the memory wall by reducing data movement. Emerging NV devices offer greater energy efficiency and memory density than MOSFETs. However the calculation is noisy due to lack of precision of analog-to-digital conversion and device to device variation. This article focuses on design efforts to mitigate these problems in crossbar-based nvCiM DNN accelerators.

The article by **Long Wang** provides a systematic overview of security compliance techniques in academia and industry. It first introduces the life-cycle of critical computing systems from a compliance perspective, and then provides a reference architecture for compliance validation or enforcement. It then introduces the four stages of security compliance, and finally surveys the techniques in each of these four stages.

The article by **Karthik Pattabiraman** is a personalized and insightful reflection of the author on his time in Ravi's research group as a PhD student. In this article, Karthik traces the development of the hugely influential Trusted Illiac machine. This was a 256 node Linux cluster with machines providing customized checking for reliability and security to applications, and was inaugurated in 2013 at the University of Illinois. A reader, even one who is not interested in the details of the Trusted Illiac experience, will find of value the lessons that Karthik draws for successful research projects.

The article by **Marcello Cinque, Domenico Cotroneo, and Antony Pecchia** gives an authoritative view of the use of system logs for analyzing production failures, i.e., failures in systems that are in operation. The authors have done seminal work on this topic and draw on that rich experience to systematize the knowledge in the field. The systematization includes how to use logging APIs and data collection protocols, how to infer failure data from the logs, what are some high-value applications of log analysis, and finally the shortcomings of the topic. The shortcomings may well lead to productive lines of work in future years.

The final article in this section is by **Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano**. This presents a comprehensive description of the state-of-the-art in SIEM (Security Information and Event Monitoring) systems. Combining security information management (SIM) and security event management (SEM), SIEM systems offer real-time monitoring and analysis of events as well as tracking and logging of security data for compliance or auditing purposes. They are an important product category in corporate IT spend. This article focuses on SIEM for critical infrastructure protection and presents an architecture for *dependable* security monitoring. It then gives a practical instantiation of this architecture in three diverse and compelling application areas.

Overall, I believe that this section will help the reader understand where we have arrived in the topic of large-scale system dependability and the role of data analytics in it. It will also help the reader appreciate the foci of the ongoing research activity on this topic, and there is significant amount of such activity. Importantly, the chapters lay out the problems that ongoing research and development is tackling, in the near to mid term.

On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks

Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi

Abstract Computing-in-memory with emerging non-volatile memory (nvCiM) is shown to be a promising candidate for accelerating deep neural networks (DNNs) with high energy efficiency. However, most non-volatile memory (NVM) devices suffer from reliability issues, resulting in a difference between actual data involved in the nvCiM computation and the weight value trained in the data center. Thus, models actually deployed on nvCiM platforms achieve lower accuracy than their counterparts trained on the conventional hardware (e.g., GPUs). In this chapter, we first offer a brief introduction to the opportunities and challenges of nvCiM DNN accelerators and then show the properties of different types of NVM devices. We then introduce the general architecture of nvCiM DNN accelerators. After that, we discuss the source of unreliability and how to efficiently model their impact. Finally, we introduce representative works that mitigate the impact of device variations.

Keywords Compute-in-memory (CIM) · Device variations · Deep neural networks (DNN)

1 Introduction

Deep Neural Networks (DNNs) have excelled human performance in various crucial tasks (e.g., image classification, object detection, and speech recognition) and have become a popular solution for them. Thus, edge devices such as automobiles, smartphones, and smart sensors that depend on these tasks are ideal platforms to be empowered by DNNs. However, due to the constrained computation resource and limited

Z. Yan · X. S. Hu · Y. Shi (✉)
University of Notre Dame, Notre Dame, IN, USA
e-mail: yshi4@nd.edu

Z. Yan
e-mail: zyan2@nd.edu

X. S. Hu
e-mail: shu@nd.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_9

power budget of edge devices, direct implementation of computational intensive DNNs on edge devices is a significant challenge.

A majority of the works addressing this challenge use application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) for DNN acceleration. These conventional special-purpose edge DNN accelerators typically use a group of on-chip process elements (PEs) to handle computation and utilize off-chip non-volatile (NV) storage (e.g., flash) to store the model information (i.e., DNN architecture and model weights) [8]. Between the static random-access memory (SRAM) in the PEs used for temporary data caching and off-chip non-volatile storage used for power-off data preservation, there is also a complex memory hierarchy, generally consisting of several levels of dynamic random access memory (DRAM)-based on-chip memories. Because of this separation of data and computation, which is a key limitation of the conventional von-Neumann architecture, these DNN edge accelerators face energy efficiency and computing latency challenges. Specifically, PEs of this kind of architecture generates a large volume of intermediate data. These intermediate data need to be moved between different levels of the memory hierarchy so that they can be used by different process elements. Data movements across different levels of memory hierarchy induce a great time and energy consumption overhead, especially when accessing the lower level of the memory hierarchy. This challenge is also called *the memory wall*.

Non-volatile Computing-in-Memory (nvCiM) DNN accelerators [21] offer a great opportunity to break the memory wall by utilizing their special architectural advantages. nvCiM architectures reduce data movement with an in-situ weight data access scheme [42]. Emerging NVM devices (e.g., RRAMs, STT-RAMS, and FeFETs) are utilized so that nvCiM platforms can achieve higher energy efficiency and memory density compared with traditional MOSFET [39] based designs. More specifically, nvCiM can achieve low latency and high energy efficiency because, (1) the CiM structure avoids the long latency for moving data across multi-level memory hierarchies to retrieve the intermediate data and/or DNN weights; (2) analog computing engine performs dot-product in a compact manner, thus reducing the amount of intermediate data (i.e., partial sums) generated in multiply-and-accumulate (MAC) operations; (3) the crossbar structured matrix-vector multiplication (VMM) engine offers high parallelism that can perform VMM in one CiM cycle, thus shortening the latency of DNN operations.

However, such accelerators suffer greatly from design limitations. Firstly, because of emerging NVM devices tend to have low precision (1–4 bits) of (i.e., single NVM device can only represent 1 to 4-bit data and more than one device is needed to represent data in higher precision) and the limit of chip area, weight precision of neural networks mapped to nvCiM accelerators is limited. Secondly, most nvCiM accelerators require digital-to-analog converters (DACs) to convert the digital input data to analog signals so that it can be processed in crossbars and also analog-to-digital converters (ADCs) to convert the computation results back to digital signal for other neural network operations (e.g., activation and normalization). The precision of intermediate activation data is limited by the precision of DACs and ADCs. Thirdly, NVM devices, ADCs, and DACs suffer from device-to-device variations

due to manufacture and programming defects and cycle-to-cycle variations due to the computational environment difference. Compared with their digital counterparts that can tolerate such noises, because of the analog nature of nvCiM accelerators, the calculations performed in these platforms are not noise-free. The noisy nature of the computations leads to performance degradation that (1) models deployed on nvCiMs typically gets lower accuracy than their ideal counterparts trained in the data centers and (2) developers will not be able to know the exact accuracy of a model before it is deployed on a certain copy of the nvCiM product.

This reliability issue and its impact on DNN performance have been studied from different levels of design, including behavioral level explorations [15, 50], architecture level analysis [49], and device-level observations [56]. Cross-layer co-design efforts that simultaneously explore DNN model and hardware design pairs that can together achieve both high perception task performance and desirable hardware reliability are the current direction of this field [23].

In this chapter, we focus on design efforts targeting crossbar-based nvCiM DNN accelerators. We first introduce three typical emerging NVM devices including resistive random access memory (RRAM), ferroelectric field-effect transistor (FeFET), and Spintronics (STT) Devices. We then describe typical nvCiM DNN accelerator designs, their key components, and their benefits. After that, we discuss the limitations of nvCiM DNN accelerators and some key findings for these limitations. Finally, we introduce methods proposed to address the unreliability issue of nvCiM DNN accelerators from three aspects, encoding, DNN model training, and DNN architecture selection.

2 Non-volatile Devices

2.1 RRAM

Resistive random access memory (RRAM) is a two-terminal device that can be programmed into different levels of resistance value by using programming voltages in different magnitudes and duration.

As shown in Fig. 1a, the major component of RRAMs is a metal-insulator-metal (MIM) stack, where a dielectric layer is stacked in the middle of two electrode layers. When provided a programming voltage, a filamentary path, also called conductive filament (CF) [19], is created by soft electrical breakdown or forming in the electrode layers. In this filamentary path, a large concentration of defects, e.g., oxygen vacancies in metal oxides [4] or metallic ions injected from the electrodes [32], are then driven by field-induced migration and diffusion. Application of a positive voltage to the top electrode, where the defects are concentrated, induces defect migration towards the bottom electrode, thus causing the transition to the low-resistance state (LRS), because conduction is enhanced at defect sites. Application of a negative voltage, to the contrary, induces defect migration back to the top electrode, thus

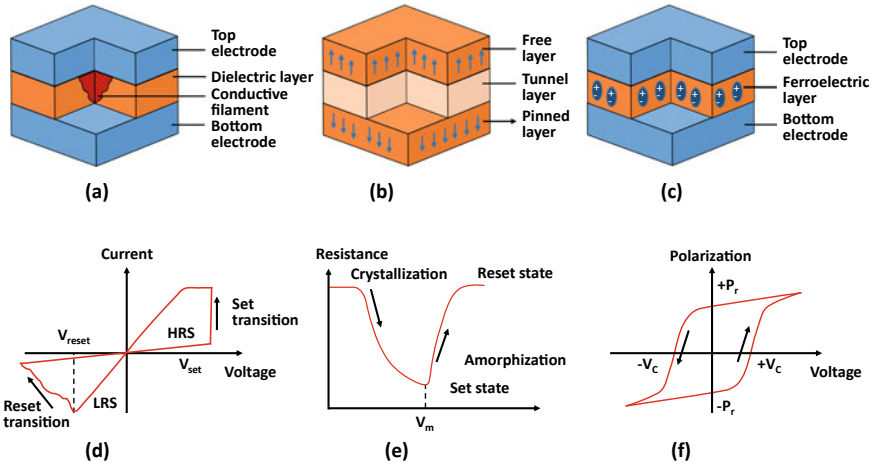


Fig. 1 Illustrations for key structures of different emerging NVM devices and their characteristics [8]. **a** RRAMs, **b** MRAMs and **c** FeFETs. The actual devices are more complex then these illustrations. **d**–**f** Are their characteristics, respectively

causing the transition to the high-resistance state (HRS) due to the disconnection of the CF. These transitions can be seen in the idealized current–voltage (I–V) characteristic in Fig. 1d, where the transition to the LRS (set operation) and the transition to the HRS (reset operation) occur at opposite voltages. Similar to the bipolar RRAM concept shown in Fig. 1d, unipolar RRAMs have also been presented, where the set and reset processes both occur under the same voltage polarity because of the dominant role of Joule heating in creating and dissolving the CF [24, 51]. All of these devices rely on the diffusion and migration of defects and will be referred to as RRAM throughout this chapter.

RRAM is a promising technology for in-memory computing thanks to the key features discussed below. First, its resistance ratio between HRS and LRS (on/off ratio) is generally greater than ten, which allows a clear distinction between digital '0' and '1'. This feature can be further exploited by dividing this gap between HRS and LRS in a non-binary manner, i.e., into multiple levels, resulting in a multi-level device that can represent multiple bits of data. This helps RRAM to offer a high-density storage scheme. Secondly, RRAM can operate at a moderately high switching speed (typically below 100 ns and some devices can achieve even in the sub-ns regime [11, 34]). Thus, RRAMs can operate in platforms with high clock speeds. Finally, RRAM is more durable compared to conventional flash storage devices [28]. This makes training DNNs on RRAM-based platforms possible.

2.2 Spintronics Devices

The spintronics devices are two or three-terminal devices equipped with a magnetic tunnel junction (MTJ) that stores information using magnetization direction of its recording layer and utilizes tunnel magnetoresistance (TMR) effect for reading where the resistance of the MTJ changes according to the stored information. In this section, we introduce the two-terminal version of this device, named spin-transfer torque (STT) device. When used as a programmable memory, this kind of device is also called Spin-transfer torque magnetic RAM (STT-MRAM).

Figure 1b shows a magnetic tunnel junction (MTJ), which is the major building block for most Spintronics Devices. The MTJ consists of a MIM structure where two ferromagnetic metal layers are divided by a thin tunnel oxide. An example of ferromagnetic metal materials used in MTJs is the CoFeB alloy, and an example material for the tunnel oxide is MgO. For the two ferromagnetic layers, one is referred to as the pinned layer and the other as the free layer. The magnetic polarization of the pinned layer is structurally fixed so that it can act as a reference point. On the other hand, the magnetic polarization of the free layer can be modified by a programming procedure.

Depending on the state of the free layer, the two ferromagnetic polarization can thus be either to the same direction (parallel) or to the opposite direction (antiparallel). Parallel polarization of the two layers puts the device into a low resistance state (LRS), and antiparallel means a high resistance state (HRS) due to the tunnel magnetoresistive effect [7]. Researchers are working on finding efficient ways to flip the state of the MTJ and the spin-transfer torque (STT) is one of the newer and more competitive candidates to offer a scalable and low-efficient flip [33]. In the STT procedure, transition to the parallel state takes place directly by conduction electrons, which are first spin-polarized by the pinned layer, then rotate the magnetic polarization of the free layer by magnetic momentum conservation [41]. Similarly, the free layer magnetization can be rotated to the antiparallel state by applying an opposite voltage (hence opposite current direction). The relative difference in resistance of the LRS and HRS, also called the magnetoresistance ratio when referring to spintronics devices, is typically around 200% [53]. STT-based devices are also fast, with a switching speed typically lower than 1 ns, and durable, with an endurance above 10^{14} [6].

In STT devices, STT induced magnetization switching [5, 41] is used to store data in to the device (write process). Its primitive cell has one cell transistor and one MTJ (1T1MTJ), which can achieve a relatively small cell size of ideally $6F^2$, where F is the feature size of the MTJ layer. The write current passes through the tunnel barrier, as is also the case with the read current. Accordingly, the read current should be small enough so that the write event, i.e., magnetization switching, does not take place, and the write current should be small enough that it does not give rise to a barrier breakdown.

2.3 FeFET

A ferroelectric transistor (FeFET) is a three-terminal device equipped with a layer of ferroelectric (FE) material. It can either be configured to a steep switching mode to serve as an efficient FET or a non-volatile (NV) mode to serve as a programmable switch.

The structure of a FeFET is similar to a regular bulk MOSFET or FinFET, except that in its gate stack, there is an additional layer of ferroelectric (FE) material. Besides this FE material, a metal layer between the FE and dielectric may or may not be included [2]. Designs of FE transistor structures with [29] and without [40] this layer both demonstrate state-of-the-art efficiencies. It is worth noting that although some FE materials (e.g., hafnium zirconium oxide (HZO)) are both efficient and highly compatible with CMOS processes and can thus be realized on the industrial scale, other FE materials (e.g., lead zirconium titanate (PZT) [3]) may be incompatible with CMOS processes.

As discussed above, FeFETs can operate in two different modes: an NV mode or a steep switching mode. Basic structures of FeFETs in these two modes are the same, except that in different configurations (e.g., material thickness, gate length, and width), the relative capacitance of the FE material and the underlying FET changes, resulting in different modes of operation. In this chapter, we discuss the properties of FeFETs in the NV mode because FeFETs used in nvCiM DNN accelerators are majorly in this mode.

NV mode of FeFETs are discovered later than its steep switching counterpart at the emergence of HZO-based FeFETs [31]. The non-volatile property results from the hysteretic polarization (P) versus voltage of the FE material (V_{FE}) shown in Fig. 1f. When the FE material is placed in series with the gate of a transistor, the hysteretic window of P versus V_{GS} is reduced because the MOS structure of the FET and the associated depolarization fields imposes a capacitance and the total capacitance between gate and source changes [44]. Nevertheless, a sufficiently thick FE broadens the hysteretic window so that the hysteretic behavior is preserved and can be observed in the $I_D - V_{GS}$ transfer characteristics of this device Fig. 1f. This corresponds to the non-volatile, hysteretic mode of FeFETs. In this mode of operation, at $V_{GS} = 0V$ (i.e., when the supply voltage is turned off), the FeFET exhibits two stable states which correspond to positive or negative polarization retention in the FE layer. For an n-type FeFET, the device exhibits high resistance states (HRS) when $P < 0$ and low resistance states (LRS) when $P > 0$. For a p-type FeFET, it is in HRS when $P > 0$ and LRS when $P < 0$. Thus, when the FE layer is sufficiently thick, non-volatility can be embedded inside a transistor, i.e., FeFET can operate as an NV memory and a transistor switch at the same time.

3 CiM DNN Accelerators

3.1 Computing-in-Memory

Conventional von-Neumann architecture is not efficient because the cost of data movements between memory and processing units is high. This issue is called *the memory wall*. More seriously, the technologies for logic units are growing faster than memory cells, causing a significant gap between computation and memory access. Thus, various efforts have been made to break *the memory wall* by moving the computations closer to memory. The integration of memory and computation is an evolving concept and is developing along with technological advances [16]. We first introduce an earlier concept which is now considered near memory computing (NMC). Researchers embed processing cores into dynamic random-access memory (DRAM) modules [12, 35, 37] so that data can be processed in the DRAM module. This avoids sending data from DRAM to CPUs across the complex memory hierarchy. However, integrating DRAMs and processing units on the same chip is not beneficial if the communication cost between memory and processing units is not reduced. The concept of 3D stacking is adopted to address this issue. By stacking multiple silicons on top of each other and utilizing through-silicon-vias (TSVs) to handle inter-silicon layer communications, 3D stacking allows the processing unit to be integrated as additional layers of the stacked chip and can provide higher bandwidth compared with putting memory and logic in different chips [13, 18, 55]. However, these methods do not actually use memory modules for data processing and are still sending data from memory to logic.

A step further from NMC is computing-in-memory (CiM), where processing is directly performed inside the memory array. The latency and energy efficiency requirements of edge devices greatly inspired researches in this field. The integration of processing and memory units can be done in different levels of granularity. The extremest design of CiM is that each of the memory cells is able to perform logic operations [26]. This is referred to as fine-grained CiM. There is also a spectrum of designs between fine-grained CiM and NMC. A typical design is to empower memory arrays (of SRAM or DRAM) with processing abilities so that data can be processed inside operations inside and between memory arrays. This can be achieved by modifying the peripheral circuitry of these memory arrays. This approach is referred to as coarse-grained CiM.

The CiM concept is further evolved with the help of new advances in emerging NVM device technologies. Specifically, NVM devices including RRAMs, STT-MRAMs, and FeFET-based RAMs can offer high density, good scalability, and high power efficiency. Thus, these devices are natural replacements for SRAMs or DRAMs in CiM architectures. Various recent efforts utilize CiM-capable NVM devices instead of SRAMs or DRAMs as building blocks of either cache or main memory. One direction of research is to use NVM simultaneously as storage and logic devices by re-designing sense amplifiers so that NVM arrays can perform a subset of logic and arithmetic operations [22, 30, 38]. Another direction is to use

NVMs to build content-addressable memories (CAMs). CAMs can perform searches in a parallel manner, thus reducing the search time significantly. Moreover, search in CAMs requires little data movement, which leads to low energy consumption. The third direction is to use NVM devices to build DNN accelerators. These accelerators can directly execute matrix-vector multiplication inside the memory array. This saves the cost of data movements. The advances of NVM-based CiM DNN accelerators are discussed in detail in the following sections.

3.2 *Crossbar-Based Vector-Matrix Multiplication Engine*

Crossbar array is the key component of nvCiM DNN accelerators. As shown in Fig. 2, a crossbar array can be considered as a processing element for matrix-vector multiplication where matrix value (i.e., weights for DNNs) are stored at the cross point of each vertical and horizontal line with resistive NVM devices such as RRAMs and FeFETs, and each vector value is propagated through horizontal data lines. In this work, we mainly introduce an RRAM-based design. Designs using other kinds of NVM devices are with similar structures. The calculation in crossbar array is performed in the analog domain but additional peripheral digital circuits are needed for other key DNN operations (e.g., non-linear activation and pooling), so DAC and ADCs are adopted between different components.

As is demonstrated in Fig. 2, every bitline (vertical) is connected to every word-line (horizontal) via NVM cells [39]. Assume that the cells in the first column are

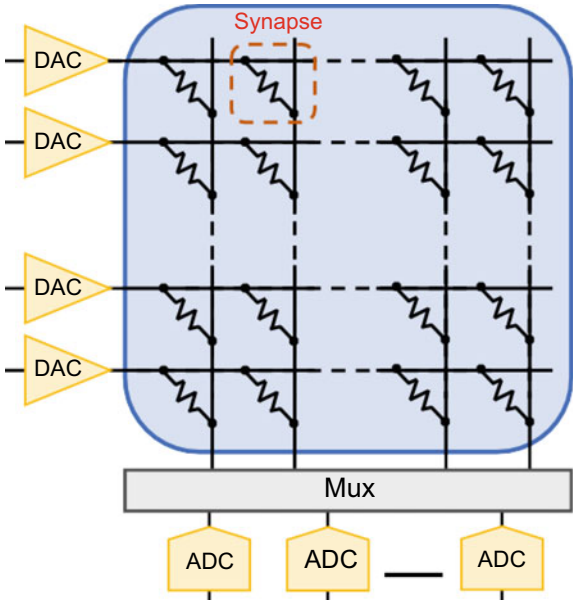


Fig. 2 Illustration of crossbar array architecture. The input is fed horizontally and multiplied by weights stored in the NVM devices at each cross point. The multiplication results are summed up vertically and the sum serves as an output

programmed to resistances $r_1, r_2, \ldots, r_n$, where n is the number of rows. The conductances of these cells, $g_1, g_2, \ldots, g_n$, are the inverses of their resistances ($g_i = 1/r_i$). If voltages $V_1, V_2, \ldots, V_n$ are applied to each row 2, cell i generates current V_i/R_i, which is equivalent to $V_i \times g_i$, into the bitline, based on Kirchoff's Law. The total current accumulated on the bitline is the sum of currents passed by each cell in the column, i.e., $I = \sum_{i=1}^{n} V_i \times g_i$. This current I represents the value of a dot product operation, where one vector is the set of input voltages at each row $\mathbf{V}$ and the second vector is the set of cell conductances $\mathbf{g}$ in a column, i.e., $I = \mathbf{V} \cdot \mathbf{g}$.

As shown in Fig. 2 $\mathbf{V}$ is applied to all columns in parallel. The currents emerging from each bitline can therefore represent multiple vector-vector dot product, which is then a vector-matrix multiplication. VMM is the key operation of DNNs. In a fully connected layer, for example, there are multiple neurons and each neuron is fed with the same input vector, but each of the neurons has a different set of synaptic weights. This operation can be represented by $\mathbf{O} = \mathbf{V}\,\mathbf{G}$ where $\mathbf{V}$ is the input, $\mathbf{G}$ is the weight matrix for neurons and $\mathbf{O}$ is its output. The crossbar array shown in Fig. 2 represents an $n \times m$ crossbar array that performs dot products on n-entry input vectors for m different outputs in a single CiM cycle.

Note that the result of the VMM operation would also need to be applied a bias value and passed through a non-linear activation function. This is done off the crossbar array. Thus, peripheral circuits are needed to perform these operations. Moreover, crossbar arrays handle VMM operations in the analog domain while other peripheral circuits are digital. DACs and ADCs are needed to transform data to and from the analogy domain. Generally, for each row of the crossbar array, there is a dedicated DAC to serve this wordline. However, ADCs are large in area and power-hungry. Thus, multiple bitlines need to share one ADC and this is achieved by the sense-and-hold circuits along with the MUX selector.

3.3 General Architecture of nvCiM DNN Accelerators

Various accelerator architectures have been proposed to utilize the nvCiM crossbar arrays for more efficient DNN acceleration. There are generally two fashions of acceleration, one only accelerates the inference path of DNN models and the other also considers DNN training acceleration. In this chapter, we focus on DNN inference acceleration and we introduce two well-known architecture level designs, ISAAC [39] and PRIME [10] for this scheme.

The first design, In-Situ Analog Arithmetic in Crossbars (ISAAC) [39] uses crossbar arrays for both DNN weight storage and processing elements for VMM operations [54]. As shown in Fig. 3, ISAAC is implemented with a hierarchical-structured architecture whose major component is "tile". Each tile consists of multiple in-situ MAC units (IMA), eDRAM buffers, and key DNN circuitries including shift-and-add (SA), sigmoid, and max-pooling units. Thus, a tile can perform DNN operations individually. Each IMA unit is equipped with a few crossbar arrays and ADCs connected by a shared bus. Different from traditional SRAM-based designs, writing NVM devices

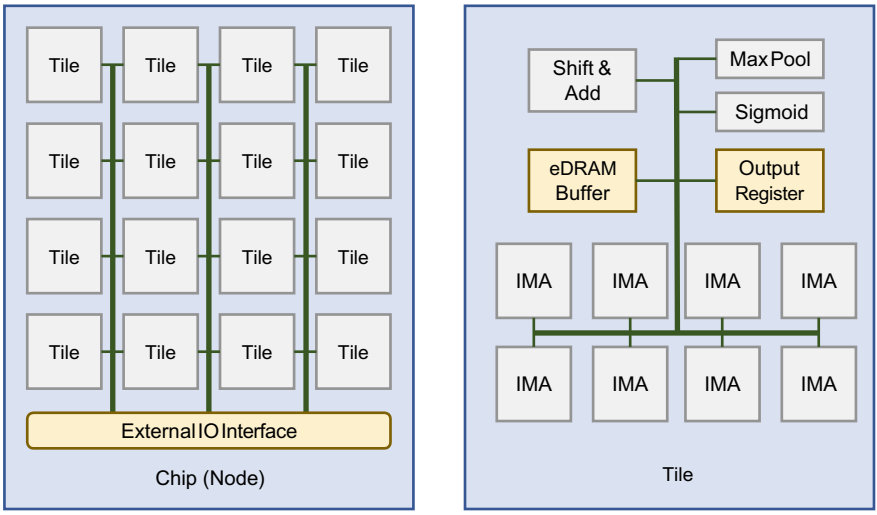


Fig. 3 Illustration of ISAAC architecture. ISSAC is composed of a group of tiles and each tile consists of multiple crossbar-based IMAs, buffers, and peripheral circuits for other key DNN operations

is expensive (both in terms of time and energy consumption), so re-configuring crossbars in runtime are not feasible and thus crossbar arrays cannot be reused and each array is dedicated to only one CNN layer. The outputs of a former layer are temporarily preserved in the eDRAM buffer so that they can be used as the input of the next layer. Note that, except for the structure inside a "tile", the architecture of ISAAC is very similar to its digital DNN accelerator counterpart DaDianNao [9], which is a state-of-the-art architecture when ISAAC is proposed. After tape out, the researchers show that, with a 16-chip configuration, ISAAC achieves 14.8× higher throughput while consuming 5.5× lower energy than DaDianNao. This means (1) ISAAC can achieve higher energy efficiency than state-of-the-art and (2) crossbar array-based design is a key contributor to this efficient design.

Different from ISAAC that never re-configures NVMs, the PRIME architecture [10] uses a scheme where a portion of the NVM arrays can alternate between storage and compute units during runtime. As shown in Fig. 4, the authors modify the standard wordline decoder and drivers (WDD), column multiplexers, and sense amplifiers so that they can better suit the RRAM-based crossbar arrays, and configure the storage banks into three different function units, memory subarrays (MS), full-function subarrays (FFS), and buffer subarrays (BS). The FFS is the key component that can alternate from memory to computational units. In the computation mode, FFS can perform VMM for DNNs, and in the storage mode, FFS buffers the intermediate data generated by VMMs. Similarly, the BS also acts as storage when FFS is not in computation mode. The sense amplifier is reconfigured to detect the higher precision analog value for computation compared to storage requirements so that matrix multiplication can be performed. The modified column multiplexer executed

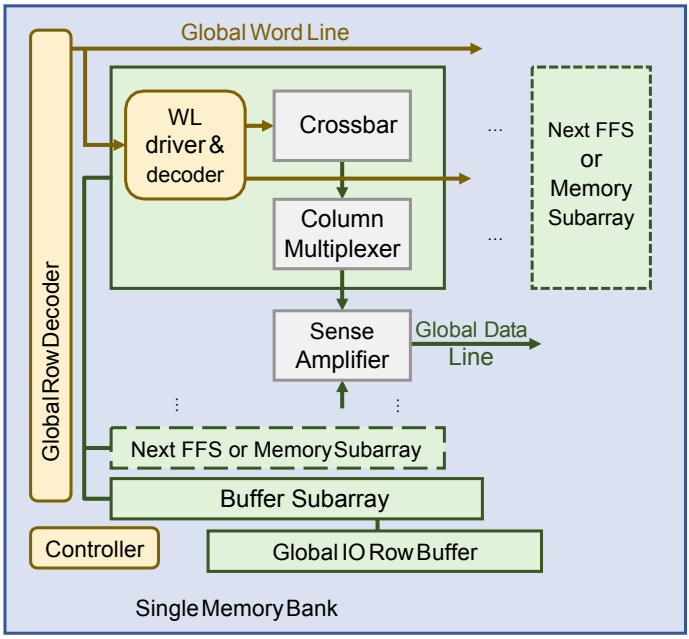


Fig. 4 Illustration of the prime memory bank. Each FFS can operate in two modes, one is computation mode for MAC operation of DNNs, and the other is storage mode buffering and data preservation

analog substractions and nonlinear threshold functions. Although their implementation exerted a 60% area overhead, the computation energy was saved by 94% by reducing external memory accesses.

4 Device and Circuit Non-idealities

Although nvCiM can offer low latency and high energy efficiency, there are two major limitations of nvCiM, low data precision, and low device reliability. For the first issue, due to the limitation of the area and power budget, both the weight stored in the NVM devices and intermediate activation data can not be represented in a high precision manner. nvCiM DNN accelerators generally use data representations of four to eight bits [10, 39]. This problem is similar to the quantization problem of the traditional digital DNN accelerators and has been sufficiently discussed [17, 45]. However, the origin, simulation method, and mitigation approach of the reliability issue of nvCiM DNN accelerators are still open questions and are still receiving heated discussions. In this section, we introduce the origin of the reliability issue of nvCiM DNN accelerators with an example of RRAM devices. For STT and FeFET

devices, the source of unreliability is similar but the significance and specific behavior of these noise sources are slightly different.

Various research about developing fault models for RRAM and other emerging NVM devices has been established. In this section, we focus on five noise sources that are directly related to the unreliability of nvCiM DNN accelerators: thermal noise, shot noise, random telegraph noise (RTN), programming errors, and endurance failures [14].

4.1 Thermal Noise

Thermal noise is also known as Johnson-Nyquist noise. It is electronic noise caused by the thermal agitation of carriers and is a property of all passive devices. It happens regardless of whether a voltage is applied to the device. A well-established model for thermal noise is by placing a current source in parallel with the ideal target device. The current source is also known as the noise current and its magnitude is modeled by a Gaussian distribution with zero mean and a standard deviation of $\sqrt{\frac{4K_B T \Delta f}{R}}$, where K_B is the Boltzmann constant ($\approx 1.38 \times 10^{-23}$ J/K), T is the temperature in Kelvins, δf is the bandwidth of the signal measured, and R is the resistance of the ideal target device. Thermal noise is a fundamental property of resistive circuit elements. From the model, we can observe that the only way to reduce thermal noise is to reduce the device temperature. To handle this source of noise, noise resilient architectures that can operate under thermal noise need to be devised.

4.2 Shot Noise

Shot noise is also a fundamental source of noise caused by the physical nature of electronic devices. This source of noise is called Poisson noise because it can be modeled by a Poisson process. The key cause of shot noise is the discrete nature of currents where electric currents actually consist of flows of discrete charges (e.g., electrons). When the number of electrons flowing through the device at a certain point of time fluctuates, a fluctuation of current through a device can be observed. This can affect the measurement accuracy when a detector is sensing the current flowing into it. Although shot noise is easy to be averaged out provided enough measurement time, devices working in high frequencies (e.g., nvCiM DNN accelerators) still suffer from such noise. As discussed above, a Poisson process is a more precise way of modeling shot noise, but this noise model is too complex when embedded in other models. A simpler model is a zero-mean Gaussian noise with a standard deviation of $\sqrt{2qI\Delta f}$, where q is the charge of an electron ($\approx 1.6 \times 10^{-19}$ C), I is the current flowing through the ideal target device, and Δf is the bandwidth of the signal measured.

4.3 Random Telegraph Noise

Random telegraph noise (RTN) exists in both CMOS and emerging NVM device circuits but is considered as a major cause of faults of emerging NVM devices [20]. RTN is also called burst noise and is caused by the charge carriers that are temporarily trapped inside the device, thus changing the effective resistance of the device. The result is a temporary and unexpected reduction in the resistance of a device at runtime. The trapping and untrapping of the charge carrier is modeled mathematically by means of the telegraph process, which is a Markovian continuous-time stochastic process that jumps discontinuously between two distinct values.

4.4 Programming Errors

Programming errors refer to the difference between the actual device resistance and the target resistance due to the non-ideal configuration of the device. This is generally caused by both the process variations and temporal variations of each device. Affected by the former noise, when applied the programming voltage of the same magnitude and duration, the resistance of different instances of emerging NVM devices can be different. The latter leads to the fact even when applied to identical programming pulses, an NVM device can be programmed to different values in different trials of programming. A complex but effective way to mitigate this issue is to use a scheme called write-and-verify [1, 36, 47]. The key operation is to iteratively apply a series of short pulses (write) and then check the difference between current and target resistance (verify), converging progressively on the target resistance. In deploying accelerators for Neural Network inference, this time-consuming progress is tolerable because once programmed, no more modifications to the resistance are needed during the entire life span of the accelerator. This scheme pulls down the programming error to less than 1%. This 1% of error can be modeled by a zero-mean Gaussian noise where the standard deviation is determined by the error upper bound of the write-and-verify process.

4.5 Endurance and Retention

Endurance Failure is about the device being able to preserve their property after multiple times of write operations or and retention is about being able to read the desired data at a long period of time after programming. The endurance of emerging NVM devices varies widely based on the material properties and write mechanisms. The typical endurance for CMOS-based SRAM is 10^{16} which means typically, after this amount of write, the device would be stuck at a certain value, and writing it would be infeasible. The typical endurance for STT-MRAM is 10^{15}, for FeFET is

10^5 and for RRAM is 10^7 [14]. On the other hand, being able to read out the correct information when it is a long period of time after the device is programmed is also an important subject. This is called the retention issue. For simple CiM implementations like Memristive Boltzmann Machine, a typical worst-case lifetime is 1.5 years, but for nvCiM DNN accelerators, the system is more complex and the lifetime is shorter. To mitigate the effect of the endurance issue, researchers proposed a fault-tolerant online training method [46] that maps the weight matrices stored in crossbars for computation around faults or endurance failures through a combination of neural network pruning and data remapping. This scheme increases the life of the neural network accelerator, allowing it to be used for training.

5 Impact of Device Variation on DNN Acceleration

5.1 Model of Device Variation

The source of device variations and their behaviors are introduced in Sect. 4, but modeling such device characteristics is not a simple task. A straightforward way is to abstract the behavior of different devices into circuit-level models [56] and utilize circuit-level simulation tools (e.g., *SPICE*) to investigate the behavior of certain nvCiM DNN accelerators. However, because of the complexity of both neural network typologies and DNN accelerator architectures, building circuit-level models for nvCiM accelerators requires great human effort and needs to be modified each time a new type of accelerator architecture is proposed. Moreover, circuit-level models are computationally intensive. Using such models to simulate complex DNN accelerators requires considerable evaluation time and is not suitable during design phase explorations. Thus, a simple and effective model for the impact of device variations is needed.

One of the effective modeling methods is to model the device variation as a whole and use a Gaussian distribution to represent it [14, 15, 23, 50]. Here we introduce one representative modeling method [15] using Gaussian variables.

The NVM device electrical property, e.g., conductance, is subject to the combined effect of different variation sources as in Sect. 4. The actual conductance values g considering variations on n devices of a crossbar array can be written as:

$$\mathbf{g} = g_{0,n\times 1} + \Delta g_g + F\left(g_{0,n\times 1,\mathbf{r}} \approx \overline{g_{0,n\times 1}} + f\left(g_{0,n\times 1}, \mathbf{r}\right)\right) \tag{1}$$

where $\overline{g_{0,n\times 1}} = g_{0,n\times 1} + \Delta g_g$ with $g_{0,n\times 1}$ denoting the expected conductance and Δg_g denoting the global conductance variation as a constant for all the devices on the same die; $\mathbf{r}$ models the underlying spatially correlated and dynamic variations; $f(g_{0,n\times 1}, \mathbf{r})$ is a function describing the dependence of variations on the expected conductance and can be approximated by $f(g_{0,n\times 1}, \mathbf{r})$ due to the relatively small value of variations w.r.t. the nominal values [11].

Since the mapped weights $\mathbf{w}$ are linearly related to conductance as $\mathbf{w} = c_1 \times \mathbf{g} + c_0$, where c_1 and c_0 are two constants, each weight w_i represented by multiple devices can be modelled as a Gaussian variable:

$$w_i = \mathcal{N}\left(u_{0,i}, \Psi\left(u_{0,i}\right)^2\right) \tag{2}$$

$$= \mathcal{N}\left(c_1 \overline{g_{0,i}}, +c_0, c_1^2 f\left(\overline{g_{0,i}}\right)^2 \left(\sum_{k=1}^{m} \lambda_{i,k}^2 + \lambda_{i,n}^2\right)\right) \tag{3}$$

5.2 Impact of Device Variation on DNN Outputs

After finishing modeling the device variations, we can then investigate the impact of device variations on nvCiM DNN accelerators. A typical study is to evaluate such impact on an accelerator targeting image classification tasks [49]. In this section, we introduce the findings of the authors of [49].

A starting point is understanding the effect of device variations on the output of a DNN model. The forward path of a DNN model can be viewed as a function of the input and the weight value of the model. Formally speaking, a DNN inference process can be defined as:

$$\mathbf{O} = F(W, \mathbf{I}) \tag{4}$$

where F is the DNN architecture, W is the DNN weights, $\mathbf{I}$ is the input vector, and $\mathbf{O}$ is the output vector.

In classification tasks, the output vector $\mathbf{O}$ for each input (not batched) is a 1-D vector whose size is the number of possible classes. Each element of this vector represents the model's confidence that the input images should be classified into a certain class. Thus, the class with maximum value in $\mathbf{O}$ is what the model predicts to be the best choice for classification. During training, $\mathbf{O}$ is passed through a *Softmax* function so that the confidence for each class is between 0 and 1 and the sum of confidences among different classes is 1. However, *Softmax* is not necessary during DNN inference because it does not change the order of the values in $\mathbf{O}$. The final predicted class of $\mathbf{I}$ is calculated by *argmax* ($\mathbf{O}$), which is the index of the item in $\mathbf{O}$ that has the maximum value. As we focus on inference, the vanilla version of $\mathbf{O}$ before *Softmax* is the key.

Taking device variation into account, a model deployed on nvCiM DNN accelerators can be represented as:

$$\mathbf{O}_{Dep} = F(W_{Dep}, I) = F(\mathcal{N}(W_{Exp}, \sigma), I) \tag{5}$$

where W_{Dep} is the weight actually deployed on the accelerator and according to Eq. 2, it can be modeled as a Gaussian variable whose mean is W_{Exp}, which is the trained value of the neural network to be deployed, and the standard deviation is σ, which can be calculated using Eq. 2. $\mathbf{O}_{Dep}$ is the affected output.

One indicator of the effect of device variations on nvCiM accelerators is the difference in output. Formally speaking, we can define *output change* as the difference between the output without device variation and the output value under the impact of device variation:

$$\mathbf{O}_{Change} = F(W_{Exp}, I) - F(\mathcal{N}(W_{Exp}, \sigma), I) \tag{6}$$

Note that $\mathbf{O}_{Change}$ is also a random variable.

In order to get a glance at the statistical behavior of $\mathbf{O}_{Change}$, according to the workflow introduced in Sect. 5.2, the authors train a LeNet model for the MNIST dataset [27] to state-of-the-art accuracy. The authors then randomly choose one input image in the test dataset and sampled 10k different instances of noise. With this setup, the authors gathered 10k different $\mathbf{O}_{Change}$ vectors.

For MNIST, $\mathbf{O}_{Change}$ is a vector of 10, with each element representing the confidence of classifying the input image into one certain number digit. Because a high-dimensional vector is not a good choice for analytical study and visualization, each element of these vectors is visualized independently, so 10 instances of distribution data are collected.

Each element of $\mathbf{O}_{Change}$ follows Gaussian distribution. To visualize this finding, the authors plot the histogram of the distribution of each element of $\mathbf{O}_{Change}$ vector and the corresponding Gaussian distribution that fits it. The visualization result for the first element of $\mathbf{O}_{Change}$ is shown in Fig. 5. It is obvious that the visualized variable is Gaussian.

This observation generalizes in various networks in various datasets. For the MNIST dataset, three models are analyzed: (1) LeNet and two-layer-multilayer perceptrons (2-layer-MLP) using (2) ReLU and (3) Sigmoid activation. For the

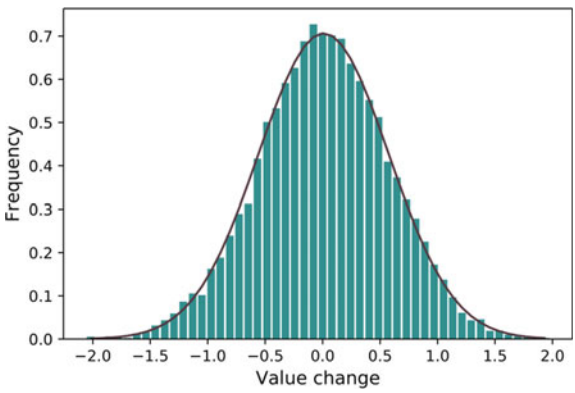


Fig. 5 $\mathbf{O}_{Change}$ distribution of LeNet for MNIST. 10k $\mathbf{O}_{Change}$ vectors are gathered from one trained LeNet model affected by 10k different instances of weight values from $\sigma = 0.04$. This figure shows the distribution of the first item of the gathered $\mathbf{O}_{Change}$ vectors. It is obvious that the visualized variable is Gaussian

CIFAR-10 dataset [25], the authors of [49] test four models: (1) a conventional floating-point CNN, (2) a quantized CNN, and two ResNets, (3) ResNet-56 and (4) ResNet-110. For each model, three different initializations are used to train three different sets of weights.

The authors of [49] collect all $\mathbf{O}_{Change}$ variables and find the closest Gaussian variable that fits each of them. To measure the similarity of $\mathbf{O}_{Change}$ and its Gaussian counterpart, two widely used standards: mean square error (MSE) and Chi-square (χ^2) test are used. For variables with one element, MSE can be described as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(O_i - E_i)^2 \tag{7}$$

and χ^2 test can be depicted as:

$$\chi^2 = \sum_{i=1}^{N}\frac{(O_i - E_i)^2}{E_i} \tag{8}$$

where O_i and E_i are the observed ($\mathbf{O}_{Change}$) and estimated (Gaussian) value of, normalized in the form of probability density, and N is a user-defined granularity. Here $N = 100$ is used because it is precise enough when there is a total of 10k instances of $\mathbf{O}_{Change}$ data. The similarity of a vector is averaged out among all of its elements and the final similarity is also averaged out among all different initializations.

The similarity of $\mathbf{O}_{Change}$ distribution and its Gaussian fit for different models are shown in Table 1. For each model tested, the average χ^2 test results among different initializations are all below 0.1 and MSE are all below 10^{-3}, which indicates we can have high confidence that $\mathbf{O}_{Change}$ distribution is Gaussian. Moreover, this observation is scalable because, for both extremely shallow (e.g., 2-layer MLP) and very deep (RestNet-110) candidates, both errors do not increase. Thus this observation generalizes across different DNN models targeting classification tasks. With this

Table 1 The similarity of $\mathbf{O}_{Change}$ distribution and its Gaussian fit for different models

Model	Dataset	χ^2 (10^{-2})	MSE (10^{-4})
MLP-ReLU	MNIST	5.22	3.20
MLP-Sigmoid	MNIST	5.81	2.20
LeNet	MNIST	4.59	2.67
Float-Conv	CIFAR-10	7.01	3.03
Fixed-Conv	CIFAR-10	6.79	2.74
ResNet-56	CIFAR-10	4.56	1.79
ResNet-110	CIFAR-10	4.81	2.01

The χ^2 test result and MSE between the $\mathbf{O}_{Change}$ and its Gaussian fit counterpart is presented. Both tests show that the $\mathbf{O}_{Change}$ is a multi-dimensional Gaussian variable w.r.t. different instances of noise

conclusion, the authors of [49] claim that, **with any independent and identically distributed Gaussian noise on weight, the output vector of the same input image follows a multi-dimensional Gaussian distribution**[1] **over different samples of noise**.

This claim is very strong and there is only empirical support for it. However, it is not counter-intuitive. The output of the first convolution layer is the summation of the multiplication result of deterministic inputs and Gaussianly distributed weights and is thus a summation of Gaussian distributions. The summation of Gaussian variables is also a Gaussian variable, so the output of the first layer is a Gaussian variable. After activation, the input of the second layer is a transformed Gaussian variable. After propagating through this layer, each output value is the sum of multiple multiplication results, and operands for each multiplication are both Gaussian variables. It is also worth noticing that, for the same layer, the standard deviation σ for each noisy weight is the same. So the results of each multiplication are close to IID and with enough number of operands for this summation, the accumulated variable can be approximated by Gaussian variables. Thus, although the final output may not strictly be a Gaussian variable, a Gaussian approximation can be observed.

6 Dealing with Device Non-idealities

The majority of noise sources of nvCiM DNN accelerators are random noise that is difficult to eliminate during device production. Fortunately, there are opportunities from the accelerator architecture, DNN topology design, and DNN training aspects that can help to mitigate the effect of device variations. In this section, the authors introduce four different efforts from these three aspects.

6.1 Error Correction

As discussed in Sect. 3.3, nvCiM accelerators process DNN models in a layer by layer manner and devise nvCiM processing units that consist of crossbar arrays and other peripheral digital blocks to perform matrix-vector multiplication and other key DNN operations including non-linear activation and pooling. From the accelerator architecture design aspect, it is a straightforward idea to equip nvCiM platforms with error correction abilities so that they can mitigate the effect of device variations.

In this section, we introduce one representative work [14] that uses error correction code to assist nvCiM computation. The authors use a group of arithmetic codes, named AN-codes [43] for error correction. Arithmetic codes are a class of error correction codes (ECCs) that can preserve the result of arithmetic operations with noisy operands. AN-codes are a set of arithmetic codes that apply arithmetic weight

[1] Note that each element of the output are deeply co-related, not independent.

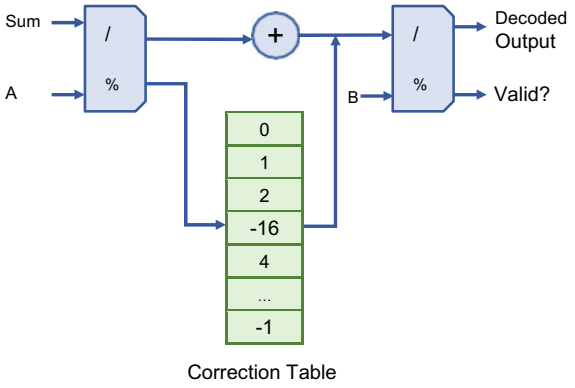


Fig. 6 Illustration of error correction unit circuitry. This is a lookup table styled design

to each operand so that it can maximize the arithmetic distance between codewords. An example of AN codes that utilizes residues is, for a given integer K and operands A and B, $KA + KB = K(A + B)$ and $(KA + KB)\%\ K \equiv 0$. The ECC units can detect and correct the error according to the residue.

The error correction unit (ECU) in [43] has three major components: two divide/residual units for the residual computation of A and B (one each), and a correction table that maps each residual to a syndrome. The output of the first divide/residual unit computes the integer division of the input by A and outputs the residual along with the quotient. The residual is used to index into the correction table, and the value read from the correction table is added to the result. This value is then fed into the second divide/residual unit where it is divided by B. The output of this unit is the final output of the error correction system and includes a flag indicating if the computation was in error. An illustration of ECU is shown in Fig. 6.

6.2 *Identifying Robust Neural Architectures*

Some DNN topologies (neural architectures) are more robust than others against device variations. Finding these neural architectures is a viable way of mitigating the effect of device variations. Meanwhile, different neural architectures require different amounts of computation power and are thus with different inference latency and power consumption. Handcrafting a neural architecture that meets all design requirements is a challenging task. Fortunately, neural architecture search (NAS) [48, 52, 57] is proposed to automatically find an optimal neural architecture in a designated design space using reinforcement learning-based algorithms.

In this section, we introduce NACIM [23], a device-circuit-architecture co-exploration framework that can automatically identify the best CiM neural accelerators from a design space including the device type, circuit topology, and neural architecture hyper-parameters. NACIM framework iteratively conducts explorations based on a reward function, which is suitable for reinforcement learning approaches

or evolutionary algorithms. By configuring the parameters of the framework, designers can customize the optimization goals in terms of their demands. The authors model the effect of device variation by modeling the shot noise as a stuck-at-low or stuck-at-high fault, and the other noise sources as a whole to be a zero-mean additional Gaussian noise extracted from widely adopted models [56] on the weight value. Experimental results show that the proposed NACIM framework can find the robust neural network with only 0.45% accuracy loss in the presence of device variation, compare with a 76.44% loss from the state-of-the-art NAS without considering device variation.

6.3 Training Robust DNNs

DNN models with the same neural architecture but different weights can have very similar accuracy in ideal conditions but very different accuracy in the existence of device variations. Thus, finding proper weights that are robust against device variations in the training process is a desirable approach.

A straightforward way to find robust weights is to simulate the noisy forward path in the training process, i.e., in each iteration of training, the algorithm sample an instance of noise and add it to the weight in the forward and backpropagation path to calculate the gradient, then remove the noise when updating the weights.

This method is used in NACIM [23] which is introduced before. For implementations in MNIST dataset [27], noise injection training can reduce the accuracy drop between the ideal model and model with device variations from 6 to 0.5%, and in CIFAR-10 dataset [25], noise injection training can reduce the accuracy drop from 76.44 to 0.45%.

A more advanced way to find robust weights is to seek help from Bayesian Neural Networks (BNN). Bayesian neural network is known for a stochastic gradient variational Bayes framework applied to approximate posterior distributions over network parameters. By employing a prior distribution over the weight space, BNN allows us to introduce variation to the learning process to better fit the observations [15].

A recent work [15] uses BNN to improve the robustness of nvCiM accelerators. BNN requires a priori distribution and uses an estimated posterior to fit this distribution. The priori can be obtained from device variation models. These models are inferred from expert knowledge with the help of measurement, simulation, and historical data. The authors also use KL divergence as the regularization term to enforce the memristor variation structural characteristics.

Although the priori used in most recent works are carefully designed, they can still be imprecise or uncertain because of the measurement imperfectness and the ever-going evolution of emerging devices. To address this issue, the authors of [15] propose a variance-adaptive priori to weigh the value of prior knowledge. The author modify the optimization objective of BNNs so that weights with larger values are more regularized by the priori, i.e., it allows placing heavier priorities on those critical weights (with higher magnitude) on crossbar arrays that are prone to receive

more impact from device variations, thereby reducing oscillations in convergence for more efficient training. Finally, to prevent the over-amplification of variation during training, the authors add an additional regularization term using L2 norm loss. In CIFAR-10 dataset, this proposed method is able to reduce the accuracy drop from 45.7 to 0.3%.

Although these two methods are effective in terms of mitigating the effect of device variations on nvCiM accelerators, they require much more training iterations to converge compared with traditional training methods. In the MNIST dataset, both methods require at least 10× more iterations of training to reach a similar accuracy as the traditional training method [23].

7 Conclusions

Computing-in-memory with emerging non-volatile devices (nvCiM) is a great candidate for efficient DNN acceleration because of its unique architecture that breaks the memory wall. However, it suffers from unreliability issues, especially the device variation issues of emerging NV devices. Understanding the property of emerging NV devices and the general architecture of nvCiM DNN accelerators helps to better model the effect of device unreliability circuit and application level. The modeling of unreliability also helps in mitigating the impact of device variations. The representative ways of mitigation include the adoption of ECC in the architecture and finding neural network topologies and training DNN weights that are more robust against device variations.

References

1. Alibart F, Gao L, Hoskins BD, Strukov DB (2012) High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. Nanotechnology 23(7):075201
2. Aziz A, Breyer ET, Chen A, Chen X, Datta S, Gupta SK, Hoffmann M, Hu XS, Ionescu A, Jerry M et al (2018) Computing with ferroelectric FETs: devices, models, systems, and applications. In: 2018 Design, automation & test in Europe conference & exhibition (DATE). IEEE, pp 1289–1298
3. Aziz A, Ghosh S, Datta S, Gupta SK (2016) Physics-based circuit-compatible spice model for ferroelectric transistors. IEEE Electron Device Lett 37(6):805–808
4. Beck A, Bednorz J, Gerber C, Rossel C, Widmer D (2000) Reproducible switching effect in thin oxide films for memory applications. Appl Phys Lett 77(1):139–141
5. Berger L (1996) Emission of spin waves by a magnetic multilayer traversed by a current. Phys Rev B 54(13):9353
6. Carboni R, Ambrogio S, Chen W, Siddik M, Harms J, Lyle A, Kula W, Sandhu G, Ielmini D (2016) Understanding cycling endurance in perpendicular spin-transfer torque (p-STT) magnetic memory. In: 2016 IEEE International electron devices meeting (IEDM). IEEE, pp 21–6
7. Chappert C, Fert A, Van Dau FN (2010) The emergence of spin electronics in data storage. Nanosci Technol Collect Rev Nat J 147–157

8. Chen WH, Dou C, Li KX, Lin WY, Li PY, Huang JH, Wang JH, Wei WC, Xue CX, Chiu YC et al (2019) CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. Nat Electron 2(9):420–428
9. Chen Y, Luo T, Liu S, Zhang S, He L, Wang J, Li L, Chen T, Xu Z, Sun N et al (2014) DaDianNao: a machine-learning supercomputer. In: 2014 47th Annual IEEE/ACM international symposium on microarchitecture. IEEE, pp 609–622
10. Chi P, Li S, Xu C, Zhang T, Zhao J, Liu Y, Wang Y, Xie Y (2016) PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. ACM SIGARCH Comput Architect News 44(3):27–39
11. Choi BJ, Torrezan AC, Strachan JP, Kotula P, Lohn A, Marinella MJ, Li Z, Williams RS, Yang JJ (2016) High-speed and low-energy nitride memristors. Adv Funct Mater 26(29):5290–5296
12. Draper J, Chame J, Hall M, Steele C, Barrett T, LaCoss J, Granacki J, Shin J, Chen C, Kang CW et al (2002) The architecture of the diva processing-in-memory chip. In: Proceedings of the 16th international conference on supercomputing, pp 14–25
13. Farmahini-Farahani A, Ahn JH, Morrow K, Kim NS (2015) NDA: near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In: 2015 IEEE 21st International symposium on high performance computer architecture (HPCA). IEEE, pp 283–295
14. Feinberg B, Wang S, Ipek E (2018) Making memristive neural network accelerators reliable. In: 2018 IEEE International symposium on high performance computer architecture (HPCA). IEEE, pp 52–65
15. Gao D, Huang Q, Zhang L, Yin X, Li B, Schlichtmann U, Zhuo C (2021) Bayesian inference based robust computing on memristor crossbar. In: 2021 56th ACM/IEEE Design automation conference (DAC). IEEE, pp 1–6
16. Gao D, Reis D, Hu XS, Zhuo C (2019) Eva-CiM: a system-level energy evaluation framework for computing-in-memory architectures. arXiv preprint arXiv:1901.09348
17. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149
18. Hsieh K, Khan S, Vijaykumar N, Chang KK, Boroumand A, Ghose S, Mutlu O (2016) Accelerating pointer chasing in 3d-stacked memory: challenges, mechanisms, evaluation. In: 2016 IEEE 34th International conference on computer design (ICCD). IEEE, pp 25–32
19. Ielmini D (2011) Modeling the universal set/reset characteristics of bipolar RRAM by field-and temperature-driven filament growth. IEEE Trans Electron Devices 58(12):4309–4317
20. Ielmini D, Nardi F, Cagli C (2010) Resistance-dependent amplitude of random telegraph-signal noise in resistive switching memories. Appl Phys Lett 96(5):053503
21. Ielmini D, Wong HSP (2018) In-memory computing with resistive switching devices. Nat Electron 1(6):333–343
22. Jain S, Ranjan A, Roy K, Raghunathan A (2017) Computing in memory with spin-transfer torque magnetic ram. IEEE Trans Very Large Scale Integr (VLSI) Syst 26(3):470–483
23. Jiang W, Lou Q, Yan Z, Yang L, Hu J, Hu XS, Shi Y (2020) Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. IEEE Trans Comput
24. Kim KM, Jeong DS, Hwang CS (2011) Nanofilamentary resistive switching in binary oxide system; a review on the present status and outlook. Nanotechnology 22(25):254002
25. Krizhevsky A et al (2009) Learning multiple layers of features from tiny images
26. Kvatinsky S, Belousov D, Liman S, Satat G, Wald N, Friedman EG, Kolodny A, Weiser UC (2014) Magic—memristor-aided logic. IEEE Trans Circ Syst II Express Briefs 61(11):895–899
27. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
28. Lee MJ, Lee CB, Lee D, Lee SR, Chang M, Hur JH, Kim YB, Kim CJ, Seo DH, Seo S et al (2011) A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures. Nat Mater 10(8):625–630
29. Li KS, Chen PG, Lai TY, Lin CH, Cheng CC, Chen CC, Wei YJ, Hou YF, Liao MH, Lee MH et al (2015) Sub-60 mV-swing negative-capacitance FinFET without hysteresis. In: 2015 IEEE International electron devices meeting (IEDM). IEEE, pp 22–6

30. Li S, Xu C, Zou Q, Zhao J, Lu Y, Xie Y (2016) Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: Proceedings of the 53rd annual design automation conference, pp 1–6
31. Li X, Sampson J, Khan A, Ma K, George S, Aziz A, Gupta SK, Salahuddin S, Chang MF, Datta S et al (2017) Enabling energy-efficient nonvolatile computing with negative capacitance FET. IEEE Trans Electron Devices 64(8):3452–3458
32. Liu Q, Sun J, Lv H, Long S, Yin K, Wan N, Li Y, Sun L, Liu M (2012) Real-time observation on dynamic growth/dissolution of conductive filaments in oxide-electrolyte-based ReRAM. Adv Mater 24(14):1844–1849
33. Locatelli N, Cros V, Grollier J (2014) Spin-torque building blocks. Nat Mater 13(1):11–20
34. Loke D, Lee T, Wang W, Shi L, Zhao R, Yeo Y, Chong T, Elliott S (2012) Breaking the speed limits of phase-change memory. Science 336(6088):1566–1569
35. Mai K, Paaske T, Jayasena N, Ho R., Dally WJ, Horowitz M (2000) Smart memories: a modular reconfigurable architecture. In: Proceedings of 27th international symposium on computer architecture (IEEE Cat. No. RS00201). IEEE, pp 161–171
36. Niu D, Xiao Y, Xie Y (2012) Low power memristor-based ReRAM design with error correcting code. In: 17th Asia and South Pacific design automation conference. IEEE, pp 79–84
37. Oskin M, Chong FT, Sherwood T (1998) Active pages: a computation model for intelligent memory. In: Proceedings of the 25th annual international symposium on computer architecture (Cat. No. 98CB36235). IEEE, pp 192–203
38. Reis D, Niemier M, Hu XS (2018) Computing in memory with FeFETs. In: Proceedings of the international symposium on low power electronics and design, pp 1–6
39. Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan JP, Hu M, Williams RS, Srikumar V (2016) Isaac: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. ACM SIGARCH Comput Architect News 44(3):14–26
40. Sharma P, Tapily K, Saha A, Zhang J, Shaughnessy A, Aziz A, Snider G, Gupta S, Clark R, Datta S (2017) Impact of total and partial dipole switching on the switching slope of gate-last negative capacitance FETs with ferroelectric hafnium zirconium oxide gate stack. In: 2017 Symposium on VLSI technology. IEEE, pp T154–T155
41. Slonczewski JC (1996) Current-driven excitation of magnetic multilayers. J Magn Magn Mater 159(1–2):L1–L7
42. Sze V, Chen YH, Yang TJ, Emer JS (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc IEEE 105(12):2295–2329
43. Van Lint J, van der Geer G (2012) Introduction to coding theory and algebraic geometry, vol 12. Birkhäuser
44. Wang D, George S, Aziz A, Datta S, Narayanan V, Gupta SK (2016) Ferroelectric transistor based non-volatile flip-flop. In: Proceedings of the 2016 international symposium on low power electronics and design, pp 10–15
45. Wang K, Liu Z, Lin Y, Lin J, Han S (2019) HAQ: hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 8612–8620
46. Xia L, Liu M, Ning X, Chakrabarty K, Wang Y (2017) Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In: Proceedings of the 54th annual design automation conference 2017, pp 1–6
47. Xu C, Niu D, Muralimanohar N, Jouppi NP, Xie Y (2013) Understanding the trade-offs in multi-level cell ReRAM memory design. In: 2013 50th ACM/EDAC/IEEE Design automation conference (DAC). IEEE, pp 1–6
48. Yan Z, Jiang W, Hu XS, Shi Y (2021) Radars: memory efficient reinforcement learning aided differentiable neural architecture search. arXiv preprint arXiv:2109.05691
49. Yan Z, Juan DC, Hu XS, Shi Y (2021) Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search. In: 2021 26th Asia and South Pacific design automation conference (ASP-DAC). IEEE, pp 859–864
50. Yan Z, Shi Y, Liao W, Hashimoto M, Zhou X, Zhuo C (2020) When single event upset meets deep neural networks: observations, explorations, and remedies. In: 2020 25th Asia and South Pacific design automation conference (ASP-DAC). IEEE, pp 163–168

51. Yang JJ, Strukov DB, Stewart DR (2013) Memristive devices for computing. Nat Nanotechnol 8(1):13–24
52. Yang L, Yan Z, Li M, Kwon H, Lai L, Krishna T, Chandra V, Jiang W, Shi Y (2020) Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In: 2020 57th ACM/IEEE Design automation conference (DAC). IEEE, pp 1–6
53. Yuasa S, Nagahama T, Fukushima A, Suzuki Y, Ando K (2004) Giant room-temperature magnetoresistance in single-crystal Fe/MgO/Fe magnetic tunnel junctions. Nat Mater 3(12):868–871
54. Zaman KS, Reaz MBI, Ali SHM, Bakar AAA, Chowdhury MEH (2021) Custom hardware architectures for deep learning on portable devices: a review. IEEE Trans Neural Networks Learn Syst
55. Zhang D, Jayasena N, Lyashevsky A, Greathouse JL, Xu L, Ignatowski M (2014) TOP-PIM: throughput-oriented programmable processing in memory. In: Proceedings of the 23rd international symposium on high-performance parallel and distributed computing, pp 85–98
56. Zhao M, Wu H, Gao B, Zhang Q, Wu W, Wang S, Xi Y, Wu D, Deng N, Yu S et al (2017) Investigation of statistical retention of filamentary analog RRAM for neuromorphic computing. In: 2017 IEEE International electron devices meeting (IEDM). IEEE, pp 39–4
57. Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. In: International conference on learning representations (ICLR)

Providing Compliance in Critical Computing Systems

Long Wang

Abstract Critical services such as health care, finance, power, public utility, are being hosted in critical computing systems like cloud, big data and AI platforms. Compliance is one of the main instruments governments exert on such computing systems for regulating security and reliability so that governments and the public can obtain the assurance that no severe unacceptable impacts or incidents may occur. This article gives a comprehensive discussion on the compliance problem in critical computing systems, and describes state-of-the-art technologies and practices of compliance validation/enforcement. This article is very helpful for those professionals working on critical computing systems and services.

1 Introduction

Cloud computing and artificial intelligence are growingly adopted for supporting services and applications, including those ones critical to human's living, survival, and even human's life, such as health care and medication, finance, education, power, public utility, telecommunication, etc. These critical services and applications are hosted on cloud systems, big data analytics systems, and traditional data centers, which are called critical computing systems in this chapter. These critical computing systems are demanded to be highly reliable/dependable and highly secure, as service unavailability, data corruption, information leaking, or security breach on the systems may result in severe impacts on human's living or human's life.

To regulate the behaviors of the computing systems and their hosted critical services, and particularly, to eliminate severe unacceptable impacts of system failures, security breaches or human errors onto hosted services, critical computing systems are usually required by governments, legislation, or industry standards, to comply with certain rules, which is called the security compliance of the computing systems. Though there is not a standard definition of security compliance, a widely accepted definition is given by Julisch [1]: "security compliance, in IT systems, is

L. Wang (✉)
Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
e-mail: longwang@tsinghua.edu.cn

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_10

the state of conformance with externally imposed functional security requirements and of providing evidence/assurance thereof." Intuitively, security compliance means the behavior of the computing system and/or the business process conforms to what is specified in the text of rules set forth by governments, legislation, or industry standards.

Compliance is one of the main instruments adopted in real-world practices for regulating security and reliability of critical computing systems, and is the major requirement for these systems to provide security assurance. This article presents the state of the art on providing compliance in critical computing systems.

2 Compliance in Critical Computing Systems

Compliance is an important property and requirement for computing systems that host critical services [2], such as cloud systems, big data analytics systems, and AI platforms. Compliance is the key instrument for governments and legislation to regulate the behavior of the computing systems with regards to dependability, security, privacy, auditability, etc. Violation of compliance usually leads to severe penalties onto the computing systems and/or their owners and operators, e.g. the computing services may be forced to get offline, or even the operator may get imprisoned [3].

Services hosted on critical computing systems span multiple domains like finance, health and medication, telecommunication, and so on. In each of these domains there are compliance rule sets (acts, regulations, directives, decisions, standards, guidelines, etc.), e.g. there are HIPAA [4] and GxP [5] in the health and medication domain, and there is Finance Service Compliance [6] in the finance domain. Different countries may have different compliance rule sets, e.g. GDPR (General Data Protection Regulations) [7] in Europe has the compliance rules on computing systems' manipulation of data, including health data, while HIPAA is the compliance on computing systems' manipulation of health data in United States.

Cloud systems are popular platforms now for hosting services and applications. So commercial cloud systems usually conform to a number of compliance rule sets. For example, Amazon Web Services (AWS) cloud observes tens of compliance rule sets including CSA, ISO9001, HIPAA, PHIPA, etc. [8]; IBM Cloud observes the compliance rule sets of CIS, CSA STAR, ISO9001, ISO20243, SOC2, etc. [9]; Microsoft Azure cloud and Google cloud also observe a number of compliance rule sets [10, 10].

2.1 High-Level Studies on Compliance

Before we describe detailed technologies of providing compliance in Sects. 3 and 4, here are a number of high-level studies on security compliance. Al-Aqrabi et al. analyzed the difficulties and challenges brought by security compliance [12], and listed compliance as one of the top three problems during the migration of services

and applications from proprietary systems to cloud platforms. Gudivada et al. [13] stated certain support at the computing system architecture required by compliance. Ragan [14] classified compliance into multiple levels, and described the compliance requirements at each level. In 2008 Julisch [1] pointed that the major focus of computing system security in industrial practice is compliance and the academia should pay more attention onto this problem. Yimam and Fernandez presented a survey on security compliance in the context of cloud computing in 2016 [15]. Von Solms differentiated two different types of compliance, one for information regulation and the other for business regulation [16]. Tianfield [17] and Hashizume et al. [18] also did similar analysis and reported compliance as one of the major challenges in cloud security. Ardagna et al. [19] reported the same conclusion in the context of big data analytics platforms.

There are also studies that focused on specific compliance acts or regulations. For example, Kibbe [20] describes the 10 high-level steps of achieving HIPAA compliance, Artnak and Benson [21] depict multiple aspects of HIPAA compliance, and the report [22] explains how Microsoft Corporation supports the Sarbanes–Oxley (SOX) compliance [23].

Besides the high-level discussion and presentation of security compliance above, Ullah et al. gave a reference framework for compliance in cloud systems [24], Mather et al. discussed certain standards of compliance in cloud systems [25], and the survey [15] also proposed an abstract reference architecture of compliance in Software-as-a-Service (SaaS) cloud systems. In addition, there are studies on specific compliance problems. Beautement et al. [26] and Kilbridge [27] studied the cost of a number of specific compliance standards; Kalaiprasath et al. [28] and Hendre and Joshi [29] discussed the compliance requirements in specific aspects including key management, privacy, and incident response. Furthermore, they discussed how to select the best cloud platforms when given requirements in these aspects. In particular, they built matrixes that list how (to what extent and in what details) different cloud platforms provide support to the compliance in these aspects. Then their algorithms search the matrixes when clients supplied certain compliance requirements in these aspects and find the best cloud platforms for the clients.

2.2 Compliance Rules

Here we use HIPAA as an example to illustrate what a compliance rule set typically looks like. HIPAA, or Health Insurance Portability and Accountability Act, was released in 1996 to protect the privacy and security of health and medical data in United States. Compliance rules are typically organized in multiple categories, as the HIPAA fragments in Fig. 1 show. The right snapshot in Fig. 1 shows specific compliance rules (e.g. “assign a unique name and/or number for identifying and tracking user identity”), and the other two snapshots show the categories of the compliance rules.

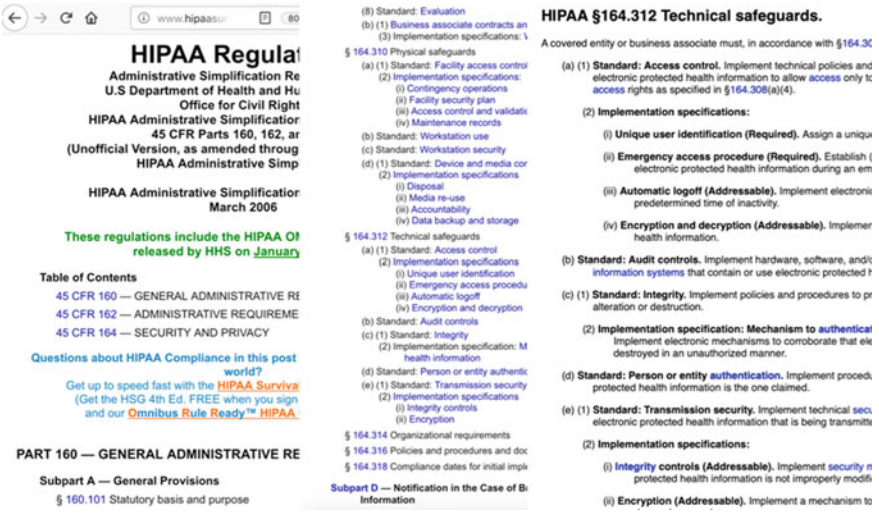


Fig. 1 Fragments of HIPAA compliance regulation rules [30]

As Fig. 1 shows, the HIPAA compliance rules have multiple categories: physical rules, technical rules, and administrative rules. Physical rules specify the controls of the facility/building access, workstation use, use of USB storage or other media, and so on. Technical rules specify the controls for the technical parts of the computing systems, e.g. what levels of cryptography should be used, and what mechanisms should be implemented for authentication and authorization. Administrative rules specify the controls on staff management, responsibility management, business operations, etc., e.g. the security official responsible for certain policies and procedures must be identified, and policies for authorizing access to Protected Health Information (PHI) must be defined and implemented.

Besides the physical, technical and administrative categories which are created from the nature of the rules, compliance rules can also be categorized in other perspectives, e.g. rules enforced during the development stage, the test stage and the operation stage, respectively.

2.3 Compliance Audit

Figure 2 illustrates the life cycle of critical computing systems from the compliance perspective. It has four phases: development, audit, release, and system operation/compliance enforcement. Agents from the government or a third-party company will do the audit against given compliance rules after the development completes. After the audit passes, the computing system is then released to serve real-world workloads, and the system and its hosted applications are in operation while the

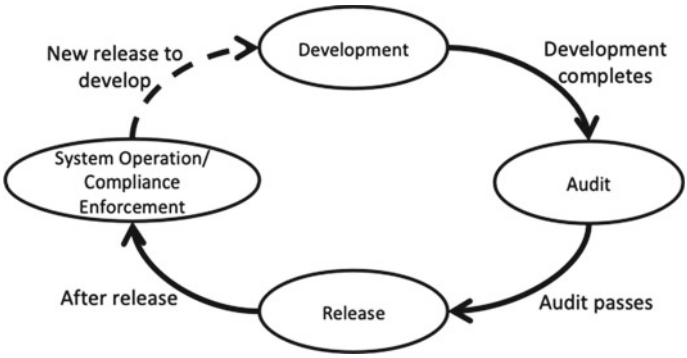


Fig. 2 Life cycle of critical computing systems from compliance perspective

compliance enforcement goes on. Note that this life cycle picture considers operation-stage compliance only, which is usually the main part of compliance rules for computing systems, and does not take development-stage or test-stage rules into account.

When a new release of the system is to be built, the new release starts its own cycle by entering the development phase while the current release continues to stay in the "system operation/compliance enforcement" phase. After the new release completes the audit and release procedures the new release replaces the current release and becomes in operation. The compliance rules are then checked against the new current release of the system in the "system operation/compliance enforcement" phase.

The common practice of audit now is based on a Quality Management System (QMS) approach as specified in the ISO9001:2015 standard [31]. The QMS is an approach that documents a system's implementation of a given compliance rule set in the form of, from top down to bottom, policies, designs, procedures, work instructions, and records and forms. The documents are reviewed, approved and signed by relevant experts and/or executives, and then are deposited into a QMS document management system. All the compliance-regulated behavior of the system should be based on the documents in the QMS system.

Currently the audit of a computing system is typically carried out as outlined in Fig. 3. The audit agents and the development staff work together during the audit to validate and verify if the system behavior conforms to those specified in the compliance rule text and the QMS documents. Particularly, for each compliance rule the audit agents ask the development staff for specific evidence that demonstrates the implementation of and the conformance to the rule according to the agents' understandings of the rule. Then the development staff collect the relevant execution data, logs, system specifications and QMS documents, and put them in a storage place called "system execution data" in Fig. 3. The audit agents look into the collected execution data, compare the execution data against relevant QMS documents, and validate that the execution data supply evidence that the system's execution conforms to the text of the compliance rule. For example, a compliance rule states that data

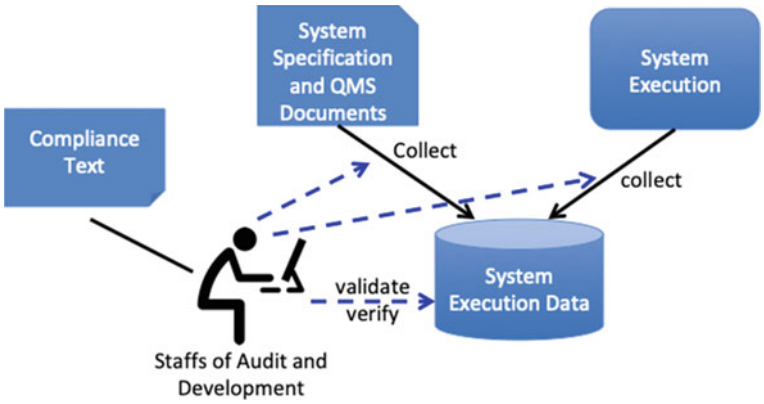


Fig. 3 Diagram of current computing system audit

transmission must be encrypted using cryptography with specified strength. Then the system design in a QMS document stipulates that data must be transmitted with the TLS protocol that uses 256-bit AES encryption. The audit agents will ask the development staff to provide the configuration of data transmission, the log of the transmission and relevant network operations, and the relevant output/screenshot. The development staff collect these data as evidence, and the agents validate if the data really show that TLS protocol with 256-bit AES is used for data transmission.

3 Reference Architecture for Compliance Validation/Enforcement

Figure 4 depicts the reference architecture of compliance validation/enforcement during the audit procedure and the system operation. The reference architecture is based on our years of research work and professional practice on the compliance of IBM Watson Cloud/Platform for Health [32], and covers the major procedures and modules of compliance validation/enforcement as far as we know. The reference architecture presents a comprehensive image of the compliance validation/enforcement.

Currently the audit procedure is conducted manually, with aid of certain scripts developed for helping with evidence collection and validation. However, a number of advanced technologies and state-of-the-art practices have been innovated in the recent decade for automating certain procedures or modules of compliance validation and enforcement. So before describing the details of these technologies and practices, here we give a brief introduction of the procedures/modules illustrated in Fig. 4.

(1) Compliance Text Analysis

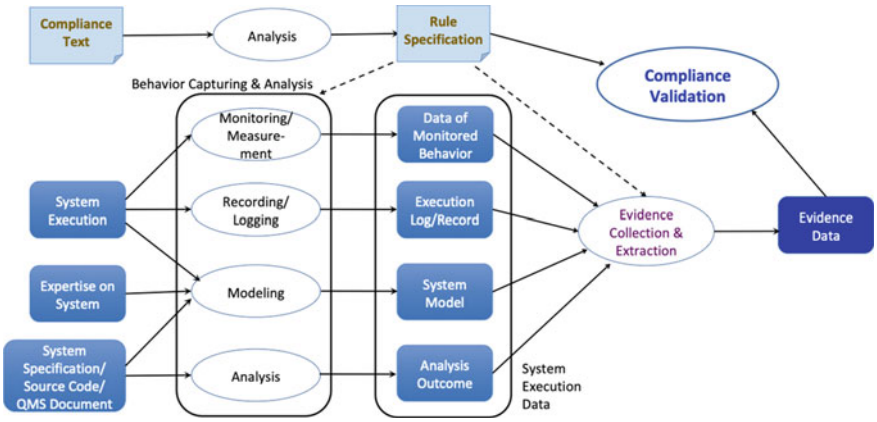


Fig. 4 Reference architecture for compliance validation during the audit and compliance enforcement during system operation

One important step in validating or enforcing compliance in a critical computing system is to understand and analyze the compliance rules. The compliance rules are from law acts, articles of standards, or government regulations. Their texts are written in languages and terms close to humans' understanding of involved services, i.e. natural languages, and should be analyzed or transformed into a form that can be more easily exploited for computers to understand or execute in order to approach the objective of automating the compliance validation.

(2) Behavior Capturing and Analysis

System execution data are collected for compliance validation/enforcement. Such data are collected through the "behavior capturing and analysis" module of the reference architecture. There are four major types of approaches for behavior capturing and analysis: monitoring and measurement, recording and logging, modeling of system behavior, and analysis of system specification, software source code and QMS documents. The rule specification may be needed for what behavior needs to be captured.

(3) Evidence Collection and Extraction

The evidence used for validating a compliance rule is extracted from the collected system execution data. In this step only those data involved in the validation of this rule are identified and used as evidence.

(4) Compliance Validation

This final step validates the compliance rule by comparing the evidence data of the system execution with the evidence data that specify the expected behavior (extracted from the analysis outcome of the system specification or QMS documents), or by checking the evidence data of the system execution against the rule specification.

4 Technologies and Practices for Compliance Validation/Enforcement

Here we give details of the state-of-the-art technologies and practices for compliance validation and enforcement.

4.1 Compliance Text Analysis

Traditionally the analysis of the text of compliance articles is conducted manually as part of the manual audit process. Experts that understand the particular compliance articles and have experience of audit on the compliance articles, e.g. IT experts and domain experts (such as medication experts for HIPAA related compliance text) together, do the analysis.

Recently there are research work that try to extract technical rules from compliance text directly. Brandic et al. [33] proposed to design languages that describe and specify compliance rules in cloud computing, including those describing compliance requirements of computing systems, those specifying domain relevant terms and contents, and those specifying to what extent the compliance requirement should be enforced.

A major improvement in compliance text analysis was made by Adam et al. [34] in IBM Research. They leveraged the standard terminology of technology defined by National Institute of Standards and Technology of US (NIST), applied Natural Language Processing and grammar analysis techniques to understand compliance text and extract rules from it, and then used text classification techniques to translate the compliance text into rules and specifications in NIST terminology.

Another thrust in automating compliance text analysis is [35]. This joint work by UIUC and IBM defined a taxonomy framework related to their target compliance text, and then applied machine learning and modeling technologies to process the natural language of the compliance text, particularly resolving the vagueness and ambiguities in the text, by means of the taxonomy framework. The result of their processing is the accurate and definite compliance rules in terms of the concepts defined in the taxonomy framework without vagueness or ambiguity.

4.2 Behavior Capturing and Analysis

Execution behavior of computing systems should be collected for purposes of compliance validation/enforcement. Modern critical computing systems such as cloud systems, big data analytics platforms and AI platforms usually provide many monitoring and measurement capabilities (e.g. Bro [36] and Zeek [37] for monitoring network events, and Software Defined Network popular in such systems also

enables certain monitoring capabilities), which can be exploited for capturing certain execution behavior. Here we describe several state-of-the-art techniques of behavior capturing and analysis for the compliance purpose.

Burg et al. [38] invented a license tracing technique for compliance validation. They intercepted system calls to trace the entire building process of a software from the source code form to the executable code form and its deployment. The tracing technique monitors the files of the software all the time. When part of the software is updated, rebuilt and re-deployed into the computing system the technique checks all steps of the process. Therefore, the license tracing technique ensures there is no license violation in all software components of the computing system and the system conforms to license compliance.

Brandic et al. [33] proposed a technique of application-aware compliance support in a cloud system. The technique requires the applications in the cloud system be aware of certain compliance rules and the cloud system provide APIs for these applications to interact with the system's compliance service for invoking relevant operations and exchange relevant information. The APIs allow the applications or the system administrators to specify what data should be collected for a certain application, what compliance requirements need to be satisfied for certain components of the system, etc. A specification language was designed in the work to support the API-based communications among the applications, the system's compliance service and the administrators.

As the reference architecture in Fig. 4 shows, system execution logs can also be used for analysis of the system/application behavior because such logs, e.g. the syslog of the Linux system, logs of certain services and monitoring tools, and certain database logs, bear important system execution information that are very helpful for compliance validation. In one example technique [39] the authors performed continuous compliance audit using transaction logs of databases.

Models of computing systems or applications can also be devised to help with compliance validation/enforcement. Here are some of such works. Majumdar et al. [40] designed a dependency model of the components of an OpenStack-based cloud system and a relevant threat model manually. Then based on these models they identified the crucial steps of different cloud operations' workflows, launched attacks onto these critical steps of the operation workflows, and checked whether the responses of the cloud system agree with the responses specified in relevant compliance rules. Esayas [41] employed the CORAS tool [42] to model the threats and risks in business processes and to validate the compliance with the models. Governatori and Rotolo [43] manually built models in a formal language for describing business processes and workflows as well as validating the compliance.

Certain compliance requirements can be checked or validated by means of analyzing configuration files, software source code or QMS documents, as illustrated by the "analysis" oval in the reference architecture of Fig. 4. For example, a cloud system may be required by compliance to only use AES for symmetric encryption and the AES key should be 256 bits or longer. During validation the relevant configuration files specifying the encryption method and the key length are fetched and checked.

4.2.1 REPTrace for Capturing Request Processing Behavior

Here we present our technology REPTrace [44] that captures the behavior of request processing in a cloud platform or a big data analytics platform. The processing of a service request in such a platform is typically executed by multiple components, and a request execution path provides the holistic view of the platform's processing of the request. In particular, Request Execution Path (REP) is the complete end-to-end execution path of processing an individual request among all involved components of the platform. The REPTrace technology provides a common and transparent way to obtain the REP by means of intercepting relevant library calls and system calls, generating events, identifying the causality or temporal order of these events, and stitching these events together into an integral unseparated view of the processing of the request.

The REPTrace technology is built on a comprehensive analysis of the execution of service request processing. Specifically, we found that there are 7 execution scenarios of request processing based on our years of experience in research, development, and operation on real-world cloud platforms and big data platforms. The 7 scenarios are illustrated in Fig. 5, and are briefly outlined below (the bullets a, b, …, g correspond to the labels in the figure):

(a) Continuous execution within a thread;
(b) The current thread creates another thread and passes the handling of the request to the new thread. The current thread may stop processing (e.g. sleep), or continue processing this request;
(c) The current process forks a process, and passes the handling of the request to the new process. The current process may stop or continue processing this request;
(d) The current process/thread sends a message to another process/thread, which may be on the same machine or a different machine. Then the latter process/thread begins the processing of the request. Network communication and other similar mechanisms, like pipe, are covered in this scenario. The current process/thread may stop or continue processing this request;

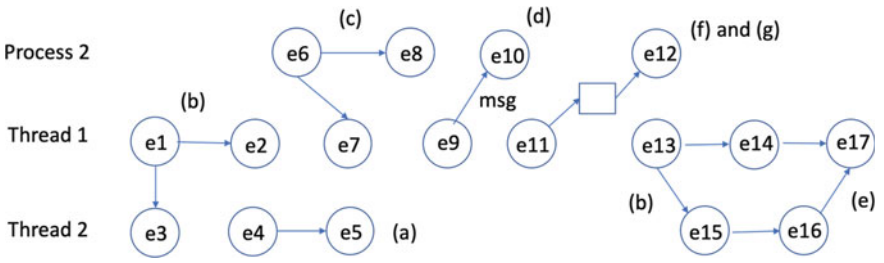


Fig. 5 7 execution scenarios of service request processing

(e) The current process/thread synchronizes the processing of the request with another existing process/thread using certain IPC (Inter-Process Communication) mechanism such as process wait, thread join, signal, lock/unlock, semaphore, etc.;
(f) The current process/thread saves the request (or its intermediate state) in a message queue. Then a different process/thread picks up the request (or the intermediate state) from the message queue and begins processing;
(g) The current process/thread passes the handling of the request to another existing process/thread using shared memory, shared variables, or mapped device.

Architecture. The REPTrace architecture consists of multiple agents (REPAgents) and a single central unit (REPGenerator). A REPAgent is installed in each compute node (virtual machine, container, or physical machine) that intercepts certain library and system calls in the compute node and sends a corresponding trace event to the central REPGenerator. Trace events contain node information, process information, thread information, function call information, and so on. REPGenerator collects these trace events from each node, identifies causal/temporal relationships between them, and links them into a REP (a directed acyclic graph).

REPTrace introduces two IDs for the identification of causal/temporal relationships between trace events:

- **MSG_ID** is the unique ID of each network message. Those trace events associated with network messages, e.g. sending or receiving of messages, are marked with the message's MSG_ID. REPGenerator links the sending event and the receiving event of the same message by matching the MSG_ID of the two events.
- **MSG_CTX_ID** is introduced to group those local events within a thread that are related to the same individual message; i.e., all those events generated within a thread during the thread's processing of one individual network message are marked with the same MSG_CTX_ID. REPGenerator links the trace events with the same MSG_CTX_ID according to their timestamps.

These two IDs are properly employed by REPGenerator to stitch all events of one request's processing into a complete REP (algorithm details available in [45]). The REP can be then used for analyzing the request processing behavior of a cloud platform or a big data analytics platform [46].

4.3 Evidence Collection and Extraction

Those relevant data of system execution, which were obtained from the behavior capturing and analysis, are collected as evidence during audit and post-release compliance validation/enforcement (see the life cycle in Fig. 2). Current practice of audit is manually performed by audit experts and developer experts, and the evidence collection and extraction during audit is also done manually.

However, the evidence collection and extraction during the post-release validation can be automated or semi-automated by providing scripts that automate corresponding manual operations during the audit process. Such automation or semi-automation is possible because mostly the evidence data for a specific compliance rule are kind of same, i.e. the evidence data used in a post-release validation and those used in the previous audit for the same rule are the "same" data which were just generated on different dates. They may be files placed in the same directory with different filenames (date information being part of the filename), or tuples in the same database table with different date information. So, scripts that automate the manual evidence collections during the audit may be developed for the automated post-release compliance validation.

4.4 Compliance Validation

Compliance validation is the process that checks the evidence data against the compliance rules or compares the evidence data with the behavior specified in QMS documents or configuration files. Similar to the evidence collection and extraction, the validation process during the audit is mainly manual, and the validation process during the post-release compliance enforcement can be automated or semi-automated by means of scripts capturing what were manually performed during the audit.

The compliance validation is tightly coupled with the compliance text analysis and the generated compliance rules (please refer to Fig. 4) because the validation is essentially the operation that implements what the rules specify. So those techniques applied for compliance text analysis may be extended for compliance validation. For example, in addition to translating natural-language compliance text into terms in NIST terminology, the technique [34] we introduced in Sect. 4.1 also developed scripts and programs as primitive functions for the NIST terms used in the result rules and specifications from the translation. The primitive functions perform query, collection or manipulation operations of IBM cloud resources and data, so executing a rule in natural language for compliance validation is automated as invoking these primitives according to the result rule in the NIST terms. Unlike the straightforward scripting of what was done during the audit process, this type of automated compliance validation does not require a manual audit process as prerequisite. Considering the generality of the NIST terminology, the technique has the potential of being applied for other cloud systems similar to IBM cloud.

4.4.1 Validating Consistency of Data Stores in a Big Data Analytics Cloud

As a case study, here is our experience of validating consistency of data stores in IBM Watson Cloud/Platform for Health. The IBM Watson Cloud for Health is a big data analytics cloud hosting life science and medication applications. The target clients

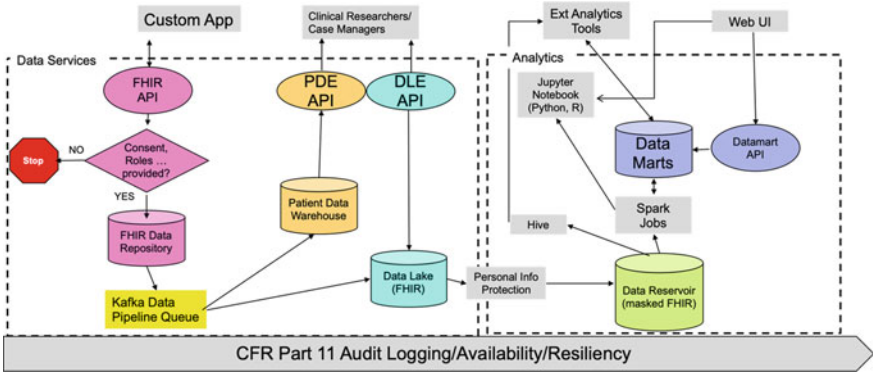


Fig. 6 High-level architecture of IBM Watson Cloud for Health (FHIR is the acronym of Fast Healthcare Interoperability Resources, a draft standard describing data formats and elements for electronic health data)

of the cloud are drug manufactures, medical device companies, hospitals and clinics, doctors and patients, and medical schools, institutions and researchers.

The high-level architecture of IBM Watson Cloud for Health is illustrated in Fig. 6. It is a Platform-as-a-Service cloud for health data, which provides the services of the acquisition, curation, validation, storage, management and governance of health data, and allows clients to develop and execute analytics applications over the health data.

There are multiple data stores in this system including FHIR Data Repository, Patient Data Warehouse, Data Lake, Data Reservoir, Data Marts, etc. According to the regulatory controls such as HIPAA and GxP (CFR Part 11 in the diagram, or Code of Federal Regulations Title 21 Part 11), the health data must be consistent among these data stores. Please refer to [47] for in-depth discussions on the consistency of data stores in IBM Watson Cloud for Health.

A simple example consistency rule is here: if a database in Data Lake contains records of a patient's visit to a doctor, then the patient's profile must be present in a database in Patient Data Warehouse, and the doctor's profile must also be present in another database in Data Lake.

Approach to Consistency Validation. All the data in the cloud system must be auditable. We designed the audit service of the cloud that maintains a set of databases for storing audit information for the data in the system.

We used these audit databases as the calibration reference for the consistency of the data stores. All the data records in the system's data stores must be consistent with the records in the audit databases. Fields associated with records in audit databases were added to the tables and databases of the data stores, and logging steps were added to operations that manipulate records of the data stores. The logging steps record the operation in the audit databases and do the bookkeeping of the associated fields in corresponding data stores.

Then we developed a data consistency validation tool based on the audit databases to continuously check all data stores and provide proof for the audit service. The tool examines the data in the data stores against tens of consistency rules, which, in our experience, were created manually by domain experts and audit experts. If inconsistencies are found, the tool changes the contents of the data stores properly to make them consistent with what are in the audit databases. The approach is best-effort validation that covers important consistency rules in the cloud system.

5 Conclusions

With the popularity of cloud platforms, big data platforms and AI platforms for hosting critical services and applications, such critical computing platforms and the critical services/applications are subject to compliance regulations imposed by governments, legislation or industry community. The compliance is the key for the governments and the public to trust these computing platforms and their hosted services, and to ensure there will be no severe unacceptable impacts of system failures, security breaches or human errors.

This article discusses the problem of providing compliance in critical computing systems, including the descriptions of compliance rules and the compliance audit process, and proposes a reference architecture for compliance validation/enforcement. Then we present state-of-the-art technologies and practices of compliance validation/enforcement. We believe this article is very helpful for those professionals and practitioners who work on critical computing systems and services.

References

1. Julisch K (2008) Security compliance: the next frontier in security research. ACM NSPW
2. Heiser J, Nicolett M (2008) Assessing the security risks of cloud computing. Gartner report. 3(27):29–52
3. Sturgeon W (2005) Jail or compliance? You decide. Directors told, on http://www.silicon.com
4. Summary of the HIPAA Security Rule. https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html
5. GxP Wiki. https://en.wikipedia.org/wiki/GxP
6. Financial services compliance overview. Google Cloud Whitepaper, May 2019. https://cloud.google.com/files/financial-services-compliance-overview.pdf
7. EU GDPR.org. https://eugdpr.org/
8. AWS Compliance Programs. https://aws.amazon.com/compliance/programs/
9. Compliance on the IBM Cloud. https://www.ibm.com/cloud/compliance
10. Azure Compliance. https://azure.microsoft.com/en-us/overview/trusted-cloud/compliance/
11. Cloud Compliance–Regulations & Certifications. https://cloud.google.com/security/compliance/
12. Al-Aqrabi H, Liu L, Xu J, Hill R, Antonopoulos N, Zhan Y (2012) Investigation of IT security and compliance challenges in security-as-a-service for cloud computing. In: 2012 IEEE 15th international symposium on object/component/service-oriented real-time distributed computing workshops. IEEE, 11 Apr 2012, pp 124–129

13. Gudivada VN, Nandigam J (2009) Corporate compliance and its implications to IT professionals. In: 2009 Sixth international conference on information technology: new generations 27 Apr 2009, pp 725–729
14. Ragan T (2006) Keeping score in the IT compliance game: ALM can help organizations meet tough IT compliance requirements. Queue 4(7):38–43
15. Yimam D, Fernandez EB (2016) A survey of compliance issues in cloud computing. J Internet Serv Appl 7(1):1–12
16. Von Solms SB (2005) Information security governance–compliance management versus operational management. Comput Secur 24(6):443–447
17. Tianfield H (2012) Security issues in cloud computing. In: 2012 IEEE international conference on systems, man, and cybernetics (SMC). IEEE, 14 Oct 2012, pp 1082–1089
18. Hashizume K, Rosado DG, Fernández-Medina E, Fernandez EB (2013) An analysis of security issues for cloud computing. J Internet Serv Appl 4(1):1–13
19. Ardagna CA, Ceravolo P, Damiani E (2016) Big data analytics as-a-service: issues and challenges. In: 2016 IEEE international conference on big data (big data). IEEE, 5 Dec 2016, pp 3638–3644
20. Kibbe DC (2005) Ten steps to HIPAA security compliance. Fam Pract Manag 12(4):43
21. Artnak KE, Benson M (2005) Evaluating HIPAA compliance: a guide for researchers, privacy boards, and IRBs. Nurs Outlook 53(2):79–87
22. Cannon JC, Byers M (2006) Compliance deconstructed: when you break it down, compliance is largely about ensuring that business processes are executed as expected. ACM Queue. 4(7):30–37
23. https://en.wikipedia.org/wiki/Sarbanes–Oxley_Act
24. Ullah KW, Ahmed AS, Ylitalo J (2013) Towards building an automated security compliance tool for the cloud. In: 2013 12th IEEE international conference on trust, security and privacy in computing and communications. IEEE, 16 Jul 2013, pp 1587–1593
25. Mather T, Kumaraswamy S, Latif S (2009) Cloud security and privacy: an enterprise perspective on risks and compliance. O'Reilly Media, Inc., 4 Sep 2009
26. Beautement A, Sasse MA, Wonham M (2008) The compliance budget: managing security behaviour in organisations. In: Proceedings of the 2008 new security paradigms workshop, 22 Sept 2008, pp 47–58
27. Kilbridge P (2003) The cost of HIPAA compliance. N Engl J Med 348(15):1423
28. Kalaiprasath R, Elankavi R, Udayakumar R (2017) Cloud security and compliance-a semantic approach in end to end security. Int J Smart Sens Intell Syst 15:10
29. Hendre A, Joshi KP (2015) A semantic approach to cloud security and compliance. In: 2015 IEEE 8th international conference on cloud computing. IEEE, 27 Jun 2015, pp 1081–1084
30. HIPAA Regulations. http://www.hipaasurvivalguide.com/hipaa-regulations/hipaa-regulations.php
31. ISO 9001:2015, Quality management systems—requirements. https://www.iso.org/standard/62085.html
32. IBM Watson Health. https://www.ibm.com/watson/health/
33. Brandic I, Dustda S et al (2010) Compliant cloud computing (C3): architecture and language support for user-driven compliance management in clouds. In: IEEE international conference on cloud computing 2010
34. Adam C, Bulut M, Hernandez M, Vukovic M (2019) Cognitive compliance: analyze, monitor and enforce compliance in the cloud. In: IEEE international conference on cloud computing 2019
35. Thakore U, Ranchal R, Wei Y, Ramasamy H (2019) Combining learning and model-based reasoning to reduce uncertainties in cloud security and compliance auditing. In: Proceeding of international symposium on reliable distributed systems (SRDS). Industry Track
36. https://github.com/bro/bro
37. https://github.com/zeek/zeek
38. Burg S, Dolstra E, McIntosh S et al (2014) Tracing software build processes to uncover license compliance inconsistencies. ACM ASE

39. Hasan R, Winslett M (2011) Efficient audit-based compliance for relational data retention. ASIACCS
40. Majumdar S, Jarraya Y, Madi T et al (2016) Proactive verification of security compliance for clouds through pre-computation: application to OpenStack. In: Computer security—ESORICS
41. Esayas S (2014) Structuring compliance risk identification using the CORAS approach: compliance as an asset. In: IEEE ISSREW
42. http://coras.sourceforge.net/coras_tool.html
43. Governatori G, Rotolo A (2009) How do agents comply with norms? In: IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology—workshops
44. Yang Y, Wang L, Gu J, Li Y (2018) Transparently capturing execution path of service/job request processing. In: International conference on service-oriented computing (ICSOC) 2018, vol 11236. Lecture Notes in Computer Science
45. Yang Y, Wang L, Gu J, Li Y Transparently capturing request execution path for anomaly detection. https://arxiv.org/abs/2001.07276
46. Gu J, Wang L, Yang Y, Li Y (2018) KEREP: experience in extracting knowledge on distributed system behavior through request execution path. In: Best paper nominee, IEEE international symposium on software reliability engineering (ISSRE). Industry Track
47. Wang L, Ramasamy HV, Salapura V, Arnold R, Wang X, Bakthavachalam S, Coulthard P, Suprenant L, Timm J, Ricard D, Harper R, Gupta A (2019) System restore in a multi-cloud data pipeline platform. In: The international conference on dependable systems and networks (DSN). Industry Track

Application-Aware Reliability and Security: The Trusted Illiac Experience

Karthik Pattabiraman

Abstract This chapter is about the author's time at the University of Illinois as a PhD student in Ravi's group working on the Trusted Illiac project at the University of Illinois (UIUC) from 2004 to 2009. The author starts by narrating his initial involvement in the project, and how it grew as time progressed. He then reflects on the lessons he learned from the project, and how the project has influenced his subsequent research career.

1 Introduction

This chapter chronicles the time from the start of 2004 to the end of 2008, when I was a PhD student at the University of Illinois in Prof. Ravishankar (Ravi) Iyer's group. The start of this period coincided with the start of what later became the Trusted Illiac project at Illinois, which was completed around the end of 2013. While I have attempted to stay true to the historical facts and timeline, many of these are based on my recollection, and are hence my biased views. Further, in the interest of space, I have focused on the most pertinent facts that led to the development of the ideas and implementation of the Trusted Illiac, even if at the time it was difficult to discern what these were (as is typical of research). Finally, though I have written this in the first person singular, the Trusted Illiac project was the collective effort of a great many people—however, I have used the pronoun 'I' rather than 'we' in most places, as this is primarily based on my personal experience.

2 Background and Beginnings

The Trusted Illiac, a 256 node Linux cluster with each node having 2 processors and onboard FPGA (Field Programmable Gate Array) boards to provide customized

K. Pattabiraman (✉)
The University of British Columbia (UBC), Vancouver, Canada
e-mail: karthikp@ece.ubc.ca

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_11

checking for reliability and security to applications, was inaugurated in 2013 at the University of Illinois [1]. However, the main ideas behind it originated almost a decade ago in 2003, with the publication of the Reliability and Security Engine (RSE) [2], by Nithin Nakka and others at the DSN 2004 conference. The RSE was an architectural framework for providing reliability and security support to applications, in a separate module that was decoupled from the main processor's pipeline.[1] The RSE was supposed to interact with the main processor through a special instruction called CHECK, which was an extension of the processor's Instruction Set Architecture (ISA). The CHECK instruction was supposed to be inserted by the compiler or the programmer, and would be invoked at runtime by the RSE module. The RSE paper demonstrated the feasibility and practicality of this idea, by implementing it in a microprocessor simulator (Simplescalar [7]), and showing that the overheads were low. However, the authors of the RSE paper had hand annotated the application binaries with CHECK instructions to demonstrate it.

When I joined Ravi's group in January 2004, I started working on an automated tool for inserting CHECK instructions into application binaries to interface with the RSE. My original thinking was that this would be a "starter" project for 6 months to a year, before I would move on to the actual research. I cringe today when I look back at how naive I was back then !

The starter project ended up forming the basis of my PhD thesis, and even some of the early research I did as a tenure-track faculty member later. This project also spawned many PhD dissertations, including my own [3], and those of some of my students. With the benefit of hindsight (and perhaps a little hubris even), I would even state that this problem was one of the most challenging pieces of the puzzle that needed to be solved for the Trusted Illiac project. Of course, at the time I started working on this, I did not know this, which was perhaps a good thing as I may have not worked on it, if I had known !

I have divided the remaining part of this chapter into three broad sections, each roughly corresponding to the three phases of the research. However, the grouping is more thematic than chronological—some aspects of the problem became clearer only when the other pieces fell into place, which necessitated multiple iterations of the three phases. Further, some of the missteps I took led me to revisit the earlier phases later. I then distill some of the key lessons learned, and the aftermath of the project in terms of its impact on my later research.

3 Early Years (Detector Placement)

As mentioned, when I started working on the project, the goal was to insert CHECK instructions at the "appropriate" places in the application binaries in order to allow the

[1] The original implementation of the RSE assumed a single core, out-of-order superscalar processor, which was the standard at that time. It did not originally involve an FPGA or any reconfigurable hardware.

RSE module to check the application's properties that were indicative of reliability and/or security violations. A natural first step was to try to decouple the two aspects, namely the placement of the detectors (we used the term detectors as it is more generic than CHECK instruction) and the actual contents of the detectors. The first problem is related to detector placement, and the second to detector derivation.

To tackle the first problem of detector placement, I initially conceptualized an "ideal" detector, or one that would have 100% coverage of faults that would propagate to it. This idea had been pioneered by Propane [4]. A simple way to implement an ideal detector is to record a complete "golden" trace of all the values in the program's dynamic execution and to compare it with the original program. However, this method has two disadvantages. First, it leads to a significant amount of data that needs to be recorded and processed, even for moderate sized programs (recall that this was in 2004 when computers were much less powerful than they are today). Second, and more importantly perhaps, even if one could record all the data, it would be infeasible to directly compare the faulty execution with the results from the golden run due to potential deviations in the program's control flow due to the injected fault. Therefore, it became clear that we needed a more robust approach to implement an ideal detector.

In the program analysis literature, there is the notion of a Dynamic Dependence Graph (DDG) proposed by Agrawal et al. [5], which captures the dependencies among the instructions in a dynamic execution trace of the program. I realized that one can leverage the DDG to trace error propagation in the program (admittedly, this idea is not novel [6], though I did not know it at that time). The DDG provided a convenient abstraction to trace a program's execution and compare the golden trace with the faulty trace without worrying about differences in the control-flow due to the fault.

However, there remained the question of how to construct the DDG of a program (efficiently). I explored different approaches to this problem, including using program analysis tools at the source code level, but unfortunately, they either proved to be rather brittle to use, or did not capture many of the dependencies that could lead to error propagation. Therefore, I decided to build my own tool for error propagation analysis based on the DDG. Because the RSE had been implemented in the Simplescalar simulator, I had some degree of familiarity with the simulator code, and decided to implement my DDG construction tool using Simplescalar itself. In retrospect, perhaps this was not the best choice, but it was very fast and scalable, and it allowed me to analyze the DDG even for moderately sized programs.

I do not wish to bore the reader with the laborious details of building the tool, but I would like to point out two things. The simplescalar simulator is a very well engineered piece of software, but there were certain aspects of it that were not realistic. In particular, its behavior when faults were injected was not necessarily faithful to how modern memory segmentation worked in programs (as that was not its main purpose). Therefore, I had to implement this in the simulator in order to use it to track error propagation in a somewhat realistic manner (this involved modeling the memory segmentation). Second, as mentioned, computers were much slower back then, and I chose to write much of the high-level trace processing in Python, a rather

new language at that time (my excuse being that I wanted to learn it). However, the default garbage collector in Python was not very sophisticated, which led to my program running out of memory, and thus crashing ! I had to tweak the Python garbage collector (i.e., its parameters) to free up memory more aggressively in order to get my program to run adequately to collect large traces.

After the tool was built, I attempted to use it to inject faults and identify the locations where (ideal) detectors should be inserted to detect faults with the highest coverage. Unfortunately, this task proved to be quite challenging as well as what worked well for one program did not for another program. I remember spending many endless nights with various statistical packages attempting to find correlations in the data that may reveal uniform properties of locations where the detectors should be placed. Ravi was very supportive of this effort and shared his experience with doing failure data analysis with similar packages. Finally, we arrived at the breakthrough result, which identified the characteristics of variables for detector placement, and was published at PRDC'05 [8]. This was my first paper in this area, and it formed the basis of my PhD thesis.

The main insight underlying this work is that the instructions that resulted in high fanout values were most likely to propagate the faulty values, and would be the places where detectors should be placed to prevent error propagation. While perhaps obvious in retrospect, this result was surprising at that time (I believe) as fanouts performed much better than more sophisticated metrics. This result has since been confirmed by other studies [9], although there are some caveats in how it works as they have pointed out. More importantly, this result established for the first time that there is a correlation between the properties of a program in terms of its data-dependencies and how errors propagate in it. Though this has been established as conventional wisdom now, it was not the case then. I owe it to Ravi who believed so much in this result that he made it a point to highlight it in all his talks, and popularized it despite its unconventional nature. This work has formed the basis of later work done by my students at UBC as well [10].

4 Middle Years (Detector Derivation)

Once I had identified the locations for placement of detectors based on the program's DDG and heuristics such as fanouts, the next problem was to determine how to derive the detectors. Recall that I had considered ideal detectors in the earlier phase, though in the real world, there is no such thing as an ideal detector. Therefore, the question was how to come up with detectors for the locations identified as the placement points.

To approach this problem, I initially profiled the values of the locations in the detector placement points using the same infrastructure I had built based on Simplescalar. To my surprise, I found that many of the values in these locations exhibited regular patterns that could be described by simple rules of the form "The values are consecutive numbers in the sequence [x, y] (x and y are integers), or are

zero". I found that there were six patterns or rules that could be used to describe more than 95% of the values at these locations, and that these rules can be checked efficiently at runtime. This led to the second paper of my PhD, and was published at EDCC 2006 [11].

Though the rules for describing the detectors were straightforward to derive by humans, I wanted to automate this process to reduce the burden on programmers. However, though most of the rules were quite simple, there were tricky corner cases that needed to be handled. Taking inspiration from work on dynamic invariant detection [12, 13] in the software engineering literature, I built another tool (using Python again) to efficiently learn the rules and the exceptions to the rules for the different detector types. I used fault injection to evaluate the detectors derived, and found that the detectors achieved both high coverage and low false-positives across a large set of program inputs.

Together with other colleagues in the group, I then worked on coming up with an efficient hardware implementation of the above dynamic detectors. We implemented the rule templates on an FPGA board that was running in conjunction with the main processor. This was one of the first hardware prototyping efforts we did in the group, and the first work to use FPGAs for error detection (to the best of my knowledge). Though we used the RSE concept for integrating the detectors with the processor, our implementation was distinct from it.

The paper [11] was the first to use dynamic analysis for automatically deriving detectors, and has spurred many follow-up efforts from both industry [14] and academia [15]. However, there were two main issues with this approach. First, it needed a significant number of representative program inputs in order to keep the false-positive rate low (i.e., detectors raise an error when there is none). However, in practice it is difficult to obtain such large input corpuses. Second, there is a saturation in the coverage provided by the detectors as a result of overlap among detectors in terms of the faults covered. Thus, though our technique was better than the state of the art at that time, it was still not amenable to be deployed in a large-scale system such as the Trusted Illiac (though we had not finalized the details, the broad contours of the system were in place then).

With Ravi's encouragement, I therefore began to work on an alternative technique to derive the detectors with *no* false-positives, and one that could scale to large systems. The idea was to use static analysis via the compiler to extract the salient properties of the application to be used as detectors. Because static analysis is sound, the detectors will hold for all legitimate code paths in the application, and will hence have no false positives. Unfortunately, the state-of-the-art compilation tools at that time were either very heavyweight or required intimate knowledge of compiler internals, which I did not have. However, I had taken a compilers class taught by Prof. Vikram Adve in which we used the LLVM compiler developed by his group. Back then, LLVM was still a research project [16], but I remember being impressed by its modularity and ease of modification. Therefore, I decided to implement my static analysis approach using the LLVM compiler.

Thus began one of the most intense but enjoyable periods of my PhD, where I would immerse myself in intricate details of the LLVM infrastructure, and learn

to appreciate deep compiler concepts such as Static Single Assignment (SSA) form [17]. This was both exhilarating and frustrating, as I had to come up with the program property that I was going to use for error detection, while at the same time implementing it in LLVM. After much experimentation, I decided to use the backward slice of an instruction to be checked as the detector, but in an optimized fashion. The idea was to isolate the backward slice in a separate basic block, and let the LLVM compiler apply its optimizations to it. It also specialized the block for each path through the program, and dynamically chose the detector based on the path executed at runtime (via program instrumentation) [18]. This idea was inspired by trace scheduling in compilers [19].

While the above idea was simple in theory, its implementation in LLVM proved to be very complex, and it was non-trivial to get it to work for even small programs. Again, Ravi's encouragement was an important reason I stuck to this task even though I did not make much progress for many weeks. It was his ability to see the end product and its benefits that inspired me to carry on. When I finally got it to work, it was really exciting to see that many of Ravi's predictions about its benefits came true. The technique proved to obtain high coverage without having any false-positives, across a broad range of programs. Furthermore, because it was implemented in a compiler, it could be easily integrated into a programmer's workflow and scale to large systems.

The paper describing the above project was published in IOLTS 2007, a hardware testing venue [18]. Though we had initially targeted dependability venues, the paper had been rejected despite getting good reviews, and I was getting nervous about its publication. Ravi convinced me that it was better to get the work published, even if not at dependability venues, and so we repositioned the paper accordingly. I believe that this was the correct decision, as this paper later came to be cited by many other papers.

Before concluding this section, I wanted to reflect on some of the lessons I learned from Ravi in the above endeavors. First, I learned to never "settle" for good enough, as though the dynamic detectors were better than the state of the art at that time, Ravi saw that they would not be scalable in the long term. It was better thus to start with a clean slate, and come up with the static analysis approach to derive the detectors. Second, it is important to visualize the final product and the benefits it will bring, even when one is buried in the details of the research. This is important for evaluating whether the research effort is "worth it". These are two lessons that have stuck with me ever since, and I have attempted to pass them on to my own students in my research career.

5 Later Years (Detector Implementation and Validation)

The detectors derived via static analysis were primarily implemented in software, and hence entailed high performance overhead. Therefore, together with my colleagues, I decided to implement them on FPGA hardware, similar to the dynamically derived

detectors. However, there were three challenges in the same. First, because the implementation of the statically derived detectors was in LLVM, we needed the ability to run LLVM on the target platform (a Leon3 core [20]). Unfortunately, the LLVM compiler at that time did not support the target platform. Second, we needed a mechanism to translate the software-based checks into a generic format that could be implemented on an FGPA. Finally, we needed a mechanism to efficiently track the control paths executed by the program, as this was done via software instrumentation.

To address the first challenge, namely the lack of support for the target platform, I used tools in the LLVM compiler to translate the Intermediate Representation (IR) to C code, after the checks had been inserted by my static analysis passes. This C code could then be cross-compiled to the Leon3 platform by a custom C compiler for that platform.

To address the second challenge, I built a generic three-address representation for the checks and my colleagues wrote an automated translator to convert these to Verilog code. For the third challenge, I converted the path-tracking code to a set of finite state machines, which could be programmed into the FPGA board and tracked at runtime. We prototyped the system on an FPGA board with the Leon3 core, and measured the performance and power overheads. This paper was published at DSN'09 [21].

In the process of implementing the detectors in hardware, another issue came up. This was related to the different layers of compilation (and cross-compilation) before the detectors were executed. At each stage, the compiler could either remove the detectors, or introduce program states that would not be subject to the error detection. This raised an important question: "how effective were the detectors after implementation".

To answer this question, I initially built a fault injection framework at the LLVM IR code level that could inject different types of faults and trace their propagation in the code. This framework allowed me to debug the corner cases of errors that were missed by the detectors, and determine whether and how to augment the detectors. The core of this framework was later developed by my graduate students and released as the LLFI fault injector, which has since become widely used in both academia and industry [22].

However, this fault injection framework still did not allow me to study what happened to the detectors after they had been transformed into assembly code via the compiler. To analyze these faults, I needed to build error propagation analysis tools at the assembly language level. At this time, I came across some work in the programming languages community on using type-safety at the assembly language level for reasoning about fault tolerance mechanisms [23]. Inspired by this approach, I decided to build a similar tool for reasoning about the detectors. I had also taken a class on formal methods at Illinois that used the Maude framework [24], and I was impressed by its ease-of-use. Therefore, I decided to use Maude to build a formal tool for error propagation analysis.

The formal tool for analyzing the error propagation analysis later came to be known as SymPLFIED (Symbolic Program Level Fault Injection and Error Detection

framework). This SymPLFIED paper [25] won the William Carter award at DSN'08.[2] More importantly, the tool was the first to integrate formal methods (i.e., model checking) with error detector validation.

I will not repeat the details of the SymPLFIED paper [25], except to say that the tool formalized the semantics of assembly language instructions (in Simplescalar assembly) and error detection, so that it was possible to run formal analysis (i.e., model checking). However, I wish to point out two things that are perhaps surprising in retrospect, at least to me. First, the entire process of building the tool from conception to evaluation and paper writing was less than four months (from mid-August to mid-December, 2007). This suggests that it was possible to get a working implementation, and paper written to a top conference like DSN if the idea behind it was clear. Admittedly, this four month period involved a significant number of long working days, but I was enjoying the act of building something from scratch and learning a new technology in the process, that I did not mind the long days. Second, the main result in the paper, that we found a potential violation in a safety–critical application such as tcas, an aircraft collision avoidance system [26] using SymPLFIED, did not materialize until the last couple of weeks before the paper deadline. I have to give credit to my colleague and co-author Nithin Nakka for this effort as he incessantly tried to use the tool on different programs despite its bugs.

I also want to acknowledge Ravi's encouragement and suggestion that I consider using formal methods in my thesis, in order to expand its scope beyond empirical methods. It was at his insistence that I took the formal methods class at Illinois, and persisted with it despite lacking the requisite mathematical background. If it had not been for that, I would not have been able to build SymPLFIED, and win the Carter award.

6 Other Directions

I have focused this narrative on reliability, as this was my primary focus. However, the Trusted Illiac project also involved security, and I was able to contribute a little to that as well [27, 28]. Unfortunately, some of this work did not get published, and I did not persist with it. Further, while I have covered the period till the end of 2008, the year I left Illinois, the Trusted Illiac project continued with other students for a few years. I was gratified to learn that some of them used the tools I had built to prototype their ideas [29, 30], though I was not directly involved. Finally, in 2013, the Trusted Illiac project was inaugurated by the Chancellor of Illinois [1].

[2] Prior to 2015 when the award was renamed as the William Carter PhD dissertation award, the award was given to a DSN paper by a PhD student that has made a fundamental contribution to dependability.

7 Lessons Learned

I have tried to distill some of the main lessons learned from the Trusted Illiac experience, which I believe apply more broadly than the specifics of the project. There are four lessons as follows.

Lesson 1: It is important to have a big-picture vision to anchor the project.

I think this was an important aspect of the Trusted Illiac project, namely providing reliability and security services in the hardware in an application-aware manner. Though the original vision did not incorporate reconfigurable hardware, the idea of deriving detectors in an application-aware manner was a direct result of the overall vision of the project. Even when we ran into some technical difficulties during the execution of the project, it was clear what needed to be done. This was an important anchor point as it allowed us to make tradeoffs in service of the vision.

Lesson 2: Don't be afraid to change or pivot in the vision if warranted.

A second lesson was that while the overall vision is important, it is all right to pivot or make changes in the vision depending on the results and changes in technology. For example, when the Trusted Illiac project was conceived, FPGA boards were not very powerful and were thought to be more as prototyping devices. When the project was completed a decade later, FPGA boards were being integrated into mainline processor boards by leading manufacturers. Through the course of the project, it became clear that this was how the wind was blowing, and so we quickly pivoted to using FPGA boards in conjunction with regular processors.

A similar lesson was learned regarding the use of compilers in the project. Initially, it was thought that the detectors would be written either by hand or by custom tools developed for this purpose. However, during the project, we realized that integrating the detectors with a industry-standard compiler infrastructure such as LLVM allowed us to leverage state-of-the-art code optimization techniques to obtain high performance, without expending significant effort. Further, as LLVM expanded, it grew to support many different hardware platforms, thereby allowing us to run our tools on these platforms with minimal effort.

Lesson 3: Build reusable tools and infrastructure rather than one-off solutions.

The third important lesson I learned is that it is very important to build reusable tools and infrastructure to support them, rather than one-off solutions. Though the latter have their value in some situations, building reusable tools allows multiple stakeholders to rapidly build on the existing infrastructure rather than rolling their own. It also allowed new students who joined the project to start contributing right away, without reinventing the wheel and building tools. Personally, I benefited from building such tools as it also allowed me to work with a large number of undergraduate and Masters' students on the project, and leverage their skills.

Lesson 4: Use tried and tested solutions for the most part, rather than inventing your own.

I believe the above three lessons contributed to the success of the project. However, there were some missteps I took, which in the end turned out to be dead ends. For example, I spent a lot of time learning new languages and tools just to try them out, rather than use tried and tested solutions for some of the tools I built. While this was a good learning experience for me, it did little to advance the aims of the project, nor did it add much research value to the project.

In the end, I believe that the Trusted Illiac project was a success as not only did it result in the usual dissertations and papers, it also led to a real working hardware used in production settings, which was unusual ! More importantly, it shaped the career trajectories of many of the people who worked on it, including myself. I elaborate more on this in the next section.

8 Aftermath: How Trusted Illiac Shaped My Subsequent Research

The Trusted Illiac project and its ideas ended up having a profound influence on my subsequent research career. After I graduated from Illinois, I joined Microsoft Research as a post-doctoral researcher, and then started a tenure track position at the University of British Columbia (UBC). Many of my research directions have been inspired by the Trusted Illiac project, as follows.

1. *Hardware-Software Integrated Fault Diagnosis*
   Soon after I joined UBC, my student and I started working on extending the results for error detection in my PhD thesis to error diagnosis of intermittent faults [31]. Our work in this area was directly inspired by the Trusted Illiac project, and involved hardware-software co-design of the diagnosis mechanism [32]. In a nutshell, the hardware logs the execution of the program in terms of the instructions it executed, while the software post-processes the traces to diagnose the fault—this is similar to the path-tracking and error detection mechanism in the static detectors.
2. *Detector Placement and Selective Protection*
   As mentioned earlier, I continued the line of work of detector placement by expanding both the scope and types of detectors, as well as the applications. In Blockwatch [33], my student and I proposed a technique to derive detectors for data parallel programs, and protect them from soft errors. With another student, I also built a program analysis technique for identifying detector placement points in approximate computing applications [10]. Finally, a third student and I worked on a machine-learning based approach to identify detector placement points for selective protection in programs [34]. All of these efforts were also inspired by the Trusted Illiac project.

3. *Fault Injection at the LLVM IR level*
   My group has also built and released the LLFI fault injector, which operates at the LLVM IR level, and is capable of injecting a wide-range of both hardware and software faults [22]. The core of LLFI is based on the fault injection tool I had built to evaluate the static error detectors. We have demonstrated the accuracy of LLFI [35] with respect to fault injections at the assembly code level, and have expanded LLFI to Graphic Processing Units (GPUs) as well [36].
4. *Error Propagation Analysis and Modeling*
   Finally, the work I did on error propagation modeling and analysis in the Trusted Illiac project has shaped many of the tools and frameworks my students and I built over the years. For example, both ePVF [37] and Trident [38] were based on the idea of modeling error propagation at the program level, as well as their offshoots, vTrident [39] and GPU-Trident [40]. Though the work has considerably evolved since then, the foundations are based on the Trusted Illiac project.

In summary, the ideas developed in the Trusted Illiac project have profoundly shaped my research, and continue to do so to this day. I, therefore, owe a considerable debt of gratitude to it, and to Ravi for initiating the project and involving me in it !

Acknowledgements The Trusted Illiac project was the collective effort of many people, with whom I collaborated such as Giacinto Paolo Sagesse, Nithin Nakka, Shelley Chen, Peter Klemperer, Daniel Chen, Galen Lyle, William Healey. I would like to thank my PhD advisors, Ravi Iyer and Zbigniew Kalbarczyk for their guidance and support throughout the project. This chapter was partially funded by the Natural Science and Engineering Research council of Canada (NSERC). I would like to thank my former colleagues in the DEPEND group at Illinois for their support, Shuo Chen, Long Wang, and Weining Gu. Finally, a big thank you to my wife Priya for her love and support.

References

1. Rick Kubetz, University of Illinois' Trusted ILLIAC Will Transform Large-Scale Computing, 2013. Available online: https://csl.illinois.edu/news/university-illinois%25E2%2580%2599-trusted-illiac-will-transform-large-scale-computing
2. Nakka N, Kalbarczyk Z, Iyer RK, Xu J (2004) An architectural framework for providing reliability and security support. In: Proceedings of the 2004 international conference on dependable systems and networks (DSN '04). In: IEEE Computer Society, USA, p 585
3. Pattabiraman K (2009) Automated derivation of application-aware error and attack detectors. University of Illinois at Urbana-Champaign
4. Hiller M, Jhumka A, Suri N (2002) PROPANE: an environment for examining the propagation of errors in software. In: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis (ISSTA '02). Association for Computing Machinery, New York, NY, USA, pp 81–85. https://doi.org/10.1145/566172.566184
5. Agrawal H, Horgan JR (1990) Dynamic program slicing. In: Proceedings of the ACM SIGPLAN 1990 conference on programming language design and implementation (PLDI '90). Association for Computing Machinery, New York, NY, USA, pp 246–256. https://doi.org/10.1145/93542.93576

6. Goradia TS Dynamic impact analysis: analyzing error propagation in program executions (Doctoral dissertation, New York University)
7. Austin T, Larson E, Ernst D (2002) SimpleScalar: an infrastructure for computer system modeling. Computer 35(2):59–67
8. Pattabiraman K, Kalbarczyk Z, Iyer RK (2005) Application-based metrics for strategic placement of detectors. In: 11th Pacific Rim international symposium on dependable computing (PRDC'05). IEEE, 12 Dec 2005, p 8
9. Ramachandran P, Hari SK, Adve SV, Naeimi H (2011) Understanding why symptom detectors work by studying data-only application values. In: Workshop on silicon errors in logic–system effects (SELSE)
10. Thomas A, Pattabiraman K (2013) Error detector placement for soft computation. In: 2013 43rd Annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 24 Jun 2013, pp 1–12
11. Pattabiraman K, Saggesse GP, Chen D, Kalbarczyk Z, Iyer R (2006) Dynamic derivation of application-specific error detectors and their hardware implementation. In: Proceedings of European dependable computing conference (EDCC)
12. Ernst MD, Czeisler A, Griswold WG, Notkin D (2000) Quickly detecting relevant program invariants. In: Proceedings of the 22nd international conference on Software engineering, 1 Jun 2000, pp 449–458
13. Hangal S, Lam MS (2002) Tracking down software bugs using automatic anomaly detection. In: Proceedings of the 24th international conference on software engineering. ICSE 2002. IEEE 25 May 2002, pp 291–301
14. Racunas P, Constantinides K, Manne S, Mukherjee SS (2007) Perturbation-based fault screening. In: Proceedings of the 2007 IEEE 13th international symposium on high performance computer architecture (HPCA '07). IEEE Computer Society, USA, pp 169–180
15. Sahoo SK, Li M, Ramachandran P, Adve SV, Adve VS, Zhou Y (2008) Using likely program invariants to detect hardware errors. In: 2008 IEEE International conference on dependable systems and networks with FTCS and DCC (DSN), pp 70–79
16. Lattner C, Adve V (2004) LLVM: a compilation framework for lifelong program analysis & transformation. In: International symposium on code generation and optimization, 2004. CGO 2004. IEEE 20 Mar 2004, pp 75–86
17. Cytron R, Ferrante J, Rosen BK, Wegman MN, Zadeck FK (1989) An efficient method of computing static single assignment form. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on principles of programming languages, 3 Jan 1989, pp 25–35
18. Pattabiraman K, Kalbarczyk Z, Iyer RK (2007) Automated derivation of application-aware error detectors using static analysis. In: 13th IEEE international on-line testing symposium (IOLTS 2007). IEEE, 8 Jul 2007, pp 211–216
19. Hank RE, Wen-mei WH, Rau BR (1997) Region-based compilation: introduction, motivation, and initial experience. Int J Parallel Prog 25(2):113–146
20. Gaisler J, Isomäki M (2006) LEON3 GR-XC3S-1500 Template Design. Copyright Gaisler Res 1–53
21. Lyle G, Chen S, Pattabiraman K, Kalbarczyk Z, Iyer R (2009) An end-to-end approach for the automatic derivation of application-aware error detectors. In: 2009 IEEE/IFIP international conference on dependable systems & networks. IEEE, 29 Jun 2009, pp 584–589
22. Lu Q, Farahani M, Wei J, Thomas A, Pattabiraman K (2015) Llfi: an intermediate code-level fault injection tool for hardware faults. In: 2015 IEEE international conference on software quality, reliability and security. IEEE, 3 Aug 2015, pp 11–16
23. Perry F, Mackey L, Reis GA, Ligatti J, August DI, Walker D (2007) Fault-tolerant typed assembly language. In: Proceedings of the 28th ACM SIGPLAN conference on programming language design and implementation, 10 Jun 2007, pp 42–53
24. Clavel M, Durán F, Hendrix J, Lucas S, Meseguer J, Ölveczky P (2007) The Maude formal tool environment. In: International conference on algebra and coalgebra in computer science. Springer, Berlin, Heidelberg, 20 Aug 2007, pp 173–178

25. Pattabiraman K, Nakka N, Kalbarczyk Z, Iyer R (2008) SymPLFIED: symbolic program-level fault injection and error detection framework. In: 2008 IEEE international conference on dependable systems and networks with FTCS and DCC (DSN). IEEE, 24 Jun 2008, pp 472–481
26. Belkin VV, Yanovsky FJ (2007) Aircraft collision avoidance system. IEEE Aerospace Conference 2007:1–9. https://doi.org/10.1109/AERO.2007.352730
27. Pattabiraman K, Healey W, Yuan F, Kalbarczyk Z, Iyer RK (2009) Insider attack detection by information-flow signature enforcement. Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep
28. Pattabiraman K, Nakka N, Kalbarczyk Z, Iyer R (2009) Discovering application-level insider attacks using symbolic execution. In: IFIP international information security conference. Springer, Berlin, Heidelberg, 18 May 2009, pp 63–75
29. Yuan FQ (2010) Formal framework and tools to derive efficient application-level detectors against memory corruption attacks
30. Jacques-Silva G, Kalbarczyk Z, Gedik B, Andrade H, Wu KL, Iyer RK (2011) Modeling stream processing applications for dependability evaluation. In: 2011 IEEE/IFIP 41st international conference on dependable systems & networks (DSN), pp 430–441
31. Rashid L, Pattabiraman K, Gopalakrishnan S (2012) Dieba: Diagnosing intermittent errors by backtracing application failures. Silicon Errors in Logic-Syst Eff
32. Dadashi M, Rashid L, Pattabiraman K, Gopalakrishnan S (2014) Hardware-software integrated diagnosis for intermittent hardware faults. In: 2014 44th Annual IEEE/IFIP international conference on dependable systems and networks. IEEE, 23 Jun 2014, pp 363–374
33. Wei J, Pattabiraman K (2012) BLOCKWATCH: leveraging similarity in parallel programs for error detection. In: IEEE/IFIP international conference on dependable systems and networks (DSN 2012). IEEE, 25 Jun 2012, pp 1–12
34. Lu Q, Pattabiraman K, Gupta MS, Rivers JA (2014) SDCTune: a model for predicting the SDC proneness of an application for configurable protection. In: Proceedings of the 2014 international conference on compilers, architecture and synthesis for embedded systems 12 Oct 2014, pp 1–10
35. Wei J, Thomas A, Li G, Pattabiraman K (2014) Quantifying the accuracy of high-level fault injection techniques for hardware faults. In: 2014 44th Annual IEEE/IFIP international conference on dependable systems and networks. IEEE, 23 Jun 2014, pp 375–382
36. Li G, Pattabiraman K, Cher CY, Bose P (2016) Understanding error propagation in GPGPU applications. In: SC'16: Proceedings of the international conference for high performance computing, networking, storage and analysis. IEEE, 13 Nov 2016, pp 240–251
37. Fang B, Lu Q, Pattabiraman K, Ripeanu M, Gurumurthi S (2016) ePVF: an enhanced program vulnerability factor methodology for cross-layer resilience analysis. In: 2016 46th Annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 28 Jun 2016, pp 168–179
38. Li G, Pattabiraman K, Hari SK, Sullivan M, Tsai T (2018) Modeling soft-error propagation in programs. In: 2018 48th Annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 25 Jun 2018, pp 27–38
39. Li G, Pattabiraman K (2018) Modeling input-dependent error propagation in programs. In: 2018 48th Annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 25 Jun 2018, pp 279–290
40. Anwer AR, Li G, Pattabiraman K, Sullivan M, Tsai T, Hari SK (2020) GPU-trident: efficient modeling of error propagation in GPU programs. In: SC20: International conference for high performance computing, networking, storage and analysis. IEEE, 9 Nov 2020, pp 1–15

Mining Dependability Properties from System Logs: What We Learned in the Last 40 Years

Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia

Abstract System logs have been extensively used over the past decades to gain insight about dependability properties of computer systems. Log files contain textual information about regular and anomalous events detected by a system under real workload conditions. By mining the information contained in the logs it is possible to characterize the real failure behavior of the system. By real, we mean considering only the failures that manifest naturally, during system operation. This chapter provides an overview of the main tools and techniques for log-based failure analysis, which have been proposed in the last four decades. By surveying the relevant work in the area, the chapter highlights the main objectives, research trends and applications, and it also discusses the main limitations and recent proposals to improve log-based failure analysis.

Keywords Event logs · Log processing · Failure analysis · Dependability evaluation

1 Introduction

Failure analysis is the process that aims to determine and to analyze the causes of system failure. It consists in collecting and analyzing failure-related data during system operation, i.e., data generated by a computer system under real workload conditions. This approach allows us to evaluate the dependability properties and the failure modes of a system in a precise way. **Event logs**, or simply *logs*, are an important source of failure data [25, 43] because they store textual information

M. Cinque (✉) · D. Cotroneo
Universitá degli Studi di Napoli Federico II, Naples, Italy
e-mail: macinque@unina.it

D. Cotroneo
e-mail: cotroneo@unina.it

A. Pecchia
Universitá degli Studi del Sannio, Benevento, Italy
e-mail: antonio.pecchia@unisannio.it

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_12

```
1 [May-12-2011 09:30:11] get manager reference from local daemon
2 [May-12-2011 09:30:11] supervision property change event notifier
3 [May-12-2011 09:30:12] event notifier is running
4 ... omissis ...
5 [May-12-2011 09:35:22] exception raised by process 'P1', PID 2264
6 [May-12-2011 09:35:23] error: managed process 'P1', PID 2264 aborted
7 ... omissis ...
8 [May-12-2011 09:38:46] created DS Sys_Converter
9 [May-12-2011 09:38:46] created SystemAccessor
```

Fig. 1 Example of entries in the event log

about regular and anomalous events detected by a system during execution. Logs have been successfully used by industry and academia for failure analysis in a variety of application domains. A non-exhaustive list includes, for example, operating systems [42, 54], control systems and mobile devices [8, 31], supercomputers [33, 46], and large-scale applications [44, 51].

Accuracy of log-based failure analysis is intertwined with the ability of inferring meaningful information from raw logs, which is a challenging task. Logs usually contain large volumes of data consisting of sequences of **text entries**, i.e., lines, produced by a variety of computing entities (e.g., operating system modules and daemons, middleware supports, application components). Figure 1 reports an example of entries taken from a real event log. Entries provide a *timestamp*, i.e., the time the event has been logged, and a text *message* describing the event. Entries may contain further data, such as the *source* (e.g., the generating process) and the *severity* (i.e., the criticality of the notification). Example shows that logs have a subjective [29] and unstructured [34] nature. More importantly, logs contain (i) many entries that are not useful for failure analysis, (ii) redundant notifications caused by error propagation phenomena [22], and (iii) different formats for similar notifications produced by different components. Overall these issues make log analysis a hard process.

Log-based failure analysis usually encompasses three steps, i.e., *Log collection and selection*, *Pre-processing*, and *Analysis*. Each of the steps is important to the objective of obtaining accurate dependability measures [2, 27]. Figure 2 depicts the overall process. This chapter describes well-established techniques supporting each of the mentioned steps, and it discusses main applications of log-based failure analysis (e.g., error/failure classification, evaluation of dependability attributes, error propagation, improvement of the logging practice and applications to security analysis) by surveying relevant work in the area. Discussion reveals benefits and potential of logs for the quantitative evaluation of complex systems. Moreover, it provides system engineers and dependability analysts with a concrete workflow to conduct log-based failure analysis campaigns.

The rest of the chapter is organized as follows. Section 2 describes main data collection protocols and tools. Section 3 presents manipulation strategies that are commonly adopted to infer the failure data from the log. Section 4 discusses analysis and relevant applications of event logs, whereas Sect. 5 concludes the chapter with final remarks and discussion on limitations of log analysis.

2 Overview of Data Collection Tools and Products

Modern organizations rely on a variety of data sources—such as, system and application event logs, resource monitoring, network audit and intrusion detection systems—to develop situational awareness. Runtime data from production systems represent a goldmine of information to detect errors, failures and incidents, to understand their impact, and to gain insights for improving resiliency and countermeasures. In the following we present some examples of relevant tools and products for data collection.

As said, the use of **event logs** has been known since the early days of computers. Logs consist of semi structured text lines generated at run-time by specific instructions intertwined with the business code; lines in the logs account for dump of key variables and data structures, execution tracing and event reporting. The placement of the logging code is an empirical procedure [48]; logging code is based on generic output functions, proprietary supports or standard logging library. For example, **UNIX syslog** [36] defines a log format and collection protocol that has become a *de facto* standard over the years. A "syslog" log line is characterized by *severity* and *facility* (i.e., the indication of the source of the event, such as kernel or the security subsystem.) that can be combined to define the priority of the message. Severity varies in the interval $\{0, \ldots, 7\}$, with `0` representing an *emergency*-level entry, down to `7`, reporting a *debug* entry. A configuration file, namely, `/etc./syslog.conf`, allows specifying how to manage the events. **Microsoft Event Log**[1] protocol is another example of a log-collection system. Each Windows machine runs an Event Log provider that is accessible by means of system calls. Once an event is logged, it can be stored in a log file and forwarded to a remote machine. Another popular framework is Apache Software Foundation's **log4x**.[2] The framework is available for C++, PHP, Java and .NET applications, and it can be configured in terms of syntax of log messages, e.g., to support automatic parsing of the entries, and destination.

Event logs gathered in production environments are typically supplemented by measurements and traces generated by specialized **monitoring tools**. At the time being, there exist a wide and increasing range of monitoring products. For example. **Sysdig**[3] allows collecting resource usage, network statistics, as well as tracing applications. **Ganglia**[4] is a scalable distributed system monitoring tool for high-performance computing systems such as clusters and grids. It supports remote visualizations and live statistics, such as CPU or network load. **Nagios**[5] is an open source software to monitor network services and resources of a given set of nodes. The key advantage of Nagios are *per node* data collection and convenient graphical interfaces. More recently, there has been a growing interest in Application Performance Management (APM); for example, Dynatrace, AppDynamics, CA and New Relic are

[1] https://docs.microsoft.com/en-us/windows/win32/eventlog/event-logging.

[2] http://logging.apache.org/.

[3] https://sysdig.com/opensource/.

[4] http://ganglia.sourceforge.net/.

[5] https://www.nagios.org/.

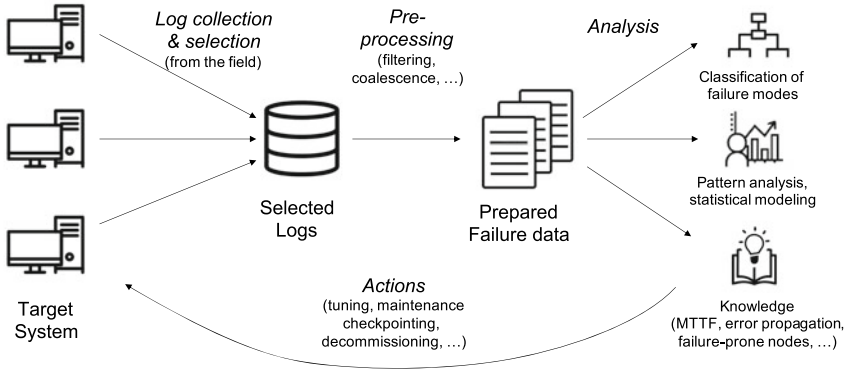


Fig. 2 The Log-based failure analysis process

well-known commercial tools that leverage source code instrumentation for monitoring purposes. **Zipkin**[6] is the open-source distributed tracer developed by Twitter in the context of microservices.

With the increasing spread and sophistication of security attacks, **intrusion detection systems** (IDS) are extensively used to collect data and alerts pertaining to suspicious activities. For example, **Snort**[7] is an open-source network-based intrusion detection and prevention system. It performs real-time analysis of network traffic and relies on a database of signatures of known attacks. **Suricata**[8] is another open-source real- time signature and rule-based intrusion detection and prevention system. **OSSEC**[9] is an open source data monitoring solution that allows configuring incidents that administrators wish to be alerted on. It mixes security aspects ranging from intrusion detection to log monitoring and provides built-in alerting for a variety of well-established data sources and protocols.

Due to the number of independent tools available within large-scale organizations, **Security Information and Event Management** (SIEM) is regarded as the state-of-the-practice to address the complexity underlying the collection and normalization of diverse data sources. SIEM products centralize to a single component real-time monitoring, correlation of events and alerting, long-term storage and analysis and compliance to regulations. There are commercial SIEM systems, such as IBM's QRadar, Splunk, LogRhythm and AlienVault Unified Security Management. In spite of the advances brought by SIEM at coping with the technical facets of collection, normalization and monitoring, data analysis still depends on many cognitive processes, such as to develop accurate alerting rules and for manual forensics activities.

[6] https://github.com/openzipkin/zipkin.

[7] https://www.snort.org/.

[8] https://suricata.io/.

[9] https://www.ossec.net/.

```
1 1167657137 n-238 +START HARDWARE ERROR STATE AT CPE
2 1167657137 n-238 +END HARDWARE ERROR STATE AT CPE
3 1167657137 n-238 +PCI Component Error Info Section
4 1167657140 n-238 +START HARDWARE ERROR STATE AT CPE
5 +214 omitted entries
```

Fig. 3 Example of failure notification in the log

3 Log Selection and Pre-processing

The first step in log-based failure analysis is represented by the selection and collection of log sources. Log Selection is the process where log relevant to the analysis task are retrieved from system components. Log data collection refers to those techniques to retrieve and to integrate data in specif nodes in charge of storing log data for the subsequent manipulation steps.

3.1 Log Pre-processing

Log pre-processing encompasses all the techniques that aim to transform the raw logs in a useful and efficient data format. As an example. logs report many non-error entries (such as lines 1, 2 in Fig. 1) that have to be excluded from the log before the analysis [22]. Several techniques are adopted to identify and remove irrelevant log entries. To this aim, well known filtering techniques can be used. For example, analysts might select entries of interest based on the *severity* field or focus on the entries containing error-specific keywords (e.g., pinpointed by means of regular expressions [59]). At a finer grain, filtering can be conducted with **de-parameterization**, which allows replacing variable fields in the text entries (e.g., usernames, IP and memory addresses, folders) with generic tokens. For example, the entries "`new connection from 192.168.0.184`" and "`new connection from 221.145.31.27`" would appear the same once the IP addresses are replaced with a generic "`IP-ADDR`" token. De-parameterization significantly reduces the number of distinct messages to scrutinize with the aim of identifying entries of interest. For examples, authors in [46] show that around 200 million entries in the log of a supercomputing system were generated by only 1124 distinct messages. De-parameterization can be supplemented by statistical approaches to faster the identification of the subset of entries that are useful to the analysis. Authors in [34] apply the Leveinshtein distance to cluster similar log messages. The work [60] presents a clustering algorithm and a tool for mining line patterns from the log.

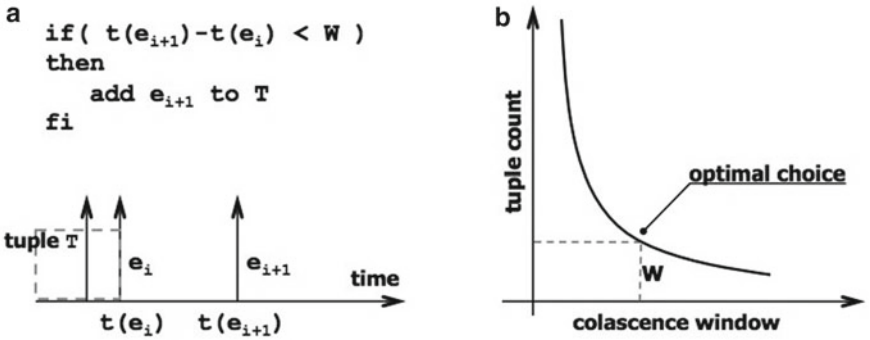


Fig. 4 Tuple heuristic: grouping condition (**a**); sensitivity analysis (**b**)

3.2 Coalescence

In practice, faults generate multiple errors (because of error propagation phenomena and the workload, which may trigger the same fault many times) and, consequently, multiple notifications in the log [22]. More in general, entries in the log that are correlated could belong to the same manifestation. Coalescence is one of the most used technique that groups different entries into the same failure point. Example reported in Fig. 3 shows that a single PCI card failure in a supercomputer log produced 218 entries. Coalescing related error entries is crucial to obtain realistic measurements [2, 27]. One of the most adopted coalescence strategies is the **tuple heuristic**. The intuition underlying the approach is that two entries in the log, if related to the same fault activation, are likely to occur close in time. Consequently, if the time distance of the entries is smaller than a predetermined threshold, i.e., the *coalescence window* (`W`), they are placed in the same group (called *tuple*). Figure 4a clarifies the concept. The value of the coalescence window is critical, because the number of tuples provides an approximation of the actual number of failures occurred at runtime. When `W` is too small, the risk is to put entries related to same problem into different tuples (truncation); viceversa, if `W` is too large, entries related to different problem might be placed in the same tuple (collision). A sensitivity analysis is conducted to investigate how the number of tuples (tuple count) varies when `W` varies. Figure 4b reports a generic plot to clarifying the output of such an analysis. Experimental studies assume that a good choice for the coalescence window is the value right after the "knee" of the curve, where the tuple count sharply flattens [22].

Several techniques have been proposed to improve the results of the tuple heuristic, i.e., to reduce the number of accidental collisions and truncations. For instance, in [58] two coalescence windows are adopted to improve grouping. A recent solution adopts spherical covariance estimates for the grouping of events in coalitions of clusters [17]. According to this scheme, two events of the same type are grouped if they fall within the typical *time to recover*. A different methodological improvement of the tuple heuristic is represented by the concept of *spatial coalescence* [33, 43]:

errors can propagate among the nodes of the system, and notifications related to the same fault manifestation might be spatially distributed as a result. Authors in [46] combine the temporal information of the entries with a statistical indicator to identify independent errors that occur close in time, reducing the incidence of accidental collisions. A rather different approach is *content-based coalescence*: in this case, events in the log are grouped based on the content of the text entries. For example, authors in [54] use a `perl` algorithm to identify OS reboots based on the sequential parsing of the log.

4 Analysis and Relevant Applications

In this section we discuss relevant applications and research trends in the area of log-based failure analysis. Works are grouped based on their main research objectives, such as event classification, failure modeling, event correlation, logging practice improvements and the use of logs for cybersecurity.

4.1 Error and Failure Classification

One of the primary outputs of a log-based failure analysis is the classification of error and failure modes, as they happen naturally, during system operation. Classification allows to determine the predominant classes of failure, to reveal what are the most failure-prone components, and to assess improvements between subsequent releases of the same software. Overall this information is valuable to drive quantitative evaluation and to supplement measurements. In the following, we present relevant examples of log-based failure analysis of different types of **target systems**.

A rather classical target system for classification studies is the **operating system**. For instance, authors in [30] present a study of a UNIX system. Analysis is based on an event log spanning around 11 months. Data in the log is classified and categorized to identify error trends preceding failures. For example, the study shows that the input-output subsystem is the most error-prone, which is still true in today's systems.

The study proposed by [55] provides a characterization of operating system reboots of Windows NT and 2K machines. The study focuses on unplanned reboots, representing the occurrence of a failure. A similar classification study is conducted in [18], which analyzes crash and usage data from Windows XP SP1 machines. These studies confirmed how the failures observed during operations are not caused by the operating system itself, but by applications, drivers, and third-party components.

More recently, due to the widespread use of mobile devices in our daily life, several research groups focused on classification the failures during the operation of **mobile operating systems**. In this case, the challenge is to perform the collection of failure data in a non-intrusive and efficient way, considering the constraints of mobile devices. A seminal work on Symbian OS [10] demonstrated that majority

of problems were due to memory access violation errors and heap management, while other studies [9, 38] confirmed that errors in applications still represent the main factor of failures, confirming the trend already observed in general purpose operating systems.

The shift of IT systems to service computing, cloud computing, and scientific data centers generated an increasing number of works focusing on log-based failure classification of **large-scale and distributed systems**, including supercomputers and server farms. In this case, the challenge of log analysis is to manage the high volume of events generated by the system at scale, combined with the difficulty to find actionable insight and real proof of failure in the vast amount of data. As a notable example, the work in [15] provides an analysis of failures and their impact for Blue Waters, the Cray hybrid (CPU/GPU) supercomputer at the University of Illinois at Urbana-Champaign, based on about 3.7 TB of automatically collected syslogs. Results were useful to understand that processor and memory protection mechanisms (x8 and x4 Chip kill, ECC, and parity) are able to handle a sustained rate of errors as high as 250 errors/h while providing a coverage of 99.997% out of a set of more than 1.5 million of analyzed errors. Software, on the other hand, was the largest contributor to the node repair hours (53%), with a total of 29 out of 39 system-wide outages involving the Lustre distributed file system. Authors in [43] present a pioneer work on the analysis of supercomputer logs, highlighting the challenges and somehow also the inadequacy, at that time, of logs for failure classification in such large scale systems, by comparing the results achieved by analysis of 5 supercomputers.

More recent works on large-scale distributed systems started to look into the architectural features of the target system, to better infer the relationships between failure events and the operational context by using system monitoring and software execution tracing. For instance, many solutions have emerged for monitoring *microservices*- and *containers*-based systems. As an example, the work [40] presents a dashboard for monitoring and managing microservices, characterized by a Spring-based infrastructure that uses Dynatrace.[10] The infrastructure allows collecting both failure rate and response time of each microservice; however, these types of approach require the instrumentation of the code to be monitored. A different idea is to accompany microservices logs with black box tracing of service invocations [12], to infer the execution context and to help practitioners in making informed decisions for troubleshooting and failure classification.

4.2 Dependability Modeling

Modeling the failure behavior of the target system is one of the main aims of log-based failure analysis. By using failure data, practitioners can infer error patterns and statistical distributions, can measure relevant attributes, such as mean time to failure

[10] https://www.dynatrace.com/.

(MTTF), mean time to recover (MTTR), and availability, and/or, more in general, can extract *knowledge* about the most failure prone components, error propagation phenomena between them, etc.

The *time to failure* variable is often modeled by fitting failure points extracted from logs, and related timestamps, with statistical distributions.

For instance, authors in [39] model the *time to failure* by adopting a hyper-exponential distribution, i.e., $\sum_{i=1}^{N} \lambda_i e^{-\lambda_i t} p_i$. This distribution is used to model failures that represent the manifestation of independent and alternate underlying causes. Distribution used in the study allowed inferring the existence of two separate recovery paths selected in a fixed ratio, resulting from two different classes of software failures.

Other statistical distributions used to model the time to failure are the lognormal and weibull. The lognormal distribution can be used when the value of a variable can be determined by the multiplication of many random factor, which is the case for software failures. For instance, in [41] the author hypothesizes that the failure rate of a complex system can be tough as a multiplicative process of independent factors, e.g., activations of faults, using then a lognormal distribution. Similar considerations are provided in [51], in the context of high-performance computing systems. The Weibull distribution, i.e., $e^{-(\lambda t)^{\alpha}}$ [35], is the most adopted to model the failure data [29, 35, 39], since the value of the shape parameter α allows modeling decreasing ($\alpha < 1$), increasing ($\alpha > 1$), and constant ($\alpha = 1$), failure hazard rates.

Markov chains, Petri nets (and their extensions such as stochastic activity networks) and finite state machines are also used to model the failure behavior. For instance, in [29] authors use log data to propose a finite state machine to model the error behavior and the availability of a LAN of Windows NT machines. The adoption of the model showed that, even if the measured system availability was around 99%, the user-perceived availability was significantly smaller, i.e., 92%: in some cases, even if a machine of the LAN was up, it was not able to provide correct service to the user.

4.3 Failure Correlation and Error Propagation Analysis

Logs contain information about errors occurring in different components of the same target system as well as data about the execution context and the workload of the system under analysis. This characteristic has been exploited since the first studies on failure analysis to understand causes and **correlation** phenomena of system failures. Works in the area, dating back to the 1980s, prove the existence of a relationship between the failure behavior and the **workload** run by a system. A performance study of a DEC system conducted in [4, 5], showed that the failure rate is not constant; nevertheless, many models adopted at that time relied on such an assumption. A doubly stochastic Poisson model was developed to highlight the relationship between the instantaneous failure rate of a resource and its usage. A similar finding has been

confirmed by authors in [26], by evaluating the relationship between system load and failures by means of empirical data.

Several works suggest that failures observed in different system components are correlated. For example, [32] applies factor and cluster analysis to pinpoint halts dependencies among components and halt patterns from the log. Although the number of errors observed during the system operations was relatively small, authors demonstrated that multiple processes were affected by the same problem, because of the presence of shared resources. The study proposed in [28] uses statistical techniques to quantify the strength of the relationship among entries in the log. The approach aimed to discriminate transient, permanent and intermittent failure manifestations by assessing the correlation between failure events. Correlation of failure entries is also used to conduct **failure prediction**, as proposed for IBM BlueGene/L [33]. The approach proposed in the paper was able to predict around 80% of memory and network failures and 47% of I/O failures.

As already observed in Sect. 3.2, faults can generate multiple errors before a failure ultimately manifests in the system. This known behavior is exploited in **error propagation analysis** to infer error models, intermediate paths and effects that pertain to the activation of faults. Analysis of propagation allows inferring error-prone components and establishing *where-what* type of errors are likely to cause system-wide failures [1]. This is of great importance for practitioners, since this information provides actionable insight on where to place error detection and recovery mechanisms in the software under study, to improve the overall dependability. Several works in the area of error propagation analysis rely on the instrumentation of the code (either at source or binary level) to generate error traces upon fault activation. For instance, *PROPANE* [23] analyzes the propagation of data errors in single-process C software systems, and identifies error paths and propagation frequency. *PROPANE* is based on a fault injection approach to induce data errors in the system and *variable instrumentation* to detect errors. However, as observed in [45], performing fault-injection may be time-consuming and cumbersome for the developer. Therefore, it is desirable to develop an automated technique to derive and place detectors in application code. In [45] authors devise detectors to be placed in strategic locations in the code in an automated way without requiring programmer intervention or fault-injection into the system.

A different approach for error propagation analysis is to capitalize on the data already produced by the system, e.g., log files, without instrumentation. This is often the case pf production environments, or for systems based on proprietary components off-the-shelf, where there is limited knowledge on system internals. In [13] we proposed an approach to infer a representation, named *error reporting graphs*, of the errors leading from faults in a given component to failures. The graph is constructed by relying solely on the error data collected from the available logs. In particular, for a given error event, it is possible to obtain the *reporting stage*, that can assume one of the following vaules (adapted from [37]):

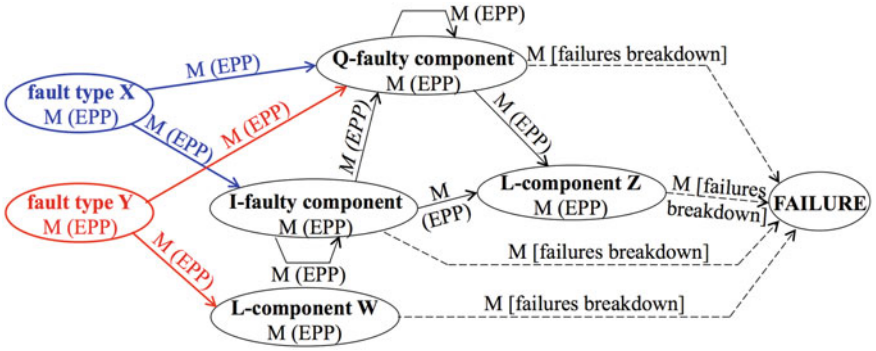


Fig. 5 Example of error reporting graph (from [13])

- **immediate (I)**: the subcomponent that reports the error is also the location of the fault;
- **quick (Q)**: the subcomponent that reports the error is not the location of the fault, although it belongs to the same component;
- **last (L)**: the subcomponent that reports the error is not the location of the fault and belongs to a different component.

Figure 5, taken from [13], shows an example of a general error reporting graph. The graph indicates both multiplicity (M) and Error Propagation Probability (EPP) of each node/arc. M is the number of log entries that contain that node/arc; the EPP of a node/arc is the ratio between the multiplicity of the node/arc and the number of failure data instances used to obtain the graph. So, starting from the fault, it is possible to reconstruct the path leading to failure, and the traversed components classified as immediate, quick, and last. The analysis of the graph allows to understand what are the most error prone components where to place error detection and recovery mechanisms. For better readability, the graph encompasses one FAILURE node, while the failure types are shown on the arcs connected to the FAILURE node (failures breakdown in the figure).

4.4 Improvement of the Logging Practice

The literature discussed up to this point witnesses how the analysis of event logs has been beneficial over the last decades for several types of systems and applications, ranging from failure classification to error propagation. All these studies have however one major problem in common: performed analysis and even the conclusions that can be drawn depend heavily on the quality of the log lines that can be collected from the target system.

Log lines are produced by the so-called *logging mechanism*, that is, the set of logging points and activation code of such logging points (if any) implemented in the source code of the target system.

The implementation of the logging mechanism is a human-driven and empirical practice [48]. Logging points can be missing or subject to erroneous activation conditions, leading to wrong or missing lines in the logs. The result is that logs may report error entries that do not necessarily represent real system failures, or, even worse, they might completely miss information about failures that happened during operation. For instance, in our earlier study [7], we demonstrated that around 60% of failures due to software faults do not leave any trace in the logs. In addition, in [48], from the analysis of the source code of large industrial projects, we observed how no strict rules exist, across different product lines of the same company, for the implementation of the logging points, even if logging constitutes a non-negligible portion of the source code (around 3.5% of the code).

Moved by these problems, researchers and practitioners in log-based failure analysis started to look at methods and techniques to produce better and more meaningful logs. Accurate logs can improve the work of the analyst, allowing to achieve rooted conclusions on the behavior of the observed system.

For instance, authors in [50], introduce a set of recommendations to improve the expressiveness of the logs, such as structuring the messages as key/value pairs or making explicit the type of the values in the log entries. Similarly, [64] proposes to enhance the logging code by adding information, e.g., *data values*, to ease the diagnosis task in case of failures. These works improve the invocations of logging functions that *already exist* in the software platform. Nevertheless, incompleteness of the logs cannot be solved acting solely on the existing functions: developers might forget to log significant events, and, in many cases, errors escape existing logging points.

For these reasons, other techniques have been adopted to detect and analyze failures, such as runtime failure detection, executable assertions, or software tracing. Runtime failure detection consists in observing, either locally or remotely, the execution state of the system [14, 61]. Executable assertions, usually adopted in the embedded systems domain [24], are check statements performed on the program variables to detect application-specific content errors, e.g., invalid variable values for the given function. Software tracing solutions [3] are widely adopted to monitor the execution of a software system, e.g., to perform Function Boundary Tracing (FBT), which register function entry and exit events. They rely on software instrumentation packages, such as DTrace. Along this direction, we proposed the *rule-based logging* approach [11], which leverages system design artifacts to define a model encompassing errors leading to failures. A set of *rules* establishes how the logging mechanism must be placed in the source code to detect such errors with high accuracy.

4.5 Security Analysis

System logs collected during the progression of malicious activities and incidents can be used for **security analysis**. We discuss some relevant applications across different application domains.

Large-scale systems and organizations. A cloud-based distributed and parallel security log analysis framework for organizational security is presented in [53]. The framework supports the analysis of system, network, and transaction logs by using a two-level master-slave architecture and streaming analysis features. A prototype, which analyzes HTTP request logs, has been implemented in the Amazon cloud environment, in order to detect HTTP sessions with blacklisted destination IP addresses. The work [63] proposes an approach that analyzes logs collected from various network devices. (i.e., web proxies, DHCP and VPN servers, Windows domain controllers, antivirus software) to detect malicious activity. The approach—called *Beehive*—consists of a parsing/filter/normalizing step, a feature-generator and a detector. The Authors have evaluated Beehive on a large set of real-world enterprise log data, and demonstrated that it improves on signature-based approaches to detecting security incidents. Authors in [52] use real forensic data on security incidents that occurred over a period of 5 years at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, USA. The proposed methodology combines automated analysis of data from security monitors and system logs with human expertise to extract and process relevant data to: (i) determine the progression of an attack, (ii) establish incident categories and characterize their severity, (iii) associate alerts with incidents, and (iv) identify incidents missed by the monitoring tools and examine the reasons for the escapes. Based on the same NCSA dataset, [47] proposes a Bayesian network approach to detect credential stealing incidents.

Critical information systems. Event logs for security analysis have been used for critical information systems protection. For example, [56] presents a Supervisory Control And Data Acquisition (SCADA) security framework for protecting electric power infrastructures. The framework consists of real-time monitoring and anomaly detection components, which rely on different sources, such as security, system, and file integrity logs. Authors in [21] propose *MELISSA*, i.e., a tool for processing SCADA logs to detect process-related threats. MELISSA relies on pattern mining to identify the most and the least frequent (expected to be anomalous) patterns of system behaviors. A framework for Situational Awareness of Critical Infrastructure and Networks (*SACIN*) is presented in [57]. The framework gathers data, e.g., industrial automation systems and intrusion detection systems logs, from different entities of a given system, and uniforms their format for subsequent analysis.

Recent trends in security analysis. With rapid growth of volume and variety of runtime data, security data analysis has been addressed by means of standard **Big Data frameworks**. Authors in [19] propose a method for the detection of Advanced Persistent Threat (APT), which uses MapReduce to analyze security events from different log sources, such as Virtual Private Network, Intrusion Detection System and firewall. The approach relies on a signature database with *known bad* information.

The work [20] proposes an approach to analyze large logs for detecting host misbehavior. The approach combines data mining and supervised/unsupervised machine learning to automate the detection. The approach uses DHCP servers, authentication servers, and firewall logs as data sources and adopts Hadoop MapReduce and involves: (i) data mining to extract a set of features from the logs, (ii) clustering to create sets of hosts with similar behaviors, and (iii) linear classification to detect misbehaving sets of hosts. More recently, security log analysis has been addressed by means of **deep learning** techniques and frameworks. The approach in [16], named *DeepLog*, uses a deep neural network to model a system log as a natural language sequence. DeepLog learns patterns from normative executions in order to detect anomalies. In [62] is proposed *nLSALog*, an anomaly detection framework that leverages log files as data source. The framework models the log as a natural language sequence and uses Long Short-Term Memory (LSTM), built using nominal training data, to detect security anomalies. Finally, *ADA* (Adaptive Deep Log Anomaly Detection) [65] allows the detection of security-related anomalies in system logs, leveraging deep neural networks with LSTM and dynamic adaptive thresholds.

It is worth noting that research advances in the area of security analysis suffer from the scarceness of real-life data gathered during spontaneous (i.e., neither induced nor simulated) incidents and attacks in production environments. This is due to confidentiality restrictions and non-disclosure policies by industry vendors and information system providers. Many studies in this area leverage honeypots, "lab-made" intrusions and ready-to-use public datasets, such as KDD-CUP'99, UNSW-NB15, CICIDS2017 and many more. As a matter of fact these datasets, surveyed in [49], have become common benchmarks for intrusion detection.

5 Conclusion and Final Remarks

Log lines emitted by computer systems provide actionable information that allows understanding the effect of errors on the system behavior. It provides accurate information on the system being observed, for the elaboration and validation of analytical models, and for the improvement of the development process. The collected data help to explain and to characterize the system under study. Qualitative analysis of failure, error and fault types observed in the field yields feedback to the development process and can thus contribute to improving the production process [6]. As stated in [25], *"there is no better way to understand dependability characteristics of computer systems than by direct measurements and analysis"*.

The variety of concrete applications discussed in this chapter shows that logs are extremely valuable to dependability engineers. Notwithstanding its practical usefulness, it must be noted that analysis is limited to manifested failures, that is, the ones reported by the log. Several types of failures, such as application crashes or hangs, can escape logging mechanisms and go unreported. Further research has been devoted to the improvement of the logging mechanism, to produce more meaningful and ready-to-use logs. Nevertheless, log production is still largely overlooked in soft-

ware industries, and left to the late stages of development for testing and debugging. More research is needed to come to common and well understood practices. The same applies when logs are used for security analysis, where the scarceness of real-life data represent the current barrier for the large application of latest advances, e.g., based on deep learning.

Finally, it should be noted that the specific conditions under which the system is observed can vary from an installation to another. Doubts can be raised on the validity of obtained results across different installations of the same system. Log-based failure analysis is particularly useful for stable installations, such as critical embedded systems, signal processing equipment, and long-running server systems. In these systems, dependability needs to be analyzed in order to be continuously improved. On this last point, it is worth noting that log analysis is partially beneficial to current system installations, while it provides crucial guidelines to improve successive releases.

References

1. Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. Dependable Secure Comput IEEE Trans 1(1):11–33. https://doi.org/10.1109/TDSC.2004.2
2. Buckley M, Siewiorek D (1996) A comparative analysis of event tupling schemes. In: Proceedings of the international symposium on fault-tolerant computing (FTCS). IEEE Comput Soc
3. Cantrill B, Shapiro MW, Leventhal AH (2004) Dynamic instrumentation of production systems. USENIX Ann Tech Conf
4. Castillo X, Siewiorek D (1980) A performace-reliability model for computing systems. In: Proceeding of the international symposium on fault-tolerant computing (FTCS). IEEE Computer Society
5. Castillo X, Siewiorek D (1981) Workload, performance, and reliability of digital computing systems. In: Proceedings of the international symposium on fault-tolerant computing (FTCS). IEEE Computer Society
6. Chillarege R, Iyer RK, Laprie JC, Musa JD (1993) Field failures and reliability in operation. In: Proceeding of the 4th IEEE international symposium on software reliability engineering
7. Cinque M, Cotroneo D, Natella R, Pecchia A (2010) Assessing and improving the effectiveness of logs for the analysis of software faults. In: Proceeding international conference on dependable systems and networks (DSN). IEEE Computer Society
8. Cinque M, Cotroneo D, Russo S (2006) Collecting and analyzing failure data of bluetooth personal area networks. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
9. Cinque M (2011) Enabling on-line dependability assessment of android smart phones. In: 2011 IEEE/IFIP 41st international conference on dependable systems and networks workshops (DSN-W), pp 286–291. https://doi.org/10.1109/DSNW.2011.5958783
10. Cinque M, Cotroneo D, Kalbarczyk Z, Iyer RK (2007) How do mobile phones fail? a failure data analysis of symbian os smart phones. In: 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07), pp 585–594. https://doi.org/10.1109/DSN.2007.54
11. Cinque M, Cotroneo D, Pecchia A (2013) Event logs for the analysis of software failures: a rule-based approach. IEEE Trans Softw Eng 39(6):806–821. https://doi.org/10.1109/TSE.2012.67

12. Cinque M, Della Corte R, Pecchia A (2019) Microservices monitoring with event logs and black box execution tracing. IEEE Trans Serv Comput pp 1–1. https://doi.org/10.1109/TSC.2019.2940009
13. Cinque M, Della Corte R, Pecchia A (2020) An empirical analysis of error propagation in critical software systems. Emp Softw Eng 25:2450–2484. https://doi.org/10.1007/s10664-020-09801-2
14. David FM, Carlyle JC, Campbell RH (2007) Exploring recovery from operating system lockups. In: ATC'07: 2007 USENIX annual technical conference on proceedings of the USENIX annual technical conference. USENIX Association, Berkeley, CA, USA, pp 1–6
15. Di Martino C, Kalbarczyk Z, Iyer RK, Baccanico F, Fullop J, Kramer W (2014) Lessons learned from the analysis of system failures at petascale: the case of blue waters. In: 2014 44th annual IEEE/IFIP international conference on dependable systems and networks, pp 610–621. https://doi.org/10.1109/DSN.2014.62
16. Du M, Li F, Zheng G, Srikumar V (2017) DeepLog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 1285–1298. ACM, New York, NY, USA. https://doi.org/10.1145/3133956.3134015
17. Fu S, Xu C (2007) Exploring event correlation for failure prediction in coalitions of clusters. In: SC '07: Proceeding of the 2007 ACM/IEEE conference on supercomputing
18. Ganapathi A, Patterson DA (2005) Crash data collection: a windows case study. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
19. Giura P, Wang W (2012) Using large scale distributed computing to unveil advanced persistent threats. Science J 1(3):93–105
20. Gonçalves D, Bota J, Correia M (2015) Big data analytics for detecting host misbehavior in large logs. In: Trustcom/BigDataSE/ISPA, 2015 IEEE. vol 1, pp 238–245. https://doi.org/10.1109/Trustcom.2015.380
21. Hadiosmanovic D, Bolzoni D, Hartel P, Etalle S (2011) Melissa: towards automated detection of undesirable user actions in critical infrastructures. In: 2011 Seventh European conference on computer network defense (EC2ND). pp 41–48. https://doi.org/10.1109/EC2ND.2011.10
22. Hansen JP, Siewiorek DP (1992) Models for time coalescence in event logs. In: Proceedings of the international symposium on fault-tolerant computing (FTCS). IEEE Computer Society
23. Hiller M, Jhumka A, Suri N (2002) Propane: an environment for examining the propagation of errors in software. In: Proceedings of the 2002 ACM SIGSOFT international symposium on software testing and analysis, pp 81–85. ISSTA, ACM, New York, NY, USA. https://doi.org/10.1145/566172.566184, http://doi.acm.org/10.1145/566172.566184
24. Hiller M (2000) Executable assertions for detecting data errors in embedded control systems. In: Proceeding of the 2000 international conference on dependable systems and networks. DSN '00
25. Iyer RK, Kalbarczyk Z, Kalyanakrishnan M (2000) Measurement-based analysis of networked system availability. Performance evaluation: origins and directions
26. Iyer RK, Rossetti DJ, Hsueh MC (1986) Measurement and modeling of computer reliability as affected by system activity. ACM Trans Comput Syst 4
27. Iyer RK, Young LT, Sridhar V (1986) Recognition of error symptoms in large systems. In: Proceedings of 1986 ACM fall joint computer conference. ACM '86
28. Iyer R, Young L, Iyer P (1990) Automatic recognition of intermittent failures: an experimental study of field data. IEEE Trans Comput
29. Kalyanakrishnam M, Kalbarczyk Z, Iyer RK (1999) Failure data analysis of a LAN of windows NT based computers. In: Proceedings of the international symposium on reliable distributed systems (SRDS). IEEE Computer Society
30. Lal R, Choi G (19998) Error and failure analysis of a UNIX server. In: IEEE international symposium on high-assurance systems engineering
31. Laplace J, Brun M (1999) Critical software for nuclear reactors: 11 years of field experience analysis. In: Proceedings of the international symposium on software reliability engineering (ISSRE). IEEE Computer Society

32. Lee I, Iyer R, Tang D (1991) Error/failure analysis using event logs from fault tolerant systems. In: Proceedings of the international symposium on fault-tolerant computing (FTCS). IEEE Computer Society
33. Liang Y, Zhang Y, Sivasubramaniam A, Jette M, Sahoo RK (2006) BlueGene/L failure analysis and prediction models. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
34. Lim C, Singh N, Yajnik S (2008) A log mining approach to failure analysis of enterprise telephony systems. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
35. Lin TT, Siewiorek D (1990) Error log analysis: statistical modeling and heuristic trend analysis. IEEE Trans Reliablity
36. Lonvick C (2001) The BSD syslog protocol. In: Request for comments 3164, The Internet Society, Network Working Group, RFC3164
37. Lyu MR et al (1996) Handbook of software reliability engineering, vol 222. IEEE Computer Society Press CA
38. Maji AK, Hao K, Sultana S, Bagchi S (2010) Characterizing failures in mobile oses: a case study with android and symbian. In: 2010 IEEE 21st international symposium on software reliability engineering, pp 249–258. https://doi.org/10.1109/10.1109/ISSRE.2010.45
39. Matz S, Votta L, Makawi M (2002) Analysis of failure recovery rates in a wireless telecommunication system. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
40. Mayer B, Weinreich R (2017) A dashboard for microservice monitoring and management. In: Proceeding international conference on software architecture workshops. IEEE, pp 66–69. https://doi.org/10.1109/ICSAW.2017.44
41. Mullen R (1998) The lognormal distribution of software failure rates: origin and evidence. In: Proceedings of the international symposium on software reliability engineering (ISSRE). IEEE Computer Society
42. Murphy B, Levidow B (2000) Windows 2000 dependability. In: MSR-TR-2000-56 Technical Report. Redmond, WA
43. Oliner AJ, Stearley J (2007) What supercomputers say: a study of five system logs. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
44. Oppenheimer DL, Ganapathi A, Patterson DA (2003) Why do internet services fail, and what can be done about It? In: USENIX symposium on internet technologies and systems (2003). http://www.usenix.org/events/usits03/tech/oppenheimer.html
45. Pattabiraman K, Saggese GP, Chen D, Kalbarczyk Z, Iyer R (2011) Automated derivation of application-specific error detectors using dynamic analysis. IEEE Trans Dependable Sec Comput 8(5):640–655. https://doi.org/10.1109/TDSC.2010.19
46. Pecchia A, Cotroneo D, Kalbarczyk Z, Iyer RK (2011) Improving log-based field failure data analysis of multi-node computing systems. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
47. Pecchia A, Sharma A, Kalbarczyk Z, Cotroneo D, Iyer RK (2011) Identifying compromised users in shared computing infrastructures: a data-driven Bayesian network approach. In: Proceedings of the international symposium on reliable distributed systems (SRDS). IEEE Computer Society
48. Pecchia A, Cinque M, Carrozza G, Cotroneo D (2015) Industry practices and event logging: assessment of a critical software development process. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering. vol 2, pp 169–178. https://doi.org/10.1109/ICSE.2015.145
49. Ring M, Wunderlich S, Scheuring D, Landes D, Hotho A (2019) A survey of network-based intrusion detection data sets. Comput Secur 86:147–167
50. Salfner F, Tschirpke S, Malek M (2004) Comprehensive logfiles for autonomic systems. In: Proceeding IEEE parallel and distributed processing symposium (IPDPS)

51. Schroeder B, Gibson GA (2006) A large-scale study of failures in high-performance computing systems. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
52. Sharma A, Kalbarczyk Z, Barlow J, Iyer R (2011) Analysis of security data from a large-scale organization. In: Proceedings of the international conference on dependable systems and networks (DSN). IEEE Computer Society
53. Shu X, Smiy J, Yao DD, Lin H (2013) Massive distributed and parallel log analysis for organizational security. In: 2013 IEEE globecom workshops (GC Wkshps), pp 194–199. https://doi.org/10.1109/GLOCOMW.2013.6824985
54. Simache C, Kaâniche M (2005) Availability assessment of SunOS/Solaris unix systems based on syslogd and wtmpx log files: a case study. In: Pacific rim international symposium on dependable computing (PRDC). IEEE Computer Society (2005). http://doi.ieeecomputersociety.org/10.1109/PRDC.2005.20
55. Simache C, Kaaniche M, Saidane A (2002) Event log based dependability analysis of windows NT and 2K systems. In: Pacific rim international symposium on dependable computing (PRDC). IEEE Computer Society
56. Ten CW, Manimaran G, Liu CC (2010) Cybersecurity for critical infrastructures: attack and defense modeling. IEEE Transactions on systems, man, and cybernetics—part A: systems and humans 40(4):853–865. https://doi.org/10.1109/TSMCA.2010.2048028
57. Timonen J, Lääperi L, Rummukainen L, Puuska S, Vankka J (2014) Situational awareness and information collection from critical infrastructure. In: 2014 6th international conference on cyber conflict (CyCon 2014), pp 157–173. https://doi.org/10.1109/CYCON.2014.6916401
58. Tsao MM, Siewiorek DP (1983) Trend analysis on system error files. In: Thirteenth annual international symposium on fault-tolerant computing
59. Vaarandi R (2002) SEC—A lightweight event correlation tool. In: Proceedings of 2002 IEEE workshop on IP operations and management (IPOM)
60. Vaarandi R (2003) A data clustering algorithm for mining patterns from event logs. In: Proceedings of 2003 IEEE workshop on IP operations and management (IPOM)
61. Wang L, Kalbarczyk Z, Gu W, Iyer R (2007) Reliability microkernel: providing application-aware reliability in the os. Reliab IEEE Trans 56(4):597–614
62. Yang R, Qu D, Gao Y, Qian Y, Tang Y (2019) nLSALog: an anomaly detection framework for log sequence in security management. IEEE Access 7:181152–181164. https://doi.org/10.1109/ACCESS.2019.2953981
63. Yen TF, Oprea A, Onarlioglu K, Leetham T, Robertson W, Juels A, Kirda E (2013) Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th annual computer security applications conference, pp 199–208. ACSAC '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2523649.2523670, http://doi.acm.org/10.1145/2523649.2523670
64. Yuan D, Zheng J, Park S, Zhou Y, Savage S (2011) Improving software diagnosability via log enhancement. In: Proceeding international conference on architectural support for programming languages and operating systems (ASPLOS 2011), pp 3–14
65. Yuan Y, Srikant Adhatarao S, Lin M, Yuan Y, Liu Z, Fu X (2020) ADA: adaptive deep log anomaly detector. In: IEEE INFOCOM 2020—IEEE conference on computer communications, pp 2449–2458

Critical Infrastructure Protection: Where Convergence of Logical and Physical Security Technologies is a Must

Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano

Abstract Security monitoring is a number one priority, especially in Critical Infrastructure Protection (CIP), where cyber attacks may have dramatic safety impacts, in that they may well result in major damage to assets and/or harm to people. Since effective protection requires that the right actions be taken at the right time, the results of the monitoring process must: (i) be made available in a timely fashion (i.e. in near real-time), and (ii) include detailed diagnostic information (i.e. clearly identify the nature of the problem and the extent of the damage). In this chapter, we: (1) propose an approach for dependable (i.e. accurate, timely, and trustworthy) security monitoring, based on correlation of logical and physical events, (2) implement the approach in a distributed architecture integrating cutting-edge commercial off-the-shelf (COTS) technologies, and (3) validate the approach with respect to a handful of case studies, characterized by challenging—and quite diverse—requirements.

1 Introduction, Problem Statement, and Contributions

A plethora of technologies exists, each one representing an individual building block of a potentially dependable security monitoring facility. Regrettably, they still very much lack integration. While recently some achievements have been made (e.g.: Security Event Monitoring (SEM) and Security Information Monitoring (SIM) have merged into Security Information and Event Monitoring (SIEM), Logical Access Control Systems (LACS) and Physical Access Control Systems (PACS) have merged

L. Coppolino · S. D'Antonio · G. Mazzeo (✉) · L. Romano
Department of Engineering, University of Naples 'Parthenope', Naples, Italy
e-mail: giovanni.mazzeo@uniparthenope.it

L. Coppolino
e-mail: luigi.coppolino@uniparthenope.it

S. D'Antonio
e-mail: salvatore.dantonio@uniparthenope.it

L. Romano
e-mail: luigi.romano@uniparthenope.it

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_13

into Identity Management (IM)), which has resulted in a leap forward of Security Operations Center (SOC) technology, much is yet to be done.

We claim that a significant advancement in the convergence of technologies for logical and physical security is needed. By convergence, we mean: effective cooperation—i.e. coordinated and results-oriented capability of cooperative work—among previously disjointed functions.

The main messages of the chapter can be summarized as follows:

- Security monitoring facilities must be implemented as dependable (i.e. accurate, timely, and trustworthy) functions. In particular, since there will always be faults and intrusions, security monitoring facilities must be designed and implemented as fault- and intrusion-tolerant systems themselves.
- The most important features of a security monitoring facility are Detection and Diagnosis, since the ability of detecting faults, attacks, and intrusions along with the availability of detailed diagnostic information on the nature of the problem as well as on the extent of the damage is the precondition for triggering appropriate reactions and for taking effective remediation actions.
- Enhanced situation awareness is needed for dependable detection and diagnosis of faults, attacks, and intrusions. Situation awareness can only be achieved via effective correlation of security-relevant events, which must be collected both in the logical and in the physical domain.

This work makes the following important contributions:

1. It proposes an approach and a conceptual architecture for improving the convergence of logical and physical security technologies. The approach relies on collection, processing, and correlation of a variety of events which are generated at multiple architectural levels in the logical and in the physical domain (including: sensors, network, Operating System and/or Virtual Machine, Data Base, Application, Business Process, and more).
2. It implements the approach and the conceptual architecture using and/or building on currently available technologies. When discussing the technical solutions, it provides a right to the point review of the current State Of The Art (SOTA) of individual technologies. Importantly, the review features a gap analysis, i.e. it points out the major limitations of such technologies and identifies the main avenues towards convergence-oriented improvement.
3. It demonstrates how the approach can be effectively used to monitor the security of real world case studies, characterized by challenging requirements. Notably, the case studies are very diverse, and taken from different application domains.

The rest of the chapter is organized as follows. In Sect. 2, we present the approach we propose for improving the convergence of logical and physical security technologies, along with a conceptual architecture suitable for implementing the approach in a fault- and intrusion-tolerant distributed system. In Sect. 3, we discuss a number of technologies for logical and physical security, focusing on their limitations (in terms of lack of integration and/or unleashed potentialities for security monitoring

purposes). In Sect. 4, we show how the proposed approach and accompanying architecture can be effectively used in real world setups. Finally, we acknowledge the two enabling factors of the research described in this chapter, namely: (1) Prof. Ravishankar K. Iyer (Ravi) for the inspiration and the scientific/technical mindset that he transferred to Luigi Romano (directly) and to the other authors (indirectly), and (2) the European Commission, for the massive funding that our research group has received throughout the years.

2 Proposed Approach and Conceptual Architecture

Figure 1 shows the conceptual architecture of the integrated security monitoring system, which builds on top of the SANS SIEM reference architecture [1]. The proposed solution collects detection events generated at multiple architectural levels, and in particular: (1) devices monitoring the *physical* world, such as: cameras, door opening sensors, smoke sensors; (2) tools monitoring the *logical* world of the hosting IT infrastructure, such as: network/host IDS, firewalls, applications' log utilities, kernel probes; (3) systems for Business Process Monitoring (BPM) and Business Activity Monitoring (BAM), capable of identifying anomalies in the process flow; and (4) threat intelligence platforms, gathering information about observable activities for threat analysis purposes. It is worth emphasizing that, in the case that the monitored system is deployed on top of a cloud platform, besides collecting events at the Operating System level, information must also be gathered at the Virtual Machine (VM) level. Event collection is implemented by means of multiple security probes,

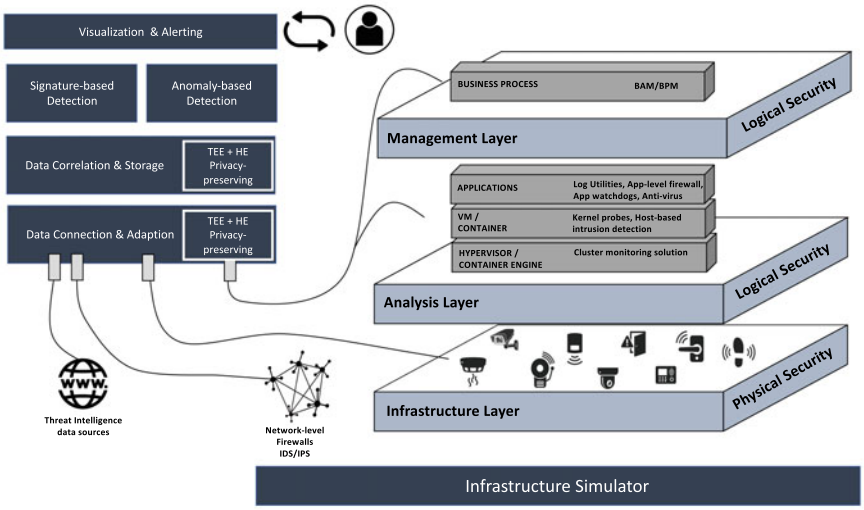


Fig. 1 Architecture of the proposed solution

which are deployed as a distributed architecture. Besides collecting data from the infrastructure, our security monitoring system also acquires live threat intelligence information from the Internet, to improve the awareness of new—and on-going—attacks occurring in similar infrastructures. This activity is particularly important in infrastructures which integrate Information Technology (IT) and Operational Technology (OT) environments, since it is one of the key enabling factors of prevention measures.

Overall, the proposed security monitoring solution consists of five main functional blocks (coloured in black):

- *Data Collection and Adaption*—Large volumes of heterogeneous data collected from the infrastructure are received from a multitude of feeds. This block may perform format adaptions, if these are needed to convert data to common formats. Our solution includes data feeds which are able to preserve the confidentiality of the collected data. We refer in particular to business process information extracted by means of probes based on BPM/BAM tools and/or on parsing of application logs produced by HIDS solutions, which can potentially contain sensitive data. This feature is extremely valuable when the monitoring system is deployed on top of a cloud platform, where protection is needed from particularly challenging types of attacks—notably: those launched by privileged users (e.g.: the cloud provider or the system administrator) and/or software (e.g.: the Hypervisor or the Operating System)—which can also target data *in-use* (e.g.: in the processor or in the main memory), as opposed to data in-transfer and data at-rest. It is worth emphasizing that—to a large extent—protection of data in use is still an open issue. To this end, we propose the use of Trusted Execution Environment (TEE) and/or Homomorphic Encryption technologies, which can be used in combination, to achieve a high level of security at an acceptable performance penalty. More precisely: if the amount of sensitive data to be protected during the processing phase is small (meaning that it fits in the protected memory areas of the CPU), then TEE technology suffices; for larger amounts of data, we propose combined use of HE and TEE technologies (more details on the approach we propose can be found in our VIrtual Secure Enclave (VISE) paper [2].
- *Data Correlation and Storage*—This layer features evidence-based correlation techniques, which correlate multiple events to provide a situational picture of the infrastructure. It is actually here that the physical and logical worlds are combined together, to provide better insight into the status of the system. The Event Correlation stage allows to escalate from fault/intrusion symptoms to the adjudged cause of the fault/intrusion, as well as to estimate the damage to individual system components. The *Aggregation* and *Correlation* of incoming entries are carried out through complex event detection operators that—based on rules specified by the operator—can spot situations which can be observed only if the global context is clearly understood, and—if needed—they can trigger alarms. As already mentioned, confidentiality of the processing of sensitive data is guaranteed. To achieve this goal, we provide: (1) operators running in TEE, and (2) operators capable of processing HE data.

- *Signature-based and Anomaly-based Analytics*—A multitude of operators is used at this level to execute analytics either on *plaintext* or on *encrypted* data. The collected data is analyzed by means of both *signature-based* and *anomaly-based* threat detection mechanisms. It should be noted that—in order to ensure the correctness of the results—if even just one data set is homomorphically encrypted, then all the others necessary for executing the specific analytics must be ciphered with the same HE scheme. Signature-based components compare log entries with a pre-existing database of known attack patterns. In this regard, HE analytics require the availability of HE signatures to detect various types of attacks. As an example, we implemented a *Check Special Char* analytics, which acts as a code injection detector by checking for the existence of special sequence of characters in audit trails entries such as combinations of dots, semicolons, commas, brackets or any other non-alphanumeric ASCII character, which is a symptom of a potential attack.

  Moreover, we also equipped the monitoring system with an analytics for privacy-preserving *anomaly-based* detection. As opposed to *signature-based*, the *anomaly-based* detection builds a normal behavior of the system through a learning phase, and during its execution it monitors if a large deviation from the model occurs, which could be considered as an anomaly. To enable this type of detection, we implemented a supervised *Machine Learning* (ML) algorithm. This is defined as a procedure for deriving models from labeled training data, which represents normal or anomalous state.
- *Visualization and Alerting*—At this layer, notifications are provided to the personnel/machinery in charge of operating the monitored system/application if faults and/or attacks must be handled/countered. The system produces technical—as well as actionable—information that is made available to the personnel/machinery in charge. This enables performing actions/procedures aiming at countering and/or mitigating the effects of faults/attacks that have been detected, as well as prosecution in court.
- *Infrastructure Simulator*—The protection of critical infrastructures requires the security monitoring system to be properly tested in order to verify that it reacts as expected to attack situations. It is important to check that rules configured in the system produce correct alerts and in a timely fashion. However, in many cases, the testing is not doable on the real industrial process as its compromise could have devastating impacts, in terms of humans' safety and damage to costly assets. For this reason, in our solution we propose this additional component that can be used—together with infrastructure administrators—to evaluate the effects that actions would have on the critical infrastructure. At a high-level, our simulator is organized in four main layers. At the top, there is a GUI, which is used to interact with the simulator. The second and third layers consists in a set of solutions for the simulation of Information Technologies (IT), and Operational Technologies (OT), respectively. Finally, the last layer supports the simulation of the industrial process of the particular critical infrastructure. The IT layer is needed to simulate the entire spectrum of technologies for information processing, including software, hardware, communication technologies and related services. The simulated OT

infrastructure includes the typical elements of a SCADA system, such as Remote Terminal Units (RTUs), Master Terminal Units (MTUs), and the Human Machine Interfaces (HMI) part. Finally, the simulated physical process is the layer that feeds data as close to reality as possible, whose dynamic model definition must be performed offline [3]. Our solution leverages the widely-accepted OPC standard to link the simulated process and the simulated IT/OT devices. In this way, we can interact with virtually any simulator (e.g., OpenModelica and Simulink) that transmits data feeds using the OPC protocol. We can also support hybrid solutions, where real and simulated elements co-exist.

3 Technology Pillars

The data collection component of the integrated monitoring system can be easily connected to a plethora of data feeds. Our solution uses *Logstash* for data collection. Logstash has already several adapters for this purpose and adding new feeds is relatively straightforward. As discussed in 2, data collection must operate at multiple layers of the monitored infrastructure. At the infrastructure layer, challenges come from the monitoring of hardware "health". Tools exist to detect equipment malfunctioning and foresee failures. As an example, S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) is a monitoring technology for computer hard disk drives that detects and reports their reliability state, in the hope of anticipating failures. Unfortunately, at present, S.M.A.R.T. is implemented individually by manufacturers, and while some aspects are standardized for compatibility, others are not. Random Access Memory (RAM) monitoring is a feature to verify that the computer can reliably store and retrieve data from memory. A test of memory is generally performed at the time of powering up, but tools exist to test the memory equipment at run-time. A computer that fails these tests—perhaps because of old hardware, damaged hardware, or poorly configured hardware—will be less stable and crash more often. Even worse, it will become even less stable over time as corrupted data is written to your hard disk. Other two key building blocks of a security monitoring facility are Business Process Management (BPM) [4] and Business Activity Monitoring (BAM) [5]. BPM has been referred to as a "holistic management" approach to aligning an organization's business processes with the wants and needs of clients. It is worth emphasizing that these processes are critical to the organization, as they (i) can generate revenue, and (ii) often represent a significant proportion of costs. BAM is software that aids in monitoring of business activities, as those activities are implemented in computer systems. It provides near real-time monitoring of business activities, measurement of Key Performance Indicators (KPIs), their presentation in dashboards, and automatic and proactive notification in case of deviations. BPM and BAM are used (almost) exclusively for monitoring the QoS at the application level. Since many emerging attacks, which evade current IDS/IPS technology, have clear symptoms in terms of QoS degradation, BPM and BAM have a great potential in terms of performance improvement of the detection process. By understanding the

Business Process Logic, it is possible to detect new categories of faults/attacks, e.g.: faults related to orchestration flaws and attacks related to the exploitation of misuse cases.

Since the security monitoring solution can be itself the target of attacks, to achieve a high level of fault- and intrusion-tolerance it is mandatory that technologies that guarantee a good level of robustness, scalability and elasticity be used throughout the lifecycle of data (i.e. when data is collected, transferred, processed, and stored). As for the processing, the correlation technology must guarantee flexibility in terms of data fusion logic and provide support to both signature and anomaly based analysis.

Based on such considerations, our choice was *ElasticSearch* for the storage and *Apache Kafka* for the propagation. As for the latter, we prefer an asynchronous communication bus to accommodate burst of data and tolerate possible temporary downtown of the network infrastructure. As for the communication format, an extension of the popular IDMEF format was adopted to keep into consideration the information related to physical positioning and additional spatial information. Regarding the processing, we use the *Apache Flink* tool, since this is a highly configurable and extensible solution. Moreover, Flink offers effective support in terms of Machine Learning technologies (e.g. Machine Learning for Flink,[1] Apache Mahout[2] or Alink from Alibaba[3]) for the implementation of anomaly detection logics.

We used two TEE technologies for the protection of data in hosting servers and devices. In the former case, we leveraged the Intel *Software Guard eXtension (SGX)* [6, 7], which allows to create a Trusted Execution Environment (TEE) based on a mechanism of "reverse sandbox", i.e., the world outside of the sensitive application (OS, Hypervisor, BIOS/Firmware) is considered as untrusted and, thus, a potential source of attacks. In SGX, the sensitive processes' address space is protected within the CPU perimeter. SGX also provides a mechanism for *Remote Attestation (RA)* [8], which enables the server's owner—e.g., the service provider–to prove via a trusted remote third-party that (i) the enclave has a valid measurement hash, (ii) it is running in a secure environment, and (iii) it has not been tampered with.

Devices instead have been protected via *ARM TrustZone*. The main idea in this case is that the processor is physically split in two execution environments, referenced as normal world (REE) and secure world (TEE). Both worlds have their own user space and kernel space, together with cache, memory and other resources. The normal world cannot access the secure world's resources while the latter can access all the resources. The normal world is used to run a basic OS, which provides a Rich Execution Environment (REE). Meanwhile, the secure world always uses a secure small kernel (TEE-kernel).ARM also introduces a *monitor mode* to handle switches between the REE and TEE. Monitor mode saves and restores the context of each environment during the switches. To enter Monitor mode, the process in each world invokes a special instruction called Secure Monitor Call (smc) with kernel privileges. Regarding the HE feature, our solution was tested two different crypto-schemes,

[1] https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/libs/ml/index.html.

[2] https://mahout.apache.org/.

[3] https://github.com/alibaba/Alink/blob/master/README.en-US.md.

i.e., DGHV [9] and TFHE [10], which ensure fully-homomorphic properties with different performance. During our validations, we discovered that the algorithm that better fits with ICS requirements is the TFHE.

4 Case Studies

In this section, we show how the proposed monitoring solution can tackle relevant issues in critical case studies. Selected case studies come from the experience maturated by the authors in the context of projects co-funded by the European Commission, and include: Dam and Water Supply Network management, Public Administration and eHealth services, Sensitive Industrial Plant for Chemical Storage. In every selected scenario, different features of the proposed security monitoring solution were leveraged, as summarized in Table 1, to afford specific challenges of the case study. We demonstrate that the proposed solution (and the accompanying architecture) can be effectively adopted to improve the security of real world setups, with challenging—and diverse—requirements.

4.1 Dam and Water Supply Network

Dams and water supply networks represent fundamental assets for the economy and the safety of a country. Recent reports about attempts of cyberattacks against these critical infrastructures have brought national and international authorities as well as equipment manufactures to propose new approaches to reduce the risk of

Table 1 Summary of features validated in case studies

Feature	Dam and water supply network	Public administration and eHealth	Sensitive industrial plant for chemical storage
Cyber-physical correlation	G	G	G
Signature-based detection		G	
Anomaly-based detection	G		G
Secure processing via TEE	G		
Secure processing via HE		G	
Simulation of infrastructures			G

cyberthreats. MASSIF and SAWSOC projects dealt with the protection of dam and water supply networks from attacks.

Dam process monitoring and control relies on the use of Supervisory Control and Data Acquisition (SCADA) systems that comprise physical sensors, actuators, Remote Terminal Units (RTUs), Human Machine Interfaces, administration consoles, master stations, historical databases, etc. Moreover, monitoring systems largely adopt Wireless Sensor Network (WSN) technology, which offers cost-effective solutions for harsh environments. Most of current dam control systems integrate Commercial Off-The-Shelf (COTS) components and they are mostly interconnected by using standard protocols such as the TCP/IP suite. In this scenario control system components communicate and interoperate with IT systems, such as data loggers and visualization stations, via the Internet. Although the operations and functionality of the overall infrastructure benefit from an interconnected and integrated control system, the connection to the Internet exposes the control system itself to the risks of cyberattacks, which can exploit system vulnerabilities to compromise components with unpredictable consequences for dam safety. Current mechanisms and policies devoted to dam infrastructure protection are mainly related to cybersecurity of the monitoring and control system and physical security of dam facilities. However, these solutions usually lack the capability of interfacing with heterogeneous systems and technologies as well as they do not support the collection and analysis of monitoring data from different sources and probes.

Using SIEMs to jointly manage all different security-relevant events and information generated by the dam monitoring probes would be a very powerful mean to increase the overall protection of such critical infrastructures. Regrettably currently available SIEM systems deal with the management of logical security aspects and are specifically designed for this security context. This might make complex the development of applications targeting security of critical infrastructures in a wider sense. For instance, correlating network and host events that could be a symptom of an ongoing cyberattack with suspicious activities detected by the dam surveillance system may greatly improve the capability of tackling coordinated and sophisticated threats, but SIEMs are not designed to deal with this kind of scenarios and therefore the correlation of security events coming from different application domains may be troublesome. In particular, current solutions usually are not able to merge physical and logical events into a single stream to process, which definitively improves the effectiveness of the detection process. The proposed approach exploits data fusion to achieve convergence between physical and logical security and applies it to improve the protection of a dam infrastructure.

Data Fusion is the process of combining information from a number of different sources to provide a robust and complete description of an environment or process of interest. Data Fusion process is applied where a large amount of data is combined and fused to obtain information of appropriate quality and integrity on which decisions can be made. In any data fusion problem, there is an environment, a process, or a parameter whose true value, situation or state is unknown. The sources provide imperfect and incomplete knowledge, that is processed and then transformed in decisions, thus effectively supporting human or automated decision making. Data

fusion is the process of combining data to refine estimates and predictions of the state that is observed.

In the dam case study correlation is driven by rules that are created according to the data fusion paradigm. A main challenge of physical-logical correlation is translating logical and physical events in a domain where they can be actually compared and correlated. The selected solution was based on the adoption of the Dempster-Shafer theory. Events captured from physical security world, like a suspicious access to the control room where the master station of the SCADA system is located, are correlated with the measurement data provided by the sensors that are distributed throughout the dam infrastructure. This is done in order to detect anomalous values of the metrics that could be the effect of the false data injection performed by a malware installed in the master station. The correlation of the data from the logical and physical domain allows to detect attacks that otherwise would be undetectable by a security system looking at only one of the two domains.

4.2 Public Administration and eHealth

Enforcing security in PAs and e-health entities is especially challenging due to a number of factors which, from a security point of view, bring together the two application fields. First, involved entities are very heterogeneous in size, budget, and services. As a consequence, requirements and threats are themselves diverse. Second, in both sectors, the core business is far away from IT but strongly depends on it. The latter consideration implies that the final quality of offered services is extremely sensitive to key competences that are often lacking within the entity. Third, in both sectors extremely sensitive data are managed and exchanged.

A first consequence of such characteristics is that in these sectors, security monitoring cannot be treated as a plug and play solution provided throughout a standard COTS tool, but it needs to be included as part of a complex security management process. The *COMPACT* project [11] specifically targets security of Local Public Administrations (LPAs). It assumes the security monitoring as one of the many building blocks of a security management and improvement process. Precisely it proposes a specialization of the well-known and consolidated Plan-Do-Check-Act (PDCA) cycle (Fig. 2), which enables LPAs to innovate their cybersecurity improvement process in compliance to the EN ISO/IEC 27001 and BS ISO/IEC 27005 standards. The *planning* stage is aimed at assessing the initial context and setting objectives. At this stage tools and techniques for security monitoring are of foremost importance to gather needed information related to processes and systems and to feed the risk assessment process. After that the *do* stage results in the installation of every planned countermeasure and the consolidation of security monitoring itself, in the *check* stage real time security monitoring comes back to continuous monitoring of security related events. While doing so, it also provides info related to the correct functioning of security tools, and spots out the need for tuning already installed mitigation solutions (*act* stage).

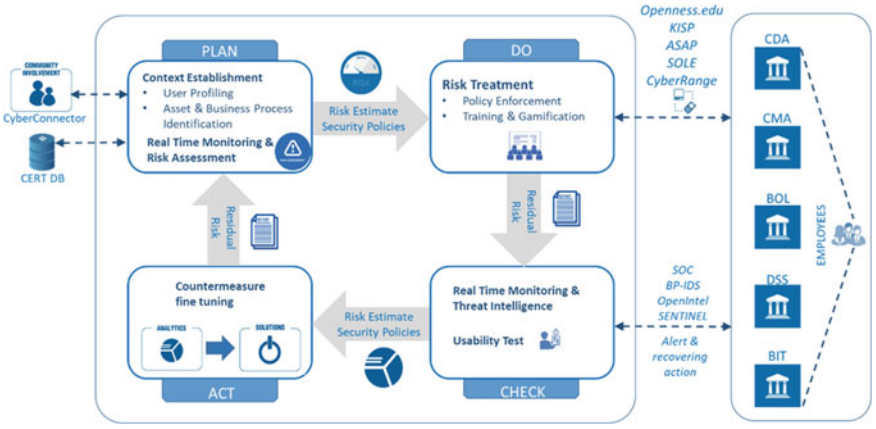


Fig. 2 The COMPACT PDCA cycle for security management and improvement in LPAs

The COMPACT project also highlighted how, in these sectors, the effectiveness of personnel training can be dramatically improved by exploiting new training experiences such as gamification. Since training results could provide a valuable data feed, they need to be integrated in the security monitoring process throughout a versatile data collection and adaptation solutions that can interact with training platform in spite of their actual implementation (software, card games, etc.).

Since entities, both in the public sector and in the e-health one, lack internal competences and proper investments in IT related stuff, it is a common choice to externalize some activities such as security monitoring. Unfortunately, while subscribing managed security services the company also accept to feed external providers with business related data including sensitive ones. Though well known methods are established for protecting the transferring and the storage of the information, data stays exposed while being processed within the service provider's perimeter: not only the service provider, but any attacker gaining admin-like privileges, can potentially access the plain information during its elaboration. With the advent of cloud computing, a number of techniques for protecting in-use data are being proposed, including Trusted Execution Environments, Homomorphic Encryption (HE), and Functional Encryption. Every technique has its own pros and cons. In the *KONFIDO* project [12], aimed at ensuring personal data security in cross border treatments of European patients, such a challenge also arise when the KONFIDO federated SIEM solution needs to collect and process data belonging to multiple countries. The solution implemented in the project (Fig. 3) exploits the TEE privacy preserving module within the general architecture presented in Sect. 2. Specifically the KONFIDO solution is built on top of the *OpenNCP* [13]. European eHealth management system. OpenNCP provides several interoperable services for cross-border exchange of PatientSummaries (PSs) and ePrescriptions. The KONFIDO SIEM solution was challenged by detection of code injection attacks operated on the value fields of a Patient Summary: spotting out the attack needs the PS value fields inspection

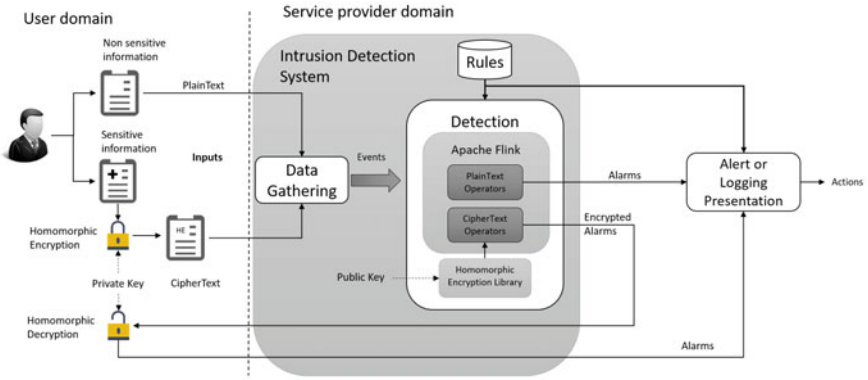


Fig. 3 The monitoring solution for the eHealth case study

but, on the other side, such fields could include patient related sensitive information. The adopted solution exchanges PSs throughout TEEs, once the PS reaches the SIEM context it leaves the TEE but only after its sensitive fields have been homomorphically encrypted. The list of sensitive fields comes with a configuration file exchanged together with the PS. The SIEM thus can inspect encrypted data by using some dedicated signatures without actually accessing the plaintext. Further details related to privacy preserving security monitoring, including performance analysis, can be found in [14].

4.3 Sensitive Industrial Plant for Chemical Storage

As already discussed in Sect. 2, the protection of critical infrastructures requires the security monitoring system to be properly tested in order to verify that it reacts as expected to attack situations. It is important to check that rules configured in the system produce correct alerts and in a timely fashion. In this section, we demonstrate *how* the feature provided in the portfolio of our security monitoring solution can support such testing operations without the risk of compromising safety–critical processes. This validation activity was made in the context of the EC-funded *InfraStress* project on a very suitable ICS case study: a chemical storage infrastructure managed by the *Attilio Carmagnani "AC" S.p.A.* company.[4]

The *Attilio Carmagnani "AC" S.p.A.* terminal covers an area of about 30,000 square meters and includes a set of 31 semi-buried or underground tanks with a total capacity of 26,840 m^3, distributed as follows: 5 tanks of 3.000 m^3, 4 tanks of 1.000 m^3, 6 tanks of 700 m^3, 2 tanks of 400 m^3, 6 tanks of 300 m^3, 8 tanks of 130 m^3. The tanks normally contain: Aromatic Hydrocarbons, Aliphatic Solvents, Acetates, Alcohols and Ethylene Glycols. The terminal is connected to Genoa Oil Terminal

[4] https://www.carmagnani.com/.

via 3 stainless-steel pipelines and is equipped with trucks and rail loading platforms connected to the main road, highway and railway, allowing easy transportation of goods towards national and international main commercial hubs. The plant is in a densely populated area: for this reason, the safety and security for this infrastructure is of paramount importance. *Attilio Carmagnani "AC" S.p.A.* has one specific process that is safety critical. This consists in a loading/unloading procedure of dangerous chemicals (highly flammable and hazardous to the aquatic environment), which could cause catastrophic consequences if attacked. The critical process involves stainless-steel pipelines connected to the facility of Genoa Oil Terminal, which are used to transfer the chemicals from vessels to infrastructure's tanks. One particular flow is dedicated to highly flammable products. This involves a special tank which is equipped with sensors and radars to control its levels of chemicals inside. If thresholds are reached, a three-way valve is switched by the ICS and the chemicals are forwarded to other reserve tanks.

The attack tree reported in Fig. 4 shows the scenarios that were tested with the security monitoring solution on the chemical storage infrastructure. In this diagram,

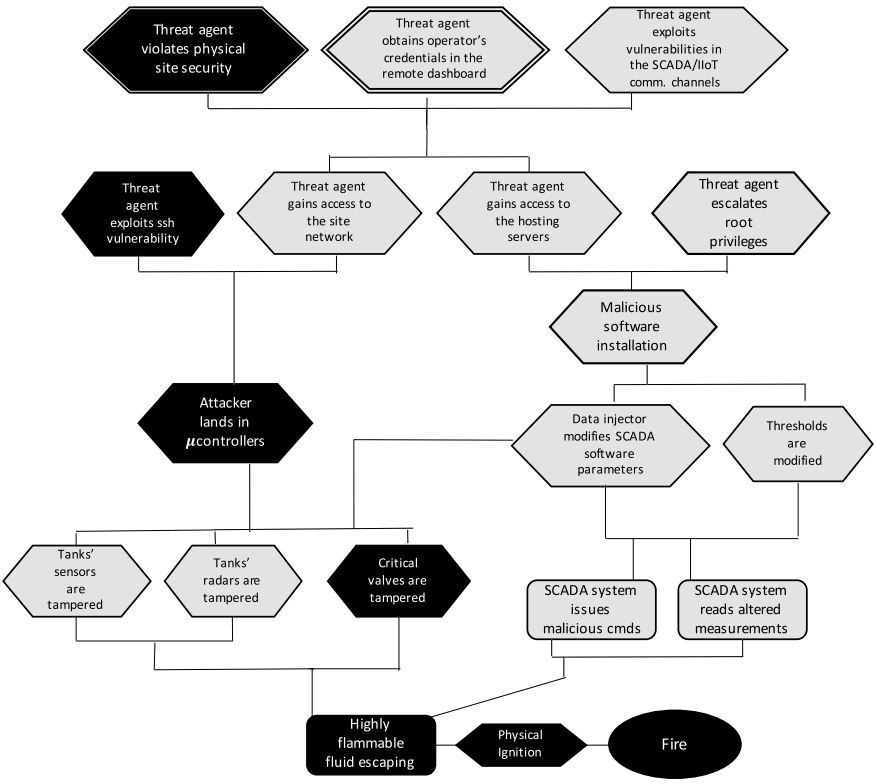


Fig. 4 The attack tree covered by the security monitoring solution for the chemical storage case study

it can be noticed that attackers can violate the security of the infrastructure leveraging physical or remote vectors. The threat agent violates the physical site security accessing the site from the railway gate or from the fence wall. If the site access was successful, then the Control Room could be the next target. Two doors protect the Control Building. In this sense, physical detectors implemented in the context of the InfraStress project raise a warning event that is sent to our security monitoring system. Once inside the infrastructure, the attacker could either log into the SCADA server and inject malicious monitoring/control parameters, or land in microcontrollers deployed over the infrastructure to, e.g., tamper sensors/actuators measurements, which could lead to highly flammable fluid escaping. Cyber detectors of our security monitoring system gather information from the units running in the ICS and send warning/alert event to the central unit which, in turn, correlate the events received from the physical and cyber world to identify a more complex pattern of attack occurring on the chemical storage infrastructure.

The simulation tool was properly tuned to reproduce the deployment discussed above. As a first step, we implemented the model of the identified critical process (see Fig. 5). To this end, we leveraged the *OpenModelica* tool. Afterwards, we configured cyber-physical detectors on top of the simulated process, which collect events from the modelled process. They use the OPC standard to communicate the data feeds. In this way, the security monitoring system can receive events from both real and simulated worlds. Finally, we configured the alarming rules in our monitoring system and tested their behavior during the attack. We considered the path highlighted in Fig. 4. One physical detector processing (a pre-recorded) video from surveillance cameras raised a warning event. A cyber detector installed in the simulated environment launched another warning after our simulated attack was made. We exploited a vulnerability in Linux OS to enter the infrastructure network. From here inside, we reached a microcontroller unit, i.e., a simulated ARM device that was exploited to tamper the valve1 shown in Fig. 5. Our security monitoring system correlated a series of detection events and reported the situation to the operator that was not aware of the on-going simulation.

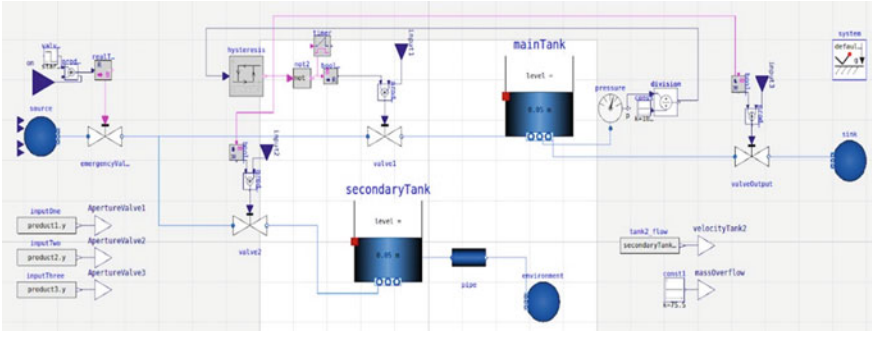


Fig. 5 The simulated *Attilio Carmagnani "AC" S.p.A.* critical process

5 Conclusion

Physical and logical security are still treated as two separate and distinct domains, even though recent attacks are more and more performed through coordinated, multi-step, and hybrid actions targeting both physical and cyber assets. A unified approach to security monitoring is definitely beneficial to the protection of an organization since the capability of collecting and processing events from both domains makes the organization more resilient and better prepared to prevent, detect, and mitigate novel and advanced attacks. In this chapter we presented an approach to security monitoring that relies on the convergence of cyber and physical security through the correlation of events coming from the two domains. This approach has been validated in four application scenarios, namely dam and water supply network, sensitive industrial plant for chemical storage, public administration, and eHealth, where challenging case studies in terms of complexity, diversity, and relevance of the critical infrastructure to be protected have been developed.

Acknowledgements First of all, authors wish to acknowledge Prof. Ravishankar K. Iyer (Ravi) for the inspiration and the scientific/technical mindset that he has transferred to Luigi Romano (directly) and to the other authors (indirectly). What Luigi learned from Ravi has shaped the research that the Fault and Intrusion Tolerant NEtworked SystemS (FITNESS) group (http://www.fitnesslab.eu/) has done in the past years and will influence the research the group will do in the years to come.

As importantly, authors acknowledge the massive funding they have received from the European Commission throughout the years. Specifically: the research leading to these results has received funding from the European Commission within the context of:

• the Seventh Framework Programme (FP7/2007-2013)—Under Grant Agreement No. 313034 (Situation AWare Security Operation Center, SAWSOC Project); Grant Agreement No. 257644 (MAnagement of Security information and events in Service Infrastructures, MASSIF Project).

• the Horizon 2020 Programme (H2020/2014-2020)—Under Grant Agreement No. 833088 (InfraStress); Grant Agreement No. 833088 (InfraStress project); Grant Agreement No. 6450311 (Secure Enclaves for REactive Cloud Applications, SERECA project); Grant Agreement No. 727528 (KONFIDO), Grant Agreement No. 740712 (COmpetitive Methods to protect local Public Administration from Cyber security Threats, COMPACT).

References

1. SANS (2018) Sans reference architecture. https://www.sans.org/reading-room/whitepapers/logging/paper/38720. Accessed 24 Oct 2019
2. Coppolino L, D'Antonio S, Formicola V, Mazzeo G, Romano L (2020) Vise: combining intel SGX and homomorphic encryption for cloud industrial control systems. IEEE Trans Comput 1–1
3. Giuliano V, Formicola V (2019) Icsrange: a simulation-based cyber range platform for industrial control systems
4. Hariyanti E, Djunaidy A, Siahaan DO (2018) A conceptual model for information security risk considering business process perspective. In: 2018 4th International conference on science and technology (ICST), pp 1–6

5. Naseer H, Maynard SB, Desouza KC (2021) Demystifying analytical information processing capability: the case of cybersecurity incident response. Decis Support Syst 143:113476. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923620302311
6. McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR (2013) Innovative instructions and software model for isolated execution. In: Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy, ser. HASP '13. ACM, New York, NY, USA, pp 10:1–10:1
7. Costan V, Devadas S (2016) Intel SGX explained. Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org/2016/086
8. Anati I, Gueron S, Johnson SP, Scarlata VR (2013) Innovative technology for CPU based attestation and sealing
9. van Dijk M, Gentry C, Halevi S, Vaikuntanathan V (2010) Fully homomorphic encryption over the integers. In: Gilbert H (ed) Advances in cryptology—EUROCRYPT 2010, Berlin, Heidelberg: Springer Berlin Heidelberg
10. Chillotti I, Gama N, Georgieva M, Izabachène M (2016) Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870. http://eprint.iacr.org/2016/870
11. Coppolino L, D'Antonio S, Mazzeo G, Romano L, Sgaglione L (2018) How to protect public administration from cybersecurity threats: the compact project. In: 2018 32nd International conference on advanced information networking and applications workshops (WAINA), pp 573–578
12. Staffa M, Coppolino L, Sgaglione L, Gelenbe E, Komnios I, Grivas E, Stan O, Castaldo L (2018) Konfido: An openNCP-based secure ehealth data exchange system. In: Gelenbe E, Campegiani P, Czachórski T, Katsikas SK, Komnios I, Romano L, Tzovaras D (eds) Security in computer and information sciences. Springer International Publishing, Cham, pp 11–27
13. Fonseca M, Karkaletsis K, Cruz IA, Berler A, Oliveira IC (2015) "OpenNCP: a novel framework to foster cross-border e-health services. In: Digital healthcare empowering Europeans—Proceedings of MIE2015, Madrid Spain, 27–29 May, 2015, pp 617–621. [Online]. Available: https://doi.org/10.3233/978-1-61499-512-8-617
14. Sgaglione L, Coppolino L, D'Antonio S, Mazzeo G, Romano L, Cotroneo D, Scognamiglio A (2019) Privacy preserving intrusion detection via homomorphic encryption. In: 2019 IEEE 28th international conference on enabling technologies: infrastructure for collaborative enterprises (WETICE). IEEE Computer Society, Los Alamitos, CA, USA, Jun 2019, pp 321–326. [Online]. Available: https://doi.ieeecomputersociety.org/. https://doi.org/10.1109/WETICE.2019.00073

Health Care and CPS

Introduction: Cyber Physical Systems and Healthcare Analytics

Arjun P. Athreya

Computing systems and computer algorithms making use of measurements derived from cyber physical systems (e.g., smart grids) or biological systems (e.g., humans) have led to several breakthroughs in the recent years. These breakthroughs grounded in novel foundational analytical formations comprise modern-day electric grids, self-driving cars and precision medicine approaches. These innovations are at the heart of the way we live, work, and evolve into the technology-rich future guided by autonomous decision-making robots. Hence, these technologies share the same if not increased needs of reliability and dependability in performance and trustworthiness of well-studied mission-critical systems such as aircrafts or supercomputers. For example, can a smart grid determine the legitimacy of a sudden and unexpected surge in load as being genuine versus an active cyber attack? Can algorithms using clinical, genomics and imaging features reliably predict which medication and what dose will a patient achieve symptomatic remission from chronic conditions?

This section demonstrates the power of novel data and analytics that embody paradigms of reliable and trustworthy technologies in cyber-physical systems and healthcare. Wu et al. propose real-time anomaly detection (ReTAD) algorithm for smart grids to detect anomalous events in real-time with improved performance in detection of multiple lines with outages. Dr. Lin discusses the approach to developing general-purpose security architecture by knowing domain-specific knowledge of the target cyber-physical system such as smart-grids. Transitioning into healthcare analytics, Drs. Wang and Weinshilboum provide a demonstration of the ability of engineering and healthcare collaborations (between Univ. of Illinois at Urbana-Champaign and Mayo Clinic) in facilitating precision medicine approaches—with a case study of major depressive disorder. Next, with an emphasis on natural language processing in healthcare, Dr. Devarakonda presents novel-data driven approaches for selecting samples to train modern neural networks. Basu et al. show the possibility

A. P. Athreya (✉)
Mayo Clinic, Rochester, MN, USA
e-mail: Athreya.arjun@mayo.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_14

to detect genomic variants using deep learning approaches, with an application to the severe acute respiratory syndrome coronavirus 2 (SARS-CoV2). Finally, Dr. Worrell elucidates the benefits of engineering, neuroscience and neurology synergy in finding technologies for individualized management of epilepsy.

This section summarizes the evolution of the use of conventional reliability and dependability approaches into the development of modern-day analytics for technologies in cyber-physical systems and healthcare under the aegis of collaborations led by Prof. Ravi Iyer.

On Improving the Reliability of Power Grids for Multiple Power Line Outages and Anomaly Detection

Jie Wu, Jinjun Xiong, and Yiyu Shi

Abstract Improving the reliability of smart grids is critical to not only cost-effectiveness of electricity delivery but also repair cost reduction. To efficiently improve the reliability, the real-time anomaly behavior detection and efficient location identification of multiple line outages play a major role in wide area monitoring of smart grids. However, capturing the features of anomalous interruption and then detecting them at real time is difficult for large-scale smart grids, because the measurement data volume and complexity increase drastically with the exponential growth of data from the immense intelligent monitoring devices to be rolled out. This is especially true for multiple line outage detection, as the methods of identifying the locations of multiple line outages face two major challenges: a limited number of Phasor Measurement Units (PMUs) and the high computational complexity. This chapter proposes an efficient real-time anomaly detection (ReTAD) algorithm to address these challenges, inspired in part by the ambiguity group theory. To characterize the performance of line outage identification, this chapter also introduces a statistical model to describe the average identification capability of multiple line outages. Under this model, we develop a global optimal PMUs placement strategy to maximize the average identification capability for a fixed budget of PMUs. Using 14-, 30-, 57-, 118- and 2383-bus systems, our experimental study demonstrates that our proposed ReTAD algorithm successfully detects the anomalous events in real-time and identifies the most likely multiple line outages with a 500× speedup when compared to the method of exhaustive search. For the IEEE 14- and 57-bus sys-

Some parts of this chapter were first appeared in IEEE TRANSACTIONS ON POWER SYSTEMS Journal in 2014, IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2014, and IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS) in 2014, respectively. The publishers were kind enough to allow the authors to reuse some of content in order to produce a more complete narrative of this book chapter.

J. Wu
Kneron, Inc, San Diego, CA, USA

J. Xiong (✉)
University of Buffalo, Buffalo, NY, USA
e-mail: jinjun@buffalo.edu

Y. Shi
University of Notre Dame, Notre Dame, IN, USA

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_15

tems, our experimental study also demonstrates that the proposed techniques can select optimal PMU locations while improving the average identification capability by about 10% compared to random PMUs placement method.

Keywords Multiple power line outages · Location identification · PMU placement · Real-time · Anomaly detection

1 Introduction

Smart grids, managed by significant intelligence, have demonstrated a great potentials to improve the electricity delivery efficiency from suppliers to consumers while still satisfying the demand for electricity. The success of smart grids, however, has been hindered by service interruption caused by abnormal events such as transmission line outages. For example, two thirds of the blackouts experienced in North America were caused by a lack of situational awareness [1]. As a result, power is becoming less reliable and maintenance is becoming more expensive. Over the last decades, as the demand for electricity increases, the transmission system is used under a high stress [2, 3] in a deregulated environment. A small abnormal event that escapes timely detection could culminate in a catastrophic service break-down, resulting in billions of dollars of loss in revenue [4]. Therefore, the reliability issue of transmission systems, especially the issue of multiple simultaneous line outages, has risen significantly in the grid [3]. Anomaly detection at an early stage and identification of the locations of multiple line outages are thus highly desirable.

In the smart grid, the intelligent monitoring devices, such as supervisory control and data acquisition (SCADA) and GPS-synchronized phasor measurement units (PMUs), are being deployed for line outage detection [5–8]. Thanks to the real-time synchronized measurements by phasor measurement units (PMUs) with high precision [9–12], PMU is widely considered as a major sensor for line outage detection [6–8, 13]. Based on the PMUs measurements, prior studies have focused on transmission line outages detection [5–8, 13]. For instance, Tate et al. proposed an algorithm to detect single line outage based on the system topology information [13]. And subsequently, Tate et al. designed an algorithm for double line outage detection [6]. Multiple line outages can be detected by adopting a Markov-dependency graph model based on the assumptions that the bus phasor angles are condition independent and the PMU measurements cover the entire network [7]. Levorato and Mitra [14] presented an algorithm using a wavelet projection method and a sparse approximation technique to detect the stochastic anomalous behavior in smart grid. Zhu and Giannakis [8] proposed a sparse signal reconstruction algorithm to capture topology-bearing information to detect the number of multiple line outages. To reduce the complexity while improving the number of line outage detection performance, Chen et al. [15] presented a detection algorithm for multiple line outages detection based on cross-entropy optimization. These studies have focused on detecting the number of line outages. Although detecting the number of line outages is an

important work for smart grids, location identification of multiple line outages will be more helpful for practitioners to solve the pragmatic problems.

This proposed method can deal with multiple outages at affordable complexity, but requires the fixed number of line outages as an input. This requirement may not suit for the scenario without the priori information on the number of line outages. Thus, how to do the real-time anomaly detection and identify the locations of multiple line outages with limited PMUs, associated with optimally selection PMUs locations within acceptable computational cost to achieve the maximum identification performance for multiple line outages remains an unresolved issue till now.

Therefore, developing the real-time anomaly detection and the location identification algorithms for multiple outage lines are difficult because of the following challenges:

1. A limited number of PMU measurements. The efficacy of location identification of line outages is affected by the information received from PMUs. Placing enough PMUs to cover the entire interconnection of a power grid is however too expensive to warrant this method feasible [16]. Hence, limited PMU measurements threaten the complete and unique identification of multiple line outages in the non-observable systems.
2. High capacity phasor angle data storage. The gigantic measurement data analysis is also the bottleneck for real time anomaly detection. For example, a PMU generates as many as 60 timestamped samples per second. The PMU data accumulation is in the range of several terabits per day based on the North American Electric Reliability Corporation (NERC) report [17]. How to fast analyze and response the anomaly events based on the high capacity PMU data is really challenge.
3. High computational complexity. The impact of multiple line outages need to be modeled, characterized, and recognized. Capturing the characteristics of multiple line outages and then identifying the location of each is difficult for large-scale smart grids. This computational complexity must be carefully considered to extend the feasibility of the location identification algorithm design.
4. The difficulty to mathematically describe the identification performance for multiple line outages. Since multiple line outages can simultaneously occur at different locations of smart grid and different line outages may lead to the same PMUs measurements, capturing the characteristics of multiple line outages and addressing the unique location of line outage are difficult. Most existing studies cannot mathematically model the identification performance of multiple line outages [6–8, 18].

These existing studies have all focused on the off-line detection of transmission line outages for smart grid. Due to the lack of a comprehensive consideration the issue of huge phasor angle data storage and analysis, these existing work cannot be extended to real time anomaly detection and location identification for multiple line outages. To alleviate the gigantic phasor angle data storage and analysis complexity for real time anomaly detection, we propose a novel anomaly detection architecture

and deploy a real time detection algorithm and a location identification methodology for multiple line outages in large scale smart grid.

In contrast to existing engineering and development efforts, the goal of this work is to, for the first time, provide an anomalous behavior modeling and a real time anomaly detection framework to enable rigorous abnormal events analysis and design. The proposed anomaly detection framework uses fast estimation technique of high dimensional covariance matrices with small sample sizes for efficient and accurate characterization of anomalous behaviors in large-scale power systems under gigantic phasor angle data volume. After detecting the anomaly behavior, under linear DC power flow model, we propose a novel location identification scheme (LIS) that can provide the most likely locations of multiple line outages with reasonable computational complexity and limited PMUs measurements. Furthermore, we also propose a mathematical model to describe the average identification capability of multiple line outages. We then explicitly formulate the PMUs placement problem for identifying multiple line outages as an optimization problem to maximize the average identification capability.

The remainder of the chapter is organized into four sections. Section 2 addresses the problem formulation of the real-time anomalous behavior detection, the location identification for multiple line outages, and the PMUs placement and average identification capability with mathematical closed-form. Section 3 discusses a fast and efficient real time anomaly detection (ReTAD) algorithm for anomaly detection at real-time, an efficient LIS algorithm to provide the most likely locations of multiple line outages with limited PMUs, and two optimal PMU placement algorithms to find the optimal PMUs locations under a PMUs budget. Section 4 introduces the experimental results of the ReTAD, LIS algorithms on 14-, 30-, 57-, 300-, and 2383-bus systems. Concluding remarks are given in Sect. 5.

2 Problem Formulation

This section firstly introduce power flow model formulation. Secondly it presents an anomalous behavior modeling and analysis framework to facilitate the proposed real time anomaly detection (ReTAD) algorithm design. Thirdly, it will introduce the location identification scheme (LIS) design. Finally, it will formulate the optimal PMUs placement problem to maximize the average identification capability (AIC).

Power flow models are used to analyze the relationship between injected power and received power. Assuming constant voltage magnitudes per bus and negligible transmission losses [19], the DC power flow model offers a simple linear analysis tool to cope with the nonlinear AC power flow. Although confined only to linear approximate analysis of the actual nonlinear system, the DC power flow model turns out to facilitate a variety of power system monitoring tasks under normal operating conditions, including contingency analysis. The DC power flow model is thus commonly used in the literatures to analyze the line outages [6–8]. We adopt the

same model for our formulation, and we will evaluate the efficacy of our approach through AC power flow models in the *Numerical Simulation* section.

The power transmission network, consisting of N buses and L transmission lines, represents the connection relationship between those buses. For each bus $i \in \{0, 1, \ldots, N-1\}$, the injected power attempts to communicate with the bus $j \in \{0, 1, \ldots, N-1\}$ of received power through the susceptance matrix (described by an N-by-N matrix $\mathbf{B}$). Reference bus denotes as 0.

$$p_i = \sum_{j=0}^{N-1} p_{ij} = \sum_{j=0}^{N-1} b_{ij}(\theta_i - \theta_j), \tag{1}$$

where p_i is the injected power at bus i. p_{ij} represents the power flow from the injected power at bus i to the received power at bus j ($i, j \in \{0, 1, \ldots, N-1\}$) through the susceptance matrix ($\mathbf{B}$). The entry of $\mathbf{B}$ (b_{ij}) represents the inverse of the line inductive reactance x_{ij} between bus i and bus j, given by

$$b_{ij} = \begin{cases} -\frac{1}{x_{ij}} & i \neq j,\ x_{ij} \neq 0 \\ \sum_{j=1}^{N} \frac{1}{x_{ij}} & i = j,\ x_{ij} \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that $b_{i,j} = 0$ means the transmission line between bus i and bus j is broken. θ_i and θ_j represent the phasor angles at the bus i and the bus j, respectively. For conciseness, we rewrite (1) in the matrix form [8] as follows:

$$\mathbf{p}_{0_{[N\times 1]}} = \mathbf{B}_{0_{[N\times N]}} \cdot \mathbf{\Theta}_{0_{[N\times 1]}}, \tag{3}$$

where 0 denotes the pre-outage value. Meanwhile, the post-outage power flow ($\mathbf{p}_1$) is expressed as:

$$\mathbf{p}_{1[N\times 1]} = \mathbf{B}_{1_{[N\times N]}} \cdot \mathbf{\Theta}^{\mathrm{T}}_{1_{[1\times N]}}, \tag{4}$$

where the N-by-1 vector $\mathbf{p}_1$ is the injected power; the N-by-N matrix $\mathbf{B}_1$ represents topology-dependent susceptance matrix in post-outage power flow. T denotes the transpose operation of matrix.

Four assumptions are made when we design the algorithms. First, the fast system dynamics are assumed to be well damped, thus bringing the system into a quasi-stable state after line outage occurs. In this quasi-stable system, any fast oscillation in the phasor angle can be filtered out with a low-pass filter [13]. Second, considering the smart grid topology as graph, we assume that no islanding scenario exists when outage occurs. This means that the underlying graph will not become disconnected [6, 8]. Under the timescale of outage, the injected power and the received power remain balanced under steady state either in a pre-outage system or in a post-outage system when outage occurs. Third, the number of PMU measurements (M) is larger than the

number of line outages (F). As a result, M is greater than F. Finally, line outages, i.e., broken lines, are the primary cause of transmission system failure [4, 20]. We simply model the outage line (l) by making the $b_l = 0$. Other types of failures will be considered in the future.

According to literature [21], given the random bus loads with the mean value at the nominal bus loading, using the cumulant method, power flow $\mathbf{p}_i$ ($i \in \{0, 1\}$) can be considered to follows Gaussian distribution under the hypothesis either pre-outage scenario ($\mathcal{L}_0$) or post-outage scenario ($\mathcal{L}_1$), which is shown as follow:

$$\mathbf{p}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad i \in \{0, 1\}, \tag{5}$$

where $\mathcal{N}$ represents the Gaussian distribution. When $i = 0$, $\mathbf{p}_i$ represents the pre-outage injected power; when $i = 1$, $\mathbf{p}_i$ represents the post-outage injected power. $\boldsymbol{\mu}$ represents N-by-1 mean vector of Gaussian distribution; $\boldsymbol{\Sigma}$ represents N-by-N covariance matrix.

Applying (3) and (4), we obtain the phase angle data $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}_0 = \mathbf{B}_0^{-1}{}_i \mathbf{p}_0, \boldsymbol{\theta}_1^{\mathrm{T}} = \mathbf{B}_1^{-1}{}_i \mathbf{p}_1. \tag{6}$$

under the pre-outage scenario ($i = 0$) and the post-outage scenario ($i = 1$). Based on the linear equation (6), the $\boldsymbol{\theta}$ is also Gaussian under the hypothesis $\mathcal{L}_0$ ($\mathcal{L}_1$), with mean and covariances ($\boldsymbol{\mu_0}$, $\boldsymbol{\Sigma_0}$) (respectively, ($\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_1$)), as follows:

$$\boldsymbol{\theta}_i \sim \mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right), \text{under } \mathcal{L}_i, \quad i \in \{0, 1\}. \tag{7}$$

in which $\boldsymbol{\mu}_i = \mathbf{B}_i^{-1}\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}_i = \mathbf{B}_i^{-1}\boldsymbol{\Sigma}\mathbf{B}_i^{-1}$ ($i \in \{0, 1\}$). Note that $(\mathbf{B}_i^{-1})^{\mathrm{T}} = \mathbf{B}_i^{-1}$ holds, due to the N-by-N symmetric matrix.

Let construct the spatial-temporal matrix of phasor angle data ($\boldsymbol{\Theta}$) as an $M \times N$ matrix of M observation samples on each of N phasor angle measurement nodes. Define the $\boldsymbol{\Theta}$ matrix as shown follows:

$$\boldsymbol{\Theta} = [\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(N)}], \tag{8}$$

where the M-by-1 vector $\boldsymbol{\theta}^{(i)} = [\theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_M^{(i)}]^{\mathrm{T}}$ ($i = 1, 2, \ldots, N$) represents M observation samples of phasor angle.

3 Algorithms Design

3.1 ReTAD Algorithm Design

The proposed work analyzes the anomalous behavior of transmission line outages and models the spatial-temporal correlation based anomalous behavior using phasor

angle data matrix. Anomalous behaviors, such as transmission line outages, are a primary reliability concern of smart grid. Based on the N_{bus} power system, combined with GPS-synchronized PMUs and Supervisory Control and Data Acquisition (SCADA) with phasor angle value, the collected phasor angle data is utilized to form a temporal-spatial matrix $\boldsymbol{\Theta} \in \mathbb{R}^{M \times N}$, where M represents the number of temporal observation samples; N presents the total number of measurements of phasor angle ($N \subseteq N_{\text{bus}}$). Given the linear DC power flow model [22], the measure vector of phasor angle $\boldsymbol{\theta}_{[1\times N]} \in \boldsymbol{\Theta}$ establish the bridge between injected power vector and topology-dependent susceptance matrix. Under the timescale of line outages, the power loading does not change considerably between a pre- and a post-outage system. The behavior of transmission line outages is thus strongly affected by the changes of phasor angle data.

Because the power system is an interconnected network, transmission line outages result in spatial-related phase angle data changes. Therefore, some spatial relationship between the measured phasor angle data from different buses will be changed due to line outages. To characterize this spatial relationship, we use the analytical method of estimating the correlation between the measured phasor angle data from different buses, called correlation coefficient $R_{\boldsymbol{\theta}^{(i)},\boldsymbol{\theta}^{(j)}}$, as follows:

$$R_{\boldsymbol{\theta}^{(i)},\boldsymbol{\theta}^{(j)}} = \frac{\text{Cov}(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(j)})}{S_{\boldsymbol{\theta}^{(i)}} \cdot S_{\boldsymbol{\theta}^{(j)}}}, \quad i, j = 1, 2, \ldots, N, \tag{9}$$

where $\text{Cov}(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(j)})$ represents the covariance between measured phasor angle samples $\boldsymbol{\theta}^{(i)}$ and $\boldsymbol{\theta}^{(j)}$; $S_{\boldsymbol{\theta}^{(i)}}$ and $S_{\boldsymbol{\theta}^{(j)}}$ are sample standard deviations of $\boldsymbol{\theta}^{(i)}$ and $\boldsymbol{\theta}^{(j)}$, respectively. The $\boldsymbol{\theta}$ is the observation samples of phasor angle which is defined in Eq. (8).

Given the M observation samples, the correlation coefficient matrix $\mathbf{R}_{[N\times N]}$ is obtained from Eq. (9). Once the correlation coefficient matrix is obtained, we adopt the similarity distance [23] to measure the distance between $\mathbf{R}_t$ and $\mathbf{R}_0$, as defined by

$$D_s = 1 - \frac{\text{Tr}(\mathbf{R}_t \mathbf{R}_0)}{\|\mathbf{R}_t\|_2 \|\mathbf{R}_0\|_2}, \tag{10}$$

where $\text{Tr}(\cdot)$ denotes trace operation; $\|\cdot\|_2$ denotes norm-2 operation; $\mathbf{R}_t$ represents the correlation coefficient matrix from target phasor angle data matrix; $\mathbf{R}_0$ represents the correlation coefficient matrix from pre-outage phasor angle data matrix. This D_s can be considered as an "anomalous score", which indicates the anomaly of the target instance $\boldsymbol{\Theta}_t$. D_s can be also viewed as the influence of the line outage phasor angle samples. A higher D_s score, which is close to one, means that the target measured phasor angle data are more likely to be anomalous. Given the sampling windows and target measured phasor angle data, if its D_s score is above some threshold, we then detect the measured phasor angle data in the sampling windows as an anomalous behavior. The complexity of anomalous score is $\mathcal{O}(N^2)$.

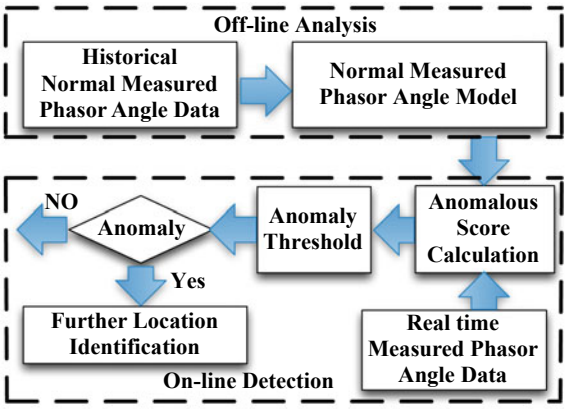


Fig. 1 Framework of real time anomaly detection

To overcome the issue of gigantic measured phasor angle data volume, we present the framework of real-time anomaly detection consisting of off-line analysis on historical measured phasor angle data and obtains the $\mathbf{R}_0$ under the outage-free scenario and on-line detection algorithm for gigantic measured phasor angle data. Anomaly detection in power system is used to distinguish normal and abnormal measured phasor angle samples. A fault such as transmission line outage inside the system is likely to trigger further catastrophic failures, causing billions of dollars loss in revenue [4]. As illustrated in Fig. 1, the framework of real time anomaly detection can be divided into two stages: the off-line analysis stage, which creates a model of the normal condition of measured phasor angle data, and the on-line detection stage, which detects anomalies by comparing the current (actual) with the modeled one via anomalous behavior model.

3.1.1 Off-Line Analysis

In the off-line analysis stage, the input data is all the historical monitored normal phasor angle samples. According to those phasor angle samples and anomalous behavior model, we directly calculate the N-by-N correlation coefficient matrix $\mathbf{R}_0$ based on the covariance matrix $\mathrm{Cov}(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(j)})$ between measured phasor angle samples $\boldsymbol{\theta}^{(i)}$ and $\boldsymbol{\theta}^{(j)}$ $(i, j = 1, 2, \ldots, N)$, which is given by

$$\mathrm{Cov}(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^{(j)}) = E[(\boldsymbol{\theta}^{(i)} - E(\boldsymbol{\theta}^{(i)}))(\boldsymbol{\theta}^{(j)} - E(\boldsymbol{\theta}^{(j)}))^{\mathrm{T}}]. \tag{11}$$

Therefore, the output of the off-line analysis is the normal correlation coefficient matrix $\mathbf{R}_0$, which is used to calculate the anomalous behavior (see Eq. 10) in the on-line detection stage.

3.1.2 On-Line Detection

During the on-line detection stage, the real time phasor angle data is used to calculate the current correlation coefficient matrix ($\mathbf{R}_t$). Combining with the normal correlation coefficient matrix in the off-line analysis, when an anomaly occurs, the anomalous score, which was proposed in (10), will be different between current and normal measured phasor angle samples.

Based on the above discussion, the estimation of high dimensional covariance matrix is a fundamental problem in Eq. (10), which uses the correlation coefficient matrix to estimate the anomalous score. However, in order to obtain the accurate unbiased estimation of high dimensional $N \times N$ covariance matrices, a large number of samples ($M \to \infty$) have to be drawn. Because a phasor data transmission rate is around 60 samples per second [17], unbiased estimation of the covariance matrix should take lots of time with huge volume of data samples. Therefore, we need to deploy a fast estimation of high dimensional N-by-N covariance matrices $\mathbf{\Sigma}_\Theta$ under small sample size, such as $N_s \ll M$.

Inspired by Ledoit-Wolf Shrinkage (LWS) method [24], we deploy a covariance estimator that is not only suitable for small sample size (N_s) and large number of measured phasor angle (N), but at the same time is also low computational complexity. Given this phasor angle data matrix as defined in 8 and N_s observation samples, to estimate the covariance matrix $\mathbf{\Sigma}_\Theta$, our goal is to find an estimator $\hat{\mathbf{\Sigma}}(\{\boldsymbol{\theta}_i\}_{i=1}^{N_s})$, which minimizes the mean squared error (MSE) (or expected quadratic error):

$$E[\|\hat{\mathbf{\Sigma}}(\{\boldsymbol{\theta}_i\}_{i=1}^{N_s}) - \mathbf{\Sigma}_\Theta\|^2]. \tag{12}$$

Seeking to minimize MSE, we decompose (12) into variance and squared bias, which is shown as follow:

$$E[\|\hat{\mathbf{\Sigma}} - \mathbf{\Sigma}_\Theta\|^2] = \|E[\hat{\mathbf{\Sigma}} - \mathbf{\Sigma}_\Theta]\|^2 + E[\|\hat{\mathbf{\Sigma}} - E[\hat{\mathbf{\Sigma}}]\|^2]. \tag{13}$$

Minimizing MSE and finding the optimal $\hat{\mathbf{\Sigma}}^*$ is to get the optimal trade-off between error due to bias (the first term on the right-hand side of (13)) and error due to variance (the second term on the right-hand side of (13)). Hence, we firstly construct the unbiased estimator of $\mathbf{\Sigma}_\Theta$, called sample covariance matrix $\boldsymbol{S}$, which is defined as follows:

$$\boldsymbol{S} = \frac{1}{N_s}\sum_{i=1}^{N_s}\boldsymbol{\theta}_i\boldsymbol{\theta}_i^{\mathrm{T}}, \tag{14}$$

where N_s represents the number of observation samples. Due to the unbiased estimator of $\mathbf{\Sigma}_\Theta$, $E[\boldsymbol{S}] = \mathbf{\Sigma}_\Theta$ holds. However, this estimator cannot achieve minimum MSE because it has high variance (see (13)). Secondly, according to Chen et al. [25], we use the most well-conditioned estimator of $\mathbf{\Sigma}_\Theta$ with low variance, which is defined as follows:

$$\boldsymbol{U} = \frac{\mathrm{Tr}(\boldsymbol{S})}{N}\boldsymbol{I}, \tag{15}$$

where $\boldsymbol{I}$ represents the identity matrix. $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. This estimator leads to reduced variance but sacrifices of bias. To minimize MSE in (13), we apply the shrinkage coefficient (ρ) and construct the linear combination estimator form to leverage between bias and variance, as defined follows:

$$\hat{\boldsymbol{\Sigma}} = \rho \boldsymbol{U} + (1-\rho)\boldsymbol{S}. \tag{16}$$

The shrinkage coefficient ($\rho, 0 \leq \rho \leq 1$) plays a critical role for estimator $\hat{\boldsymbol{\Sigma}}$. The optimal ρ provides a reasonable trade-off between low bias and low variance of estimator.

Therefore, the following *MSE Design Optimization* problem can be formulated intuitively to determine optimal shrinkage coefficient ρ in order to meet the minimum MSE requirement.

Problem 1 (*MSE Design Optimization*) Given the sample covariance matrix $\boldsymbol{S}$, the MSE design optimization problem is formulated as follows:

$$\begin{aligned} \mathrm{Min}_{\rho} \quad & E[\|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}_{\Theta}\|^2] \\ \text{s.t.} \quad & \hat{\boldsymbol{\Sigma}} = \rho \boldsymbol{U} + (1-\rho)\boldsymbol{S}, \end{aligned} \tag{17}$$

where $\boldsymbol{U}$ and $\boldsymbol{S}$ follow (15) and (14), respectively.

Note that the shrinkage coefficient ρ is nonrandom. The physical meaning of ρ is a properly normalized measure of the error of the sample covariance matrix $\boldsymbol{S}$, which was discussed in Lemma 2. To illustrate the Lemma 2, we will introduce Lemma 1 first.

Lemma 1 *Define* $\gamma^2 = \|\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2$, $\xi^2 = E[\|\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta}\|^2]$, $\lambda^2 = E[\|\boldsymbol{S} - \boldsymbol{U}\|^2]$. *Then the relationship* $\lambda^2 = \gamma^2 + \xi^2$ *holds.*

Proof

$$\begin{aligned} \lambda^2 &= E[\|\boldsymbol{S} - \boldsymbol{U}\|^2] = E[\|\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta} + \boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2] \\ &= E[\|\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta}\|^2] + E[\|\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2] \\ &\quad + 2E[(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})^{\mathrm{T}}(\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U})] \end{aligned} \tag{18}$$

Because (15) holds, the equivalent formulation: $E[\|\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2] = \|\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2$ holds. Thus,

$$\begin{aligned} \lambda^2 = E[\|\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta}\|^2] + \|\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}\|^2 + \mathrm{Tr}(E[(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})^{\mathrm{T}}(\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}) \\ + (\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U})^{T}(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})]) \end{aligned} \tag{19}$$

holds. Because of the unbiased estimator $\boldsymbol{S}$, $E[\boldsymbol{S}] = \boldsymbol{\Sigma}_{\Theta}$ holds. The equation $\mathrm{Tr}(E[(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})^{\mathrm{T}}(\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U}) + (\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U})^{\mathrm{T}}(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})])$ can rewrite as $\mathrm{Tr}(E[(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})^{\mathrm{T}}(\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U})] + E[(\boldsymbol{\Sigma}_{\Theta} - \boldsymbol{U})^{\mathrm{T}}(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})])$. Given (14), $(\boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta})^{\mathrm{T}} = \boldsymbol{S} - \boldsymbol{\Sigma}_{\Theta}$ holds.

So, $E[(\boldsymbol{S}-\boldsymbol{\Sigma}_\Theta)^{\mathrm{T}}(\boldsymbol{\Sigma}_\Theta-\boldsymbol{U})]=E[(\boldsymbol{S}-\boldsymbol{\Sigma}_\Theta)(\boldsymbol{\Sigma}_\Theta-\boldsymbol{U})]=E[\boldsymbol{S}\boldsymbol{\Sigma}_\Theta-\boldsymbol{S}\boldsymbol{U}-\boldsymbol{\Sigma}_\Theta^2+\boldsymbol{\Sigma}_\Theta\boldsymbol{U}]$. Due to $E[\boldsymbol{S}]=\boldsymbol{\Sigma}_\Theta$ holds, $E[\boldsymbol{S}\boldsymbol{\Sigma}_\Theta-\boldsymbol{S}\boldsymbol{U}-\boldsymbol{\Sigma}_\Theta^2+\boldsymbol{\Sigma}_\Theta\boldsymbol{U}]$ equals zero. Lemma 1 holds. □

Lemma 2 *Given* $\gamma^2=\|\boldsymbol{\Sigma}_\Theta-\boldsymbol{U}\|^2$, $\xi^2=E[\|\boldsymbol{S}-\boldsymbol{\Sigma}_\Theta\|^2]$, $\lambda^2=E[\|\boldsymbol{S}-\boldsymbol{U}\|^2]$, *and* $\lambda^2=\gamma^2+\xi^2$, *the optimal shrinkage coefficient* ρ^* *in (17) is shown as follows:*

$$\rho^*=\frac{E[\|\boldsymbol{S}-\boldsymbol{\Sigma}_\Theta\|^2]-E[\|\hat{\boldsymbol{\Sigma}}^*-\boldsymbol{\Sigma}_\Theta\|^2]}{E[\|\boldsymbol{S}-\boldsymbol{U}\|^2]}=\frac{\xi^2}{\lambda^2}, \tag{20}$$

where $\hat{\boldsymbol{\Sigma}}^*$ *represents the optimal estimation of* $\boldsymbol{\Sigma}_\Theta$ *with minimum MSE.*

Proof Using the algebra knowledge, Lemma 1, and $E[\boldsymbol{S}]=\boldsymbol{\Sigma}_\Theta$, we rewrite the objective function as follows:

$$\begin{aligned}E[\|\hat{\boldsymbol{\Sigma}}-\boldsymbol{\Sigma}_\Theta\|^2]&=\rho^2\|\boldsymbol{\Sigma}_\Theta-\boldsymbol{U}\|^2+(1-\rho)^2E[\|\boldsymbol{S}-\boldsymbol{\Sigma}_\Theta\|^2]\\&=\rho^2\gamma^2+(1-\rho)^2\xi^2.\end{aligned} \tag{21}$$

Applying the first-order condition for ρ: $2\rho\gamma^2-2(1-\rho)\xi^2=0$, the optimal solution is shown follows: $\rho^*=\frac{\xi^2}{\gamma^2+\xi^2}=\frac{\xi^2}{\lambda^2}$. At this optimum point, the objective function of *Problem MSE Design Optimization* can be written by

$$E[\|\hat{\boldsymbol{\Sigma}}^*-\boldsymbol{\Sigma}_\Theta\|^2]=(\frac{\xi^2}{\lambda^2})^2\gamma^2+(\frac{\gamma^2}{\lambda^2})^2\xi^2=\frac{\gamma^2\xi^2}{\lambda^2}. \tag{22}$$

Thus, this Lemma holds. □

Based on physical meaning of ρ, the solution of *Problem MSE Design Optimization* should guarantee the following equations.

$$\begin{aligned}\hat{\boldsymbol{\Sigma}}^*&=\frac{\xi^2}{\lambda^2}\boldsymbol{U}+\frac{\gamma^2}{\lambda^2}\boldsymbol{S}\\E[\|\hat{\boldsymbol{\Sigma}}^*-\boldsymbol{\Sigma}_\Theta\|^2]&=\frac{\gamma^2\xi^2}{\lambda^2}.\end{aligned} \tag{23}$$

According to (23), we get the intuitive sense: if $\boldsymbol{S}$ is accurate, then you don't need shrink too much; otherwise, you should shrink it a lot to reduce the MSE.

However, the estimator defined by (17) is optimal but cannot be implemented practically. The reason is that the optimal solution (ρ^*), which is specified by (20), depends on the unknown matrix $\boldsymbol{\Sigma}_\Theta$. Ledoit and Wolf [24] proposed an approximately optimal ρ^*_{LW} ($0\le\rho^*_{LW}\le 1$) without any knowledge of the sample distribution, as shown follows:

$$\rho^*_{LW}=\frac{\sum_{i=1}^{N_s}\|\boldsymbol{\theta}_i\boldsymbol{\theta}_i^{\mathrm{T}}-\boldsymbol{S}\|^2}{N_s^2[\mathrm{Tr}(\boldsymbol{S}^2)-\frac{\mathrm{Tr}^2(\boldsymbol{S})}{N}]}, \tag{24}$$

Algorithm 1 The Real Time Anomaly Detection (ReTAD)

Initialization: Off-line Analysis
  Construct the normal phasor angle data matrix $\mathbf{\Theta}_0 = [\boldsymbol{\theta}_1^{\mathrm{T}}; \boldsymbol{\theta}_2^{\mathrm{T}}; \ldots; \mathbf{x}_M^{\mathrm{T}}]$ based on history phasor angle samples.
  Compute the correlation coefficient matrix $\mathbf{R}_0$ under outage-free scenario
  Applying the fast RBLWS covariance estimator to obtain the optimal sampling window with minimum MSE.
On-line Detection
  for the time t_i at i time step, starting from $i = 1$ **to** n time steps **do**
    Step 1: Construct phasor angle data matrix $\mathbf{\Theta}_{t_i} = [\boldsymbol{\theta}_1^{\mathrm{T}}; \boldsymbol{\theta}_2^{\mathrm{T}}; \ldots; \boldsymbol{\theta}_{N_s}^{\mathrm{T}}]$
    Step 2: Estimate the covariance matrix $\hat{\mathbf{\Sigma}}_{t_i}$ by RBLWS estimator.
    Step 3: Compute the current correlation coefficient matrix $\mathbf{R}_{t_i}$
    Step 4: Compute the anomalous score D_s by applying Eq. (10).
    if $D_s \geq$ threshold
      The samples are marked as anomaly event and continue to detect
    else Go to step 1 and continue to detect
    end if
  end for
return All anomalous samples are marked

where N_s is the number of observation samples; the N-by-1 vector $\boldsymbol{\theta}_i = [\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}]^{\mathrm{T}}$ $(i = 1, 2, \ldots, N_s)$ represents the i observation sample of phasor angle data in an N dimensional space. Applying (24) into (16), the LWS estimator $\hat{\mathbf{\Sigma}}_{LW}^{*}$ is derived. According to [24], the asymptotic condition is shown as follows: when both $N_s, N \to \infty$ and $N/N_s \to \varsigma$ $(0 < \varsigma < \infty)$, the Eq. (24) converges to (20) in probability without regard for the sample distribution. Further more, to improve the performance of estimator, the modified shrinkage coefficient ρ_{LW}^{**} [24] is defined as

$$\rho_{LW}^{**} = \min(\rho_{LW}^{*}, 1). \tag{25}$$

Thus, the optimal LWS covariance estimator $\hat{\mathbf{\Sigma}}_{LW}^{**}$ can be derived by applying (25) in the Eq. (16).

Because the θ follows Gaussian distribution (see (44)), the N-by-1 vector $\mathbf{x}_i = [\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}]^{\mathrm{T}}$ $(i = 1, 2, \ldots, N_s)$ is also a Gaussian vector. We improve the LWS estimation method under the Gaussian model. Inspired by Rao-Blackwell LWS estimation theory [25, 26], given independent N-dimensional Gaussian vectors $\{\mathbf{x}_i\}_{i=1}^{N_s}$ and sample covariance matrix $\boldsymbol{S}$, the conditioned expectation of the LWS covariance estimator ($\hat{\mathbf{\Sigma}}_{LWRB} = E[\hat{\mathbf{\Sigma}}_{LW}|\boldsymbol{S}]$) is never worse than the original estimator ($\hat{\mathbf{\Sigma}}_{LW}$) under any convex loss criterion,[1] as defined by

$$E[\|\hat{\mathbf{\Sigma}}_{LWRB} - \mathbf{\Sigma}_{\Theta}\|^2] \leq E[\|\hat{\mathbf{\Sigma}}_{LW} - \mathbf{\Sigma}_{\Theta}\|^2]. \tag{26}$$

Meanwhile, the optimal ρ_{LWRB}^{*} is shown as follows:

$$\rho_{LWRB}^{*} = \frac{(N_s - 2)\mathrm{Tr}(\boldsymbol{S}^2) + \mathrm{Tr}^2(\boldsymbol{S})}{N_s(N_s + 2)\left(\mathrm{Tr}(\boldsymbol{S}^2 - \frac{\mathrm{Tr}^2(\boldsymbol{S})}{N})\right)}. \tag{27}$$

[1] The Rao-Blackwell estimation theorem is referred to [25, 26] for further discussion.

Similarly to the ρ^{**}_{LW}, the updated optimal shrinkage coefficient is defined by $\rho^{**}_{LWRB} = \min(\rho^{*}_{LWRB}, 1)$. The optimal estimator is thus derived as

$$\hat{\boldsymbol{\Sigma}}^{*}_{LWRB} = \rho^{**}_{LWRB}\boldsymbol{U} + (1 - \rho^{**}_{LWRB})\boldsymbol{S}. \tag{28}$$

Using the fast covariance matrix estimator, called RBLWS estimator (see (28)), (9), (10), and historical correlation coefficient matrix $\mathbf{R}_0$, we calculate the anomalous score at real time. If the anomalous score is higher than the threshold, the system show alert that the abnormal event occurs; other wise, no anomaly event occur. The detailed algorithm is shown in *The Real Time Anomaly Detection (ReTAD) Algorithm* .

Based on the ReTAD algorithm, choosing an appropriate threshold is very critical. The reason is that the threshold serves as an effective tuning knob between anomaly detection rate and false positive rate.

3.2 LIS Algorithm Design

Once the anomaly behavior of line outages is detected at real-time, it needs to do identification of the multiple-line outages in order to avoid catastrophic outages. Inspired by the large change sensitivity technique [27, 28], the vector $\boldsymbol{\delta}_h$ with $h \in L = \{1, 2, \ldots, F\}$ represents the susceptance sensitivity values between the pre- and the post-outage system. F is the number of simultaneous line outages. The variations of the susceptance between the pre- and the post-outage system are represented by $\boldsymbol{\Delta B}$, as follows:

$$\boldsymbol{\Delta B} = \sum_{h=1}^{F} \mathbf{q}_h \boldsymbol{\delta}_h \mathbf{q}_h^{\mathrm{T}} = \mathbf{Q}_{NF}\mathbf{Z}_{\delta}\mathbf{Q}_{NF}^{\mathrm{T}}, \tag{29}$$

where the h-th diagonal entry of the diagonal matrix $\mathbf{Z}_{\delta}$ equals δ_h, $h \in \{1, 2, \ldots, F\}$. The incidence matrix $\mathbf{Q}_{NF}$ is formed from N-by-1 vectors $\mathbf{q}_h$ $(h = 1, 2, \ldots, F)$ as its columns. Denote the selection matrix as $\mathbf{S}_{[L\times F]}$. Note that each column of $\mathbf{S}_{[L\times F]}$ is a L-by-1 vector of zero entries, except for ith row ($i \in \{1, 2, \ldots, F\}$), which equals one. The matrix $\mathbf{Q}_{NF}$ is thus formed from $\mathbf{Q}_{NL}\mathbf{S}_{[L\times F]}$. Therefore, with unknown simultaneous line outages (F), we derive the updated susceptance matrix $\mathbf{B}'$ as follows:

$$\begin{aligned}\mathbf{B}'_{[N\times N]} &= \mathbf{B}_{0_{[N\times N]}} + \Delta\mathbf{B}_{[N\times N]} \\ &= \mathbf{B}_{0_{[N\times N]}} + \mathbf{Q}_{NF_{[N\times F]}}\mathbf{Z}_{\delta_{[F\times F]}}\mathbf{Q}^{\mathrm{T}}_{NF_{[N\times F]}},\end{aligned} \tag{30}$$

where $\mathbf{B}_0$ is the susceptance matrix in a pre-outage system.

$\mathbf{B}'^{-1}$ is derived by applying the matrix transpose technique from the Woodbury formula [29]:

$$\mathbf{B}'^{-1} = \mathbf{B}_0^{-1} - \mathbf{B}_0^{-1}\mathbf{Q}_{NF}(\mathbf{Z}_\delta^{-1} + \mathbf{Q}_{NF}^{\mathrm{T}}\mathbf{B}_0^{-1}\mathbf{Q}_{NF})^{-1}\mathbf{Q}_{NF}^{\mathrm{T}}\mathbf{B}_0^{-1}. \tag{31}$$

Similarly, the definition of variations in the phasor angle between a pre- and a post-outage system, $\mathbf{\Delta\Theta}$, is shown as:

$$\mathbf{\Delta\Theta} = \mathbf{\Theta}' - \mathbf{\Theta}_0, \tag{32}$$

where $\mathbf{\Theta}_0$ and $\mathbf{\Theta}'$ denote the phasor angle in the pre- and the post-outage system, respectively. And then $\mathbf{\Delta\Theta}$ is expressed with (3), (4), (31), and (32):

$$\begin{aligned}\mathbf{\Delta\Theta}_{[N\times 1]} = -\mathbf{B}_{0_{[N\times N]}}^{-1}\mathbf{Q}_{NF_{[N\times F]}}\big(\mathbf{Z}_{\delta_{[F\times F]}}^{-1} \\ + \mathbf{Q}_{NF_{[N\times F]}}^{\mathrm{T}}\mathbf{B}_{0_{[N\times N]}}^{-1}\mathbf{Q}_{NF_{[N\times F]}}\big)^{-1}\mathbf{Q}_{NF_{[N\times F]}}^{\mathrm{T}}\mathbf{\Theta}_{0_{[N\times 1]}}.\end{aligned} \tag{33}$$

For simplicity's sake, denote the N-by-F matrix $\mathbf{X}_{NF}$ as

$$\mathbf{X}_{NF} = -\mathbf{B}_0^{-1}\mathbf{Q}_{NF}, \tag{34}$$

and denote the F-by-1 vector $\boldsymbol{\beta}_F$ as

$$\boldsymbol{\beta}_{F_{[F\times 1]}} = \big(\mathbf{Z}_{\delta_{[F\times F]}}^{-1} + \mathbf{Q}_{NF_{[N\times F]}}^{\mathrm{T}}\mathbf{B}_{0_{[N\times N]}}^{-1}\mathbf{Q}_{NF_{[N\times F]}}\big)^{-1}\mathbf{Q}_{NF_{[N\times F]}}^{\mathrm{T}}\mathbf{\Theta}_{0_{[N\times 1]}}. \tag{35}$$

The $\mathbf{\Delta\Theta}$ can now be rewritten:

$$\mathbf{\Delta\Theta}_{[N\times 1]} = \mathbf{X}_{NF_{[N\times F]}}\boldsymbol{\beta}_{F_{[F\times 1]}}. \tag{36}$$

According to the selection matrix $\mathbf{S}_{[L\times F]}$, $\mathbf{Q}_{NF} = \mathbf{Q}_{NL}\mathbf{S}_{[L\times F]}$ holds. Applying the similar definition in (34), we define matrix $\mathbf{X}_{NL}$ with respect to all transmission lines L as follows:

$$\mathbf{X}_{NL} = -\mathbf{B}_0^{-1}\mathbf{Q}_{NL}, \tag{37}$$

According to the M PMUs measurements $\mathbf{\Theta_M} = [\Theta_1, \ldots, \Theta_M]$, the selection matrix for M measurements should be denoted as $\mathbf{K}_{[M\times N]}$. Note that each row of $\mathbf{K}$ denotes a 1-by-N vector of zeros, except for the ith column corresponding to Θ_i, $i \in \{1, 2, \ldots, M\}$, which equals one. We left multiply both sides of (36) by $\mathbf{K}$ to obtain $\mathbf{\Delta\Theta}_M$ (known as the *outage identification equation*) as follows:

$$\mathbf{\Delta\Theta}_{M_{[M\times 1]}} = \mathbf{X}_{MF_{[M\times F]}}\boldsymbol{\beta}_{F_{[F\times 1]}}, \tag{38}$$

where $\mathbf{X}_{MF_{[M\times F]}} = \mathbf{K}_{[M\times N]}\mathbf{X}_{NF_{[N\times F]}}$. The columns from $\mathbf{X}_{MF_{[M\times F]}}$ corresponding to line outage locations are determined by the known matrix $\mathbf{X}_{ML}$. Similarly, left multiplying both sides of (37), $\mathbf{X}_{ML}$ is obtained as follows:

$$\mathbf{X}_{ML_{[M\times L]}} = \mathbf{K}_{[M\times N]}\mathbf{X}_{NL_{[N\times L]}}. \tag{39}$$

Therefore, $\mathbf{X}_{MF}$ is comprised of F selection columns from $\mathbf{X}_{ML}$ that correspond to line outage locations.

Because the susceptance matrix attributes its changes to line outages, we must select appropriate columns (F) from $\mathbf{X}_{ML}$ to satisfy the (38). This equation establishes a relationship between the measured response of the network matrix ($\mathbf{X}_{MF}$) in a post-outage system with deviations from line outages (a function of $\boldsymbol{\delta}$ based on the definition of $\boldsymbol{\beta}_F$). In a nutshell, the issue of location identification for multiple line outages transforms into analysis of the outage identification Eq. (38) and locating the multiple simultaneous line outages. To clarify the LIS algorithm design, we define the solution of line outage locations as follows:

Definition 1 (*Solution*) Given the limited PMUs, a solution (s) is a set of selection columns from $\mathbf{X}_{ML}$ that correspond to either one or more line outage locations satisfying (38).

3.2.1 Problem Reformulation Using Ambiguity Group

Distinct line outages lead to the same PMUs measurements, which are commonly encountered in a real-world scenario. Thus, addressing the unique locations of multiple simultaneous line outages with the limited PMUs is difficult. Inspired by the ambiguity group theory [5, 28, 30, 31], we develop the LIS algorithm to identify the most likely line outage locations. Based on the principle of parsimony [32], the most likely set of multiple line outage locations is the smallest set of multiple line locations, which can be identified by limited PMUs.

We first introduce the important definition known as ambiguity group (A) in LIS algorithm design.

Definition 2 (*Ambiguity group*) The ambiguity group is a group of w solutions $(s_1, s_2, \ldots, s_w)$, described in Definition 1, that produce the same PMUs measurements in a post-outage system. These same measurements prevent the unique line outage locations from being further identified by limited PMUs. Note that $A = \{s_1, s_2, \ldots, s_w\}$.

Next, according to the *Outage Identification Equation* (38), location identification of multiple line outages is formulated to determine which set (or sets) of columns from $\mathbf{X}_{ML}$ can satisfy (38). Based on (38), if vector $\boldsymbol{\Delta\Theta}_M$ is a vector of zeros, then no line outages can be identified by the given PMUs. If, however, it is not a vector of zeros, at least one line outage can be identified. In this scenario, more than one column in $\mathbf{X}_{ML_{[M\times L]}}$ may satisfy (38).

A naive method involves adopting an exhaustive search (ES) algorithm to examine all columns in $\mathbf{X}_{ML}$. Given both the line outages (F) and the total number of lines (L), the ES algorithm requires the number of operations to be $\mathcal{O}\left(\sum_{i=1}^{F}\binom{L}{i}\right)$. It shows that ES algorithm has a high computational complexity. A more effective method for location identification is expected to reduce the computational cost.

Because the required set (or sets) of columns from $\mathbf{X}_{ML}$ is not unique, we hope to determine the smallest set of multiple outage line locations. According to the ambiguity group theory [28, 31], the ambiguity group with minimum size can provide the information about the solvability of location identification problem with respect to each solution based on the given PMUs measurements. Denote the ambiguity group with minimum size as $A_{\min} = \{s_1, s_2, \ldots, s_t\}$. Thus, the goal of LIS algorithm is to find the smallest set of multiple line outage locations ($\min\{|s_1|, |s_2|, \ldots, |s_t|\}$, where $|s_i|$ ($i \in \{1, 2, \ldots, t\}$) is the size of solution s_i). The possible solution s_i corresponds to the set (or sets) of columns from $\mathbf{X}_{ML}$. To analyze characteristics of those possible solutions of line outage locations, we introduce Lemma 3 as follows:

Lemma 3 *Any two solutions of line outage locations from the ambiguity group form linearly dependent columns of* $\boldsymbol{X}_{ML}$.

Proof Due to limited PMUs, multiple solutions produce the same PMU measurements, which obey the *Outage Identification Equation* (38). We arbitrarily select k ($k \geq 2$) solutions from multiple solutions. Based on Definitions 1 and 2, the Eq. (40) holds for each possible solution s_i.

$$\mathbf{X}_{Ms_i}\boldsymbol{\beta}_{s_i} = \boldsymbol{\Delta}\theta_M, i = 1, 2, \ldots, k, \tag{40}$$

where s_i represents the set (or sets) of columns from $\mathbf{X}_{ML}$, known as the solution of line outage locations. $\mathbf{X}_{Ms_i}$ is comprised of selection columns (s_i) from $\mathbf{X}_{ML}$. $\boldsymbol{\beta}_{s_i}$ corresponds to s_i-by-1 vector, which has the similar definition in (35). Due to Eq. (40) holds, we always find a set of λ_i ($i = 1, 2, \ldots, k$) and let Eq. (41) hold.

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{\Delta}\theta_M = 0. \tag{41}$$

Applying (40) to (41), then,

$$\left[\mathbf{X}_{Ms_1}\ \mathbf{X}_{Ms_2}\ \ldots\ \mathbf{X}_{Ms_k}\right] \begin{bmatrix} \lambda_1\beta_{\mathbf{s_1}} \\ \lambda_2\beta_{\mathbf{s_2}} \\ \vdots \\ \lambda_k\beta_{\mathbf{s_k}} \end{bmatrix} = \mathbf{0}, \tag{42}$$

where $\sum_{i=1}^{k} \lambda_i = 0$, $\lambda_i \neq 0$, $i = 1, 2, \ldots, k$. Note that $\boldsymbol{\beta}_{s_i} \neq 0$. Because the vector $\left[\lambda_1\beta_{\mathbf{s_1}}\ \lambda_2\beta_{\mathbf{s_2}}\ \ldots\ \lambda_k\beta_{\mathbf{s_k}}\right]'$ exists with nonzero coefficients, a set of solutions corresponding to columns from $\mathbf{X}_{ML}$ are linearly dependent. This lemma holds. □

Given the limited PMUs, the following *LIS* problem can be formulated intuitively as an optimization problem to determine the smallest set of line outage locations that satisfying (38).

Problem 2 (*LIS*) Given both the system topology and the limited PMUs, the LIS problem is formulated as follows:

$$\begin{aligned} &\text{minimize } F \\ &\text{subject to } \boldsymbol{\Delta\Theta}_M = \mathbf{X}_{MF}\boldsymbol{\beta}_F \end{aligned} \tag{43}$$

where F represents the possible solution's size ($s_i, i \in \{1, 2, \ldots, t\}$) that correspond to the number of concurrent outage lines.

3.2.2 Procedure of LIS Algorithm

Since the ambiguity group with minimum size provides the information about the solvability of location identification problem with respect to each solution based on the given PMUs measurements [28, 31], the critical task for determining the smallest set of line outage locations is to find the ambiguity group with minimum size ($A_{\min}$). According to characteristics of possible solutions (see Lemma 3), the ambiguity group with minimum size ($A_{\min}$) is comprised of the selection linearly dependent columns from $\mathbf{X}_{ML}$. Direct method to obtain those linearly dependent columns is also a combinatorial search with a high computational complexity. Thus, we analyze the dependencies between those columns from $\mathbf{X}_{ML}$ and adopt the linear column-dependence matrix $\mathbf{C}$ to represent an expansion of linearly dependent columns from $\mathbf{X}_{ML}$. Minimizing ambiguity group's size for location identification is thus computed with $\mathbf{C}$ in minimum form, which corresponds to the minimum number of non-zero entries in $\mathbf{C}$.

We develop the LIS algorithm procedure given in Algorithm 2. Minimizing the solution of line outage locations involves four basic steps. These steps are described as follows.

Step 1: ***Construct a dependence matrix***

To analyze the measurement vector's ($\boldsymbol{\Delta\Theta}_M$) dependency on the desired set(s) of $\mathbf{X}_{ML}$ columns, we define the concatenation matrix as $\mathbf{H}_{s_{[M\times(L+1)]}} = \left[\boldsymbol{\Delta\Theta}_{M_{[M\times 1]}}\ \mathbf{X}_{ML_{[M\times L]}}\right]$. To eliminate the dependence on $\boldsymbol{\Delta\Theta}_M$, the matrix $\hat{\mathbf{H}}_{s_{[M\times(L+1)]}}$ arises by eliminating the first row of $\mathbf{H}_s$ using the first column [33] as follows:

$$\hat{\mathbf{H}}_{s_{[M\times(L+1)]}} = \begin{bmatrix} 1_{[1\times 1]} & \mathbf{0}_{[1\times L]} \\ \Delta\hat{\boldsymbol{\Theta}}_{[(M-1)\times 1]} & \mathbf{H}_{[(M-1)\times L]} \end{bmatrix}, \tag{44}$$

where $\mathbf{H}_{[(M-1)\times L]}$ is the dependence matrix. This matrix reflects the dependencies of the desired columns from $\mathbf{X}_{ML}$.

Step 2: ***Minimize the form of a column-dependence matrix***

We define r as the rank of $\mathbf{H}_{[(M-1)\times L]}$, denoted as $\mathbf{H}$ for simplicity, and then decompose $\mathbf{H}$ into two linearly dependent sub-matrices:

Algorithm 2 The LIS Algorithm

Initialization
  Calculate $\mathbf{B}_0$ and $\mathbf{Q}_{NL}$ in a pre-outage power system
  By applying M PMUs measurements, $\mathbf{X}_{ML}$ is obtained from (39).
  Compute $\boldsymbol{\Delta\Theta}_M$.
Determine the smallest set of line outage locations
  Step 1: Construct dependence matrix $\mathbf{H}$
  Step 2: Minimize the form of a column-dependence matrix $\mathbf{C}$
    repeat
      if All sub-matrices of $\mathbf{C}$ satisfy Lemma 4 OR not satisfy Lemma 5
        The number of non-zero entries in $\mathbf{C}$ cannot be decreased.
      else
        Select $\mathbf{C}_{\text{sub}}$ from $\mathbf{C}$ satisfying Lemma 5
        Do swapping operation to obtain $\mathbf{C}_{\text{update}}$ via (51) and (52).
        if The number of non-zero entries in $\mathbf{C}_{\text{update}}$ less than the number of non-zero entries in $\mathbf{C}$
          Let $\mathbf{C} := \mathbf{C}_{\text{update}}$.
        else Go to the stage 12 until the number of non-zero entries in $\mathbf{C}$ cannot be decreased
        end if
      end if
    until Minimum form of $\mathbf{C}$ is achieved
  Step 3: Calculate possible solutions of line outage locations.
  Sort solutions $s_i (i = 1, 2, \ldots, w)$ in descending order by the size of solution
  Step 4: Select the best solution s^*

$$s^* = \arg \min_{i=1,2,\ldots,w} \|(\boldsymbol{\Delta\Theta}_{M_{\text{cal}_i}} - \boldsymbol{\Delta\Theta}_M)\|_2,$$

return The minimal solution with satisfying *Outage Identification Equation*.

$$\begin{aligned} \mathbf{H} &= [\mathbf{H}_1 \ \mathbf{H}_2] = \mathbf{H}_1[I \ \mathbf{C}], \\ \mathbf{H}_2 &= \mathbf{H}_1\mathbf{C}, \end{aligned} \tag{45}$$

where $\mathbf{H}_{1_{(M-1)\times r}}$ has a full column rank r. $\mathbf{C}_{r\times(L-r)}$ is defined as a column-dependence matrix, whose columns expand a set of basis columns from $\mathbf{H}_1$ into the corresponding co-basis columns from $\mathbf{H}_2$, which is defined as follows:

Definition 3 (*Basis*) The basis is a set of components that correspond to the columns in matrix $\mathbf{H}_1$.

Definition 4 (*Co-basis*) The co-basis is a set of components that correspond to the columns in matrix $\mathbf{H}_2$.

Based on the linear algebra theory [34], the rows in $\mathbf{C}$ correspond to components of the basis; the columns in $\mathbf{C}$ correspond to components of the co-basis. Thus, columns and rows of $\mathbf{C}$ with non-zero entries indicate the components of the co-basis $\mathbf{H}_2$ and components of the basis $\mathbf{H}_1$, respectively.

Therefore, the minimum form of $\mathbf{C}$ is defined as follows:

Definition 5 (*Minimum form*) The minimum form of $\mathbf{C}$ is the minimum number of non-zero entries in $\mathbf{C}$, which cannot be further decreased.

To minimize the form of $\mathbf{C}$, we must derive $\mathbf{C}$ efficiently. The QR decomposition technique can yield a numerically stable solution [34] of linear equations

with minimum least square error [35]. As a result, we explore a numerically robust decomposition method using the QR decomposition technique on $\mathbf{H}_{[(M-1)\times L]}$ to obtain $\mathbf{HE} = \mathbf{UR}$, where $\mathbf{E}_{[L\times L]}$ is the column permutation matrix. The orthogonal matrix is $\mathbf{U}_{[(M-1)\times(M-1)]}$. The upper triangular matrix is $\mathbf{R}_{[(M-1)\times L]}$. $\mathbf{HE}$ denotes a permutation of columns in $\mathbf{H}$. Then, $\mathbf{R}$ is broken into

$$\mathbf{R} = \begin{bmatrix} \mathbf{\Psi 1}_{[\mathbf{r}\times\mathbf{r}]} & \mathbf{\Psi 2}_{[r\times(L-r)]} \\ \mathbf{0}_{[(M-1-r)\times r]} & \mathbf{0}_{[(M-1-r)\times(L-r)]} \end{bmatrix}. \tag{46}$$

The rank of $\mathbf{\Psi}_1$ equals the rank of $\mathbf{H}$, named as r. And then, we introduce the following proposition to derive $\mathbf{C}$.

Proposition *A column-dependence matrix* ($\mathbf{C}$) *is evaluated with a numerical expression, as follows:*

$$\mathbf{C} = \mathbf{\Psi}_1^{-1}\mathbf{\Psi}_2. \tag{47}$$

Proof Due to the results gathered from QR decomposition, $\mathbf{HE} = \mathbf{UR} = \mathbf{U_r}[\mathbf{\Psi_1}\mathbf{\Psi_2}]$ holds. $\mathbf{U}_r$ is constituted by the first r columns of $\mathbf{U}$. Note that r is the rank of $\mathbf{H}$. Additionally, according to (45), $\mathbf{HE}$ is partitioned as

$$\mathbf{HE} = [\mathbf{H}_1'\mathbf{H}_2'], \tag{48}$$

where $\mathbf{H}_1'$ and $\mathbf{H}_2'$ each have a similar definition in (45). Substituting (48) into $\mathbf{HE} = \mathbf{U_r}[\mathbf{\Psi_1}\mathbf{\Psi_2}]$, hence, $\mathbf{H}_1' = \mathbf{U_r}\mathbf{\Psi_1}$ and $\mathbf{H}_2' = \mathbf{U_r}\mathbf{\Psi_2}$ hold. Because $\mathbf{H}_2' = \mathbf{H}_1'\mathbf{C}$ holds, the equivalent equation is expressed as

$$\mathbf{U_r}\mathbf{\Psi_2} = \mathbf{U_r}\mathbf{\Psi_1}\mathbf{C}. \tag{49}$$

Because the columns within the orthogonal matrix $\mathbf{U}_r$ are full, $\mathbf{U_r^T}\mathbf{U_r} = I$ holds. Multiplying both sides of (49) by $\mathbf{U}_r^{\mathrm{T}}$ produces $\mathbf{\Psi_2} = \mathbf{\Psi_1}\mathbf{C}$. This result yields $\mathbf{C} = \mathbf{\Psi_1^{-1}}\mathbf{\Psi_2}$. □

A single QR decomposition, however, cannot guarantee $\mathbf{C}$ in minimum form. To minimize the number of non-zero entries in $\mathbf{C}$, we need to swap a component in the basis with a component in the co-basis.

Denote a sub-matrix of $\mathbf{C}$:

$$\mathbf{C}_{\mathrm{sub}} = \begin{bmatrix} c_{ik} & c_{im} \\ c_{jk} & c_{jm} \end{bmatrix}. \tag{50}$$

Assuming c_{jk} is a non-zero entry, the jth component in the basis is swapped with the kth component in the co-basis. The updated entries from the kth column of $\mathbf{C}$ then becomes:

$$\mathbf{C}_k = -(1/c_{jk}) \cdot \left[c_{1k} \cdots c_{j-1\,k} \; -1 \; c_{j+1\,k} \cdots c_{rk} \right]^{\mathrm{T}}. \tag{51}$$

Additionally, other columns of $\mathbf{C}$ are expressed as

$$\mathbf{C}_n = \left[c_{1n} - \frac{c_{jn}c_{1k}}{c_{jk}} \cdots \frac{c_{jn}}{c_{jk}} \cdots c_{rn} - \frac{c_{jn}c_{rk}}{c_{jk}} \right]^{\mathrm{T}} \tag{52}$$

where $n = 1, 2, \ldots L - r, n \neq k$. All zero entries in the kth column of updated $\mathbf{C}$ after swapping remain zero as the same in the original $\mathbf{C}$. Additionally, the non-zero entry c_{im} in $\mathbf{C}_{\text{sub}}$ becomes zero after swapping. Therefore, we provide Lemma 4 as a sufficient condition for $\mathbf{C}$ in minimum form.

Lemma 4 *If any two columns of* $\mathbf{C}$ *have concurrent non-zero entries in, at most, one common row, then* $\mathbf{C}$ *achieves the minimum form.*

Proof By contradiction, suppose this Lemma is false, i.e., any two columns of $\mathbf{C}$ have concurrent non-zero entries in, at most, one common row, but $\mathbf{C}$ is not in minimum form. This means that $\mathbf{C}$ can be further decreased the number of non-zero entries. We arbitrarily choose a general sub-matrix $\mathbf{C}_{\text{sub}}$ from $\mathbf{C}$ in (50). Assuming c_{ik}=0, $c_{im} \neq 0$, $c_{jk} \neq 0$, and $c_{jm} \neq 0$ hold, $\mathbf{C}_{\text{sub}}$ satisfies two columns of concurrent non-zero entries in, at most, one common row. Thus, we further decrease the number of non-zero entries in $\mathbf{C}_{\text{sub}}$ and obtain the updated sub-matrix $\mathbf{C}_{\text{sub}_{\text{update}}}$ via (51) and (52), which is expressed as

$$\mathbf{C}_{\text{sub}_{\text{update}}} = \begin{bmatrix} -\frac{c_{ik}}{c_{jk}} & c_{im} - \frac{c_{jm}c_{ik}}{c_{jk}} \\ \frac{1}{c_{jk}} & \frac{c_{jm}}{c_{jk}} \end{bmatrix} = \begin{bmatrix} 0 & c_{im} \\ \frac{1}{c_{jk}} & \frac{c_{jm}}{c_{jk}} \end{bmatrix}. \tag{53}$$

$\mathbf{C}_{\text{sub}_{\text{update}}}$ cannot be decreased the number of non-zero entries, contradicting our assumption. Thus, this lemma holds. □

Then, we provide the Lemma 5 for $\mathbf{C}$, by which can be decreased the number of non-zero entries.

Lemma 5 *The number of non-zero entries in* $\mathbf{C}$ *can be decreased only if* $\mathbf{C}$ *has a sub-matrix* $\mathbf{C}_{sub}$ *with both all non-zero entries and* $\det(\mathbf{C}_{sub}) = 0$ *holding.*

Proof Assume the number of non-zero entries in a general sub-matrix ($\mathbf{C}_{\text{sub}}$) from $\mathbf{C}$ in (50) can be decreased. According to Lemma 4, all entries in $\mathbf{C}_{\text{sub}}$ must be non-zero. Based on (51) and (52), the updated sub-matrix $\mathbf{C}_{\text{sub}_{\text{updated}}}$, after swapping the jth component of the basis with the kth component of the co-basis, is expressed as

$$\mathbf{C}_{\text{sub}_{\text{updated}}} = \begin{bmatrix} -\frac{c_{ik}}{c_{jk}} & c_{im} - \frac{c_{jm}c_{ik}}{c_{jk}} \\ \frac{1}{c_{jk}} & \frac{c_{jm}}{c_{jk}} \end{bmatrix}. \tag{54}$$

Because all entries in $\mathbf{C}_{\text{sub}}$ are non-zero, decreasing the number of non-zero entries of $\mathbf{C}_{\text{sub}}$ requires that $c_{im} - \frac{c_{jm}c_{ik}}{c_{jk}} = 0$. Thus, $\det(\mathbf{C}_{sub}) = c_{im}c_{jk} - c_{jm}c_{ik} = 0$ holds. □

Note that the computational complexity of searching the non-zero 2-by-2 sub-matrices of $\mathbf{C}$ can be limited, because $\mathbf{C}$ is the sparse matrix. By removing the rows

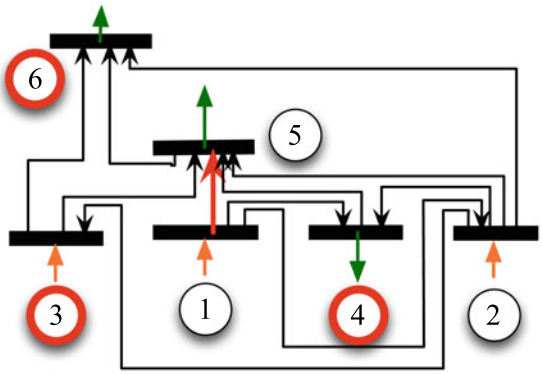


Fig. 2 A six-bus test power system

with all zero entries and searching the intersection sets of non-zero column index among any two rows with non-zero entries, the non-zero 2-by-2 sub-matrices of $\mathbf{C}$ can be found efficiently. In summary, the minimum form of $\mathbf{C}$ can be achieved from line 7 to 19 in the LIS Algorithm 2 (Fig. 2).

Step 3: ***Calculate possible solutions of line outage locations***

Based on Lemma 3, the ambiguity group is constituted by the selection linearly dependent columns of $\mathbf{X}_{ML}$, which are obtained by $\mathbf{C}$. Hence, one or more zero entries in one column of $\mathbf{C}$ form a possible solution, which corresponds to the line outage locations. More specifically, both this column and all rows with non-zero entries correspond to locations of line outage as indicated by the component of the co-basis of $\mathbf{H}_2'$ and components of the basis $\mathbf{H}_1'$, respectively.

Summary, the possible solution is constituted of two parts: (1) the component in the co-basis, which corresponds to one column from $\mathbf{C}$ with zero entries and (2) the components of the basis, which correspond to rows of $\mathbf{C}$ with non-zero entries.

Step 4: ***Select the best solution***

Due to the limited PMUs measurements, $\mathbf{\Delta\Theta}_M$ is known. We calculate the changes in the phasor angle $\mathbf{\Delta\Theta}_{M_{\text{cal}_i}}$ from the obtained potential solutions s_i $(i = 1, 2, \ldots, w)$. To do so, we use the following equation as the rule to select the best solution s^*

$$s^* = \arg \min_{i=1,2,\ldots,w} \|(\mathbf{\Delta\Theta}_{M_{\text{cal}_i}} - \mathbf{\Delta\Theta}_M)\|_2, \tag{55}$$

We can determine the most likely line outage locations when (55) is satisfied.

3.2.3 Generalization of the LIS Algorithm

This section generalizes the LIS algorithm combined with the loading variation vector in a practical scenario.

To analyze the effect of loading variation ($\boldsymbol{\epsilon}$) in the post-outage system, we define $\mathbf{p}''$ to describe the post-outage power flow combined with the loading variation effect, which is shown as follows:

$$\begin{aligned}\mathbf{p}''_{[N\times 1]} &= \mathbf{p}_{0_{[N\times 1]}} + \boldsymbol{\epsilon}_{[N\times 1]} \\ &= \mathbf{B}'_{[N\times N]} \cdot \boldsymbol{\Theta}''_{[N\times 1]},\end{aligned} \tag{36}$$

where $\mathbf{p}''$ is the post-outage power values with loading variation. The susceptance matrix ($\mathbf{B}'$) in a post-outage system combined with loading variation, is included. Note that $\mathbf{B}'$ is the same in a post-outage system even when loading variation exists. We define $\boldsymbol{\Theta}''$ as the phasor angle data combined with the loading variation in a post-outage system. The loading variation is represented by $\boldsymbol{\epsilon}$. In this study, Denote the $\boldsymbol{\epsilon}$ as the vector of zero-mean, with a covariance matrix $\sigma_\epsilon^2\mathbf{I}$ [21].

By applying the LIS algorithm 3.2, we can generalize the outage identification equation with loading variation included. This subsection derives the generalized outage identification equation based on the identical assumptions in Sect. 3.2 and Sect. 3.2.2.

On account of the loading variation in a post-outage system, we substitute (4) into (36) to derive $\boldsymbol{\Theta}''$ as follows:

$$\boldsymbol{\Theta}'' = \boldsymbol{\Theta}' + \mathbf{B}'^{-1} \cdot \boldsymbol{\epsilon}, \tag{37}$$

where $\boldsymbol{\epsilon}$ is the vector of loading variation in a post-outage system.

Using both (32) and (37), we can determine the difference between $\boldsymbol{\Theta}_0$ in a pre-outage system and $\boldsymbol{\Theta}''$ in a post-outage system with loading variation, as defined by $\boldsymbol{\Delta\Theta}'$:

$$\boldsymbol{\Delta\Theta}' = \boldsymbol{\Theta}'' - \boldsymbol{\Theta}_0 = \boldsymbol{\Delta\Theta} + \mathbf{B}'^{-1} \cdot \boldsymbol{\epsilon}. \tag{38}$$

By substituting (31) and (33) into (38):

$$\begin{aligned}\boldsymbol{\Delta\Theta}' = &- \mathbf{B}_0^{-1}\mathbf{Q}_{NF}\left(\mathbf{Z}_\delta^{-1} + \mathbf{Q}_{NF}^{\mathrm{T}}\mathbf{B}_0^{-1}\mathbf{Q}_{NF}\right)^{-1} \\ &\cdot \mathbf{Q}_{NF}^{\mathrm{T}}(\boldsymbol{\Theta}_0 + \mathbf{B}_0^{-1}\boldsymbol{\epsilon}) + \mathbf{B}_0^{-1}\boldsymbol{\epsilon}.\end{aligned} \tag{39}$$

By applying (34), (39) is thus described as:

$$\boldsymbol{\Delta\Theta}'_{[N\times 1]} = \mathbf{X}_{NF_{[N\times F]}}\boldsymbol{\beta}_{\boldsymbol{\epsilon}_{[F\times 1]}} + \mathbf{B}^{-1}_{0_{[N\times N]}}\boldsymbol{\epsilon}_{[N\times 1]}, \tag{40}$$

where $\boldsymbol{\beta}_{\boldsymbol{\epsilon}}$ is defined as

$$\boldsymbol{\beta}_{\boldsymbol{\epsilon}_{[F\times 1]}} = \boldsymbol{\alpha}_{[F\times N]}\left(\boldsymbol{\Theta}_{0_{[N\times 1]}} + \mathbf{B}^{-1}_{0_{[N\times N]}}\boldsymbol{\epsilon}_{[N\times 1]}\right), \tag{41}$$

in which

$$\boldsymbol{\alpha}_{[F\times N]} = \left(\mathbf{Z}^{-1}_{\delta_{[F\times F]}} + \mathbf{Q}^{\mathrm{T}}_{NF_{[N\times F]}}\mathbf{B}^{-1}_{0_{[N\times N]}}\mathbf{Q}_{NF_{[N\times F]}}\right)^{-1}\mathbf{Q}^{\mathrm{T}}_{NF_{[N\times F]}}. \tag{42}$$

Given the limited PMUs measurements (M), we use the same selection matrix $\mathbf{K}_{[M\times N]}$ to left multiply both sides of (40). We then obtain $\boldsymbol{\Delta\Theta}'_M$. This is defined as the *generalized outage identification equation*:

$$\boldsymbol{\Delta\Theta}'_{M_{[M\times 1]}} = \mathbf{X}_{\boldsymbol{\epsilon}_{MF_{[M\times F]}}}\boldsymbol{\beta}_{\boldsymbol{\epsilon}_{[F\times 1]}} + \mathbf{K}_{[M\times N]}\left(\mathbf{B}^{-1}_{0_{[N\times N]}}\boldsymbol{\epsilon}_{[N\times 1]}\right). \tag{43}$$

According to Sect. 3.2.2, the generalized outage identification equation can again be solved with the LIS algorithm.

3.3 Optimal Strategy of PMUs Placement Algorithm Design

Based on the previous anomaly behavior of multiple line outages analysis, the success of location identification for multiple-line outages, however, has been hindered by limited PMU measurements. The difficulty is to mathematically describe the identification performance for multiple-line outages. Therefore, in this subsection, we presents an average identification capability (AIC) modeling and analysis framework for multiple line outages to facilitate the proposed PMUs placement optimization.

3.3.1 Overview of Optimal PMUs Placement for Identifying Multiple Line Outages

As illustrated in Fig. 3, the framework of finding optimal PMUs placement consists of three components. Firstly, applying statistic analysis of transmission line outages and power demand, we build the mathematical model to capture the average identification capability of multiple line outages. Secondly, given the budget of PMUs, we develop the optimization solver to maximize the average identification capability. Finally, we obtain the optimal selection of PMU locations under the budget of PMUs.

Three assumptions are made when we explore the optimal PMU placement. First, fast system dynamics are considered well damped. It thus brings the system into a quasi-static state when line outage occurs. In this quasi-stable system, any fast oscillation in the phasor angle data can be filtered out with a low-pass filter [13]. Second, no islanding scenario exists after line outage occurs [6, 8].

According to the problem formulation 2, the vector $\boldsymbol{\Theta} \in \mathbb{R}^N$ is denoted as all phase angle data, where N represents the total number of buses. Define the number of selected PMU locations by M ($M \leq N$). We assume that transmission line can be in one of two states: $\mathcal{T}_1$ outage-occurred and $\mathcal{T}_0$ outage-free. Mathematically, selecting M out of N buses can be represented by a linear map $\mathbb{R}^N \to \mathbb{R}^M$, $\boldsymbol{\Theta} \longmapsto \mathbf{y} = \mathbf{K}^{\mathbf{T}}\boldsymbol{\Theta}$, in which the selection matrix $\mathbf{K} \in \mathbb{R}^{N\times M}$ is the rank M matrix that has exactly one

unit entry per column, corresponding to a selected PMU location, and the other entries in columns being zero. Note that $\mathbf{K^T K} = \mathbf{I}_M$, in which $\mathbf{I}_M$ is the M-by-M identity matrix. Under the scenario $\mathcal{T}_i$, $i = \{0, 1\}$, $\mathbf{y}$ is a linear transformation of $\mathbf{\Theta}$. Thus, $\mathbf{y}$ also follows Gaussian distribution:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{K}^{\mathrm{T}}\mu_i, \mathbf{K}^{\mathrm{T}}\Sigma_i\mathbf{K}\right), \text{under } \mathcal{T}_i, \quad i = 0, 1. \tag{44}$$

Let define the probability density of $\mathcal{N}\left(\mathbf{K}^{\mathrm{T}}\mu_i, \mathbf{K}^{\mathrm{T}}\Sigma_i\mathbf{K}\right)$ is defined as $f_i(\mathbf{y}; \mathbf{K})$ ($i = \{0, 1\}$).

Let define the scenarios of line outages $\ell \in \{1, \ldots, |L|\}$, where $|L| = \sum_{f=1}^{F} \binom{N}{f}$ presents the total possible scenarios of line outages. F is the total number of simultaneous line outages. Because different line outages result in different susceptance matrix $\mathbf{B}$, each $f_1^{(\ell)}(\mathbf{y}; \mathbf{K})$ represents the probability density of PMU measurement corresponding to ℓ scenario of line outage.

Inspired by Kullback-Leibler (KL) distance [36, 37], the average identification capability (AIC) is defined as the average dissimilarity distance between $f_1^{(\ell)}$ and f_0 ($\ell = 1, 2, \ldots, |L|$), which is shown as follows:

$$\mathrm{AIC}(\mathbf{K}) = \sum_{\ell=1}^{|L|} \alpha_\ell d^{(\ell)}(\mathbf{K}), \tag{45}$$

where the function $d^{(\ell)}(\mathbf{K})$ is described as

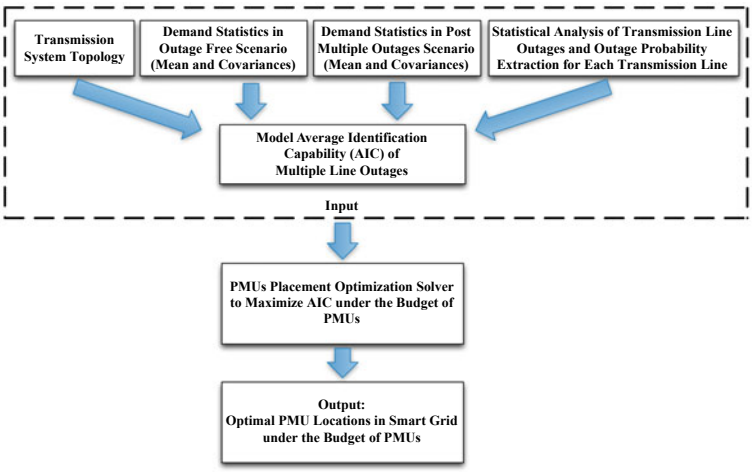


Fig. 3 Framework of finding optimal PMUs placement for identifying multiple line outages

$$
\begin{aligned}
d^{(\ell)}(\mathbf{K}) :&= d^{(\ell)}\left(\mathcal{N}\left(\mathbf{K}^{\mathrm{T}}\mu_1^{(\ell)}, \mathbf{K}^{\mathrm{T}}\Sigma_1^{(\ell)}\mathbf{K}\right) || \mathcal{N}\left(\mathbf{K}^{\mathrm{T}}\mu_0, \mathbf{K}^{\mathrm{T}}\Sigma_0\mathbf{K}\right)\right) \\
&= \frac{1}{2}\Big\{(\mu_{\mathbf{1}}^{(\ell)} - \mu_{\mathbf{0}})^{\mathbf{T}}\mathbf{K}\left(\mathbf{K}^{\mathbf{T}}\Sigma_{\mathbf{0}}\mathbf{K}\right)^{-\mathbf{1}}\mathbf{K}^{\mathbf{T}}(\mu_{\mathbf{1}}^{(\ell)} - \mu_{\mathbf{0}}) \\
&\quad + \mathrm{tr}\left(\left(\mathbf{K}^{\mathbf{T}}\Sigma_{\mathbf{0}}\mathbf{K}\right)^{-\mathbf{1}}\mathbf{K}^{\mathbf{T}}\Sigma_{\mathbf{1}}^{(\ell)}\mathbf{K}\right) \\
&\quad - \log(|\mathbf{K}^{\mathbf{T}}\Sigma_{\mathbf{0}}\mathbf{S}|)^{-1}|(\mathbf{K}^{\mathbf{T}}\Sigma_{\mathbf{1}}^{(\ell)}\mathbf{K})| - M\Big\},
\end{aligned} \tag{46}
$$

where α_ℓ represents the probability of ℓ line outage; $\mathrm{tr}(\cdot)$ denotes trace operation; $|\cdot|$ denotes determinant operation; M is the dimension, which represents the number of PMU measurements.

Therefore, Given the budget of PMUs, our objective function is to maximize the average identification capability of multiple line outages, which can be mathematically formulated as an optimization problem in Problem 3.

Problem 3 Given the budget $C \in \mathbb{R}$ and the cost c_{ij} for each PMU ($i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, M$), the problem of finding the optimal PMU locations is formulated as:

$$
\begin{aligned}
&\text{maximize } \mathrm{AIC}(\mathbf{K}) \\
&\text{subject to } k_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, N; \ j = 1, 2, \ldots, M \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}c_{ij} \le C, \\
&\qquad \sum_{i=1}^{N} k_{ij} = 1,
\end{aligned} \tag{47}
$$

where k_{ij} is the element of matrix $\mathbf{K}$.

To prove Problem 3 to be an NP hard problem, we assume that each PMU has the same cost c, for the sake of simplicity. Thus, the constraint $\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}c_{ij} \le C$ in Problem 3 converts to $M \le \lfloor \frac{C}{c} \rfloor$. Considering M to be the upper bound $M = \lfloor \frac{C}{c} \rfloor$, the Problem 3 can be casted as follows:

Problem 4 Given the number of PMU measurements $M = \lfloor \frac{C}{c} \rfloor$ ($M \le N$), in which $C \in \mathbb{R}$ is the system budget and c is the PMU cost, the problem of finding the optimal PMU locations is formulated as:

$$
\begin{aligned}
&\text{maximize } \mathrm{AIC}(\mathbf{K}) \\
&\text{subject to } k_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, N; \ j = 1, 2, \ldots, M \\
&\qquad \sum_{i=1}^{N} k_{ij} = 1.
\end{aligned} \tag{48}
$$

Optimization Problem 4 is combinatorial. We will prove that the Problem 4 is an NP hard problem, which is shown as follows.

Theorem 1 *Optimization Problem 4 is the NP hard problem.*

Proof Inspired by maximal clique problem (MAX-CLIQUE), which is known to be NP hard [38], we plan to reduce the Problem (4) to be a MAX-CLIQUE problem. First

of all, we define the MAX-CLIQUE problem as follows. **MAX-CLIQUE** [38]: The graph $\mathcal{G} = (\mathcal{N}, \Xi)$ is an undirected graph, where $\mathcal{N} = \{1, 2, \ldots, N\}$ is the vertex set of $\mathcal{G}$ with cardinality $|\mathcal{N}| = N$, and $\Xi \subseteq \mathcal{N} \times \mathcal{N}$ is the set of undirected edges of $\mathcal{G}$. The MAX-CLIQUE is the problem for finding the maximum size clique in $\mathcal{G}$, where a clique in $\mathcal{G}$ is a subset of vertices ($\mathcal{C} \subseteq \mathcal{N}$), such that all pairs of vertices in $\mathcal{C}$ are connected by an edge, i.e., $\forall v, v^{'} \in \mathcal{C}, \{v, v^{'}\} \in \Xi$ and the size of a clique $\mathcal{C}$ is the number of vertices in $\mathcal{C}$.

For the sake of simplicity, we adopt a special structure $N \times N$ matrix $\mathbf{V}(\mathcal{G})$, which is shown in Eq. (49), to reduce the MAX-CLIQUE problem.

$$\mathbf{V}(\mathcal{G}) = \begin{cases} 2N & \text{if } i = j \\ -1 & \text{if } i \neq j,\ i,\ j \in \Xi \\ 0 & \text{otherwise.} \end{cases} \tag{49}$$

Because the matrix $\mathbf{V}(\mathcal{G})$ has positive diagonal elements and is strictly diagonally dominant, $\mathbf{V}(\mathcal{G})$ is the positive definite matrix. Given a maximum size M ($M \leq N$) of clique in $\mathcal{G}$, a set of matrices are represented as $\mathbf{W}_M = \{\mathbf{W} \in \mathbb{R}^{M \times M} : \mathbf{W} = \mathbf{W^T}\}$:

$$W_{ij} = \begin{cases} 2N, & i = j \\ \in \{0, -1\}, & i \neq j, \end{cases} \tag{50}$$

where $i, j = \{1, 2, \ldots, M\}$. Applying the same theory, all matrices in $\mathbf{W}_M$ are positive definite. Moreover, let denote $\mathbf{1}_M$ as the column vector with all entries equal to 1. We also define the function $v(\mathbf{W}) = \mathbf{1_M^T W^{-1} 1_M}$ follows features as:

Lemma 6 *For all matrices* $\mathbf{S} \in \mathbf{S_M}$*, the following claims holds.*

1. *Let define* $\mathbf{A} = \mathbf{W^{-1}}$*. Then,* $\forall i, j,\ A_{ij} \geq 0$ *holds.*
2. *if* $W_{ij} = 0$*, then* $v(\mathbf{W} - g_i g_j^{\mathrm{T}} - g_j g_i^{\mathrm{T}}) \geq v(\mathbf{W})$*, where* g_i *denotes* i*th canonical vector.*
3. $v(\mathbf{W}) \leq \frac{M}{2N-M+1}$*, where the equality holds if and only if* $\mathbf{W} = \mathbf{W}^* = 2N\mathbf{I} - \mathbf{1}_M \mathbf{1}_M^{\mathrm{T}} + \mathbf{I}$*. Note that* $\mathbf{I}$ *is the* M*-by-*M *identity matrix.*

□

Proof For claim (1), based on the literature [39], the matrix $\mathbf{A} = \mathbf{W^{-1}} \geq \mathbf{0}$ holds for all entries. For claim (2), the function v is convex and differentiable on the set of positive definite variables [40]. Therefore, applying the first-order Taylor expansion at $\mathbf{S}$, the equation $v(\mathbf{W} - g_i g_j^{\mathrm{T}} - g_j g_i^{\mathrm{T}}) \geq v(\mathbf{W})$ holds. For claim (3), we firstly prove the statement: "if $v(\mathbf{W}) \leq \frac{M}{2N-M+1}$ holds, then $\mathbf{W} = \mathbf{W}^*$ equals $2N\mathbf{I} - \mathbf{1}_M \mathbf{1}_M^{\mathrm{T}} + \mathbf{I}$." Based on the condition of this statement, $v(\mathbf{W}^*) = \frac{M}{2N-M+1}$ holds. According to the definition of function $v(\cdot)$, we obtain that W^* equals $2N\mathbf{I} - \mathbf{1}_M \mathbf{1}_M^{\mathrm{T}} + \mathbf{I}$. Secondly, let's prove the statement: "if $\mathbf{W} = \mathbf{W}^*$ equals $2N\mathbf{I} - \mathbf{1}_M \mathbf{1}_M^{\mathrm{T}} + \mathbf{I}$, then inequality $v(\mathbf{W}) \leq \frac{M}{2N-M+1}$ holds." For the sake of description, denote the $\mathbf{W}^*$ as a maximum of $v(\mathbf{W})$ over the set $\mathbf{W}_M$. According to claim (2), it shows that when the more negative

variables are in the function $v(\cdot)$, the value of $v(\mathbf{W})$ becomes higher. Therefore, it suffices to illustrate that, $\forall i, j$ $(1 \leq i, j \leq M)$, $v(\mathbf{W}^*) = \frac{M}{2N-P+1}$. The claim (3) holds.

According to the this Lemma, we reduce the Problem 4 to MAX-CLIQUE. Given the graph $\mathcal{G}$ and the positive integer M $(1 \leq M \leq N)$, we will find whether $\mathcal{G}$ has a clique of maximal size M. Assume that the matrix $\mathbf{V}(\mathcal{G})$ satisfies the conditions in (49). Consider the objective function AIC in Problem 4 as an instance with the following data: (1) fixed clique size M; (2) $\mu_{\mathbf{1}} = \mathbf{1_N}, \mu_{\mathbf{0}} = \mathbf{0_N}$; (3) $\Sigma_{\mathbf{1}} = \Sigma_{\mathbf{0}} = Z(\mathcal{G})$; (4) AIC $= \frac{1}{2}\frac{M}{2N-M+1}$. Therefore, the Problem 4 becomes the MAX-CLIQUE if and only if the graph $\mathcal{G}$ has a clique of size at M.

Hence, the Problem 4 is an NP hard problem. □

To solve this NP hard problem, we introduce two optimal PMUs placement algorithms to solve the Problem 3. One is the exhaustive search optimal method (ESOM); the other is the greedy heuristic optimal method (GHOM).

3.3.2 Exhaustive Search Optimal Method (ESOM)

To simplify the description of ESOM, we assume that the PMU has the same cost c. Thus, the Problem 3 transforms into Problem 4. Given the numbers of PMUs (M), we denote $|J| = \binom{N}{M}$ as the total number of possible combinations of selected PMUs. The total solution space of Problem 4 is denoted as $\Omega = \{\boldsymbol{\omega_1}, \boldsymbol{\omega_2}, \ldots, \boldsymbol{\omega}_{|\mathbf{J}|}\}$. Note that the vector N-by-1 $\boldsymbol{\omega}_q$ is shown as follows:

$$\boldsymbol{\omega}_q = \{I_i\}_{i=1}^{N}, \quad \{I_i\} \in \{0, 1\}, \text{ for } q = 1, 2, \ldots, |J|, \tag{51}$$

where the indicator function I_i indicates that the ith location is installed PMU. Based on the definition of a selection matrix $\mathbf{K}$ of phasor measurement data, which has one unit entry per column, corresponding to a selected PMU location, and the other entries in columns being zero, a selection matrix $\mathbf{K}$ is constructed by a $\boldsymbol{\omega}_q$. Thus, the basic idea of ESOM is to search $\boldsymbol{\omega}_q$ through the whole solution space Ω to find the maximal average dissimilarity distance AIC($\mathbf{K}$) under the given number of PMU measurements $(M = \lfloor \frac{C}{c} \rfloor)$.

When N and M are small, an ESOM can efficiently find the global optimal solution for the Problem 3. However, as N grows up, the Algorithm 3 becomes computationally infeasible. To reduce the computational complexity, we develop the sub-optimal algorithm, called greedy heuristic optimal method (GHOM), to address the Problem 3 with acceptable computational complexity.

3.3.3 Greedy Heuristic Optimal Method (GHOM)

The GHOM is adopted to find the sub-optimal PMUs placement for identifying multiple line outages, as illustrated in Algorithm 4. For the sake of simplicity, we

Algorithm 3 (ESOM): Input: μ_0, μ_1, Σ_0, Σ_1, and initial ω_1. Output: $\mathbf{K}$

1: **Initialization** ω_1;
2: Calculate $\mathbf{K}^{(1)}$ using ω_1;
3: Calculate $\text{AIC}_{\max} \leftarrow \text{AIC}^{(1)}$;
4: **for** $q = 2, 3, \ldots, |J| = \binom{N}{M}$ **do**
5: Calculate $\mathbf{K}^{(q)}$ using ω_q;
6: Calculate $\text{AIC}^{(q)}$ by $\mathbf{K}^{(q)}$;
7: **if** $\text{AIC}^{(q)} \geq \text{AIC}_{\max}$
8: $\text{AIC}_{\max} \leftarrow \text{AIC}^{(q)}$
9: $\mathbf{K} \leftarrow \mathbf{K}^{(q)}$
10: **end if**
11: **end for**
12: **return**

Algorithm 4 (GHOM): Input: μ_0, μ_1, Σ_0, Σ_1, and initial selected PMUs set t_p^{init} (size of t_p^{init} equals $N_{\text{P}}^{\text{init}}$). Output: $t_p = t_p \cup \{n_\rho\}_{\rho=1}^{M-N_{\text{P}}^{\text{init}}} \rightarrow \mathbf{K}$

1: **Initialization** the selected PMUs set $t_p \leftarrow t_p^{\text{init}}$;
2: Calculate $\mathbf{K}$ using t_{P};
3: Calculate AIC($\mathbf{K}$);
4: **for** $\rho = 1, 2, \ldots, M - N_{\text{P}}^{\text{init}}$ **do**
5: Find $n_\rho = \arg\max_{n \notin s_p} \text{AIC}(\mathbf{K}(\text{t}_\text{p} \cup \text{n}))$;
6: Update $t_p \leftarrow t_p \cup \{n_\rho\}$.
7: **end for**
8: Obtain the final selection matrix $\mathbf{K}$;
9: **return**

also assume that all PMUs have the same cost c. The Problem 3 is thus reformulated as Problem 4. This GHOM is initialized with a selected subset of bus nodes, defined as t_p^{init}, size of $t_p^{\text{init}} \leq M$. In each iteration, it chooses one more bus node as PMU installation location to achieve the maximum average identification capability in terms of Eq. (45), until the total number of PMUs placement reaches the budget of number of PMUs ($M = \lfloor \frac{C}{c} \rfloor$). According to GHOM algorithm 4, the complexity of each iteration grows linearly with the number of available PMUs, which can be extended easily in large-scale smart grid for identifying multiple line outages. Moreover, this GHOM algorithm is also suitable for implementation future expansion plans of monitoring system when more available PMUs are installed in the monitoring system.

4 Numerical Experiment

This section evaluates the performance and the operational effectiveness of our proposed anomaly detection approach on anomalous events (i.e., transmission line outages) and location identification algorithm of multiple line outages, as well as the PMUs placement algorithms in smart grid.

4.1 Experimental Setup

We first summarize the numerical experimental settings. The software toolbox MATPOWER [41] is used to generate both pertinent AC power flow and PMUs measurements. We utilize IEEE 14-, 30-, 57-, 300-bus systems, and Polish 2383-bus systems as benchmarks. The first four are IEEE benchmark systems [42]. The last is from the Polish Power System provided by MATPOWER "case2383wp" file. We assume that each transmission line has the same outage probability and all PMUs have the same installation cost. For each bus node, the power demand is considered to be random with the mean value at the nominal power level [42], and the standard deviation varies from 0.03 to 0.06. All experiments run on a workstation with four 2.7 GHz Intel processors and 4 GB of memory.

4.2 Real-Time Anomaly Detection Experiments

The proposed solution builds an anomalous behavior model and conducts fast covariance matrix estimation to effectively and fast explore the correlation-based anomaly detection algorithm.

Measured phasor angle data profiles: Applying the AC power flow model and the software toolbox MATPOWER [41], we generate both pertinent AC power flow and PMUs measurements. Specifically, we firstly apply Gaussian distribution with mean value at the nominal power level [42] for each bus node, and standard deviation varies from 0.2 to 0.8 to generate power demand profile. After that, we utilize the MATPOWER toolbox to generate phasor angle data profiles with sample-scale resolution based on AC power flow model. Since the measured phasor angle data is achieved by GPS-based PMUs and regular SCADA, we use parameter time skew level to describe the timing synchronized gap between SCADA and PMUs.

Multiple transmission line outages: We choose 5000 random scenarios with double line outages under measured phasor angle nodes at 30% of the number of buses.

Running time analysis: The following studies are conducted on a workstation with a [2.7] GHz Intel processor and [4] GB of memory. Overall, the proposed anomaly detection flow with built in anomalous score modeling is highly efficient, with a running time of 2.3 s on average over 5000 observation samples.

Design metrics: Considering the effectiveness of the anomaly detection algorithm, two performance design metrics of interest are utilized: detection probability (TP), also called true positive rate, and false alarm probability (FP), also called false positive rate. More specifically,

- *Detection probability* is defined as the probability that the anomalous measured phasor angle samples are detected under specific threshold. In other words, TP determine how many truly abnormal PMU data are picked up. For the sake of the illustration, we define TP as follows:

$$\begin{aligned} \text{TP} &= \Pr\{D_s > t | \text{abnormal event}\} \\ &= \frac{N_{d_{anomaly}}}{N_{\text{anomaly}}}, \end{aligned} \tag{52}$$

where t represents the threshold level. $N_{d_{anomaly}}$ represents the truly detection numbers of anomalous measured phasor angle samples when anomalous score (D_s) is larger than threshold. N_{anomaly} is the total number of anomalous PMUs.

- *False alarm probability* is defined as the percentage of normal measured phasor angle data we misclassified as anomalies. FP is thus mathematically defined as:

$$\begin{aligned} \text{FP} &= \Pr\{D_s > t | \text{normal event}\} \\ &= \frac{N_{\text{mc}}}{N_{\text{normal}}} = \frac{N_d - N_{d_{anomaly}}}{N_{\text{normal}}}, \end{aligned} \tag{53}$$

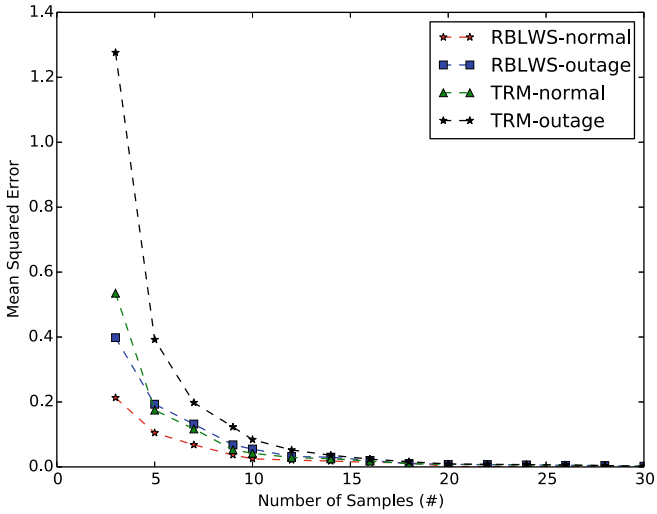
where N_{mc} denotes as the number of normal measured phasor angle data we misclassified as anomalies when anomalous score (D_s) is larger than threshold. N_d is the total detection number of measured phasor angle samples, when $D_s > t$. N_{normal} is the total number of normal PMU data.

In this chapter, we use the receiver operating characteristic (ROC) [43, 44] to visualize the trade-off between the detection probability and the false alarm probability.

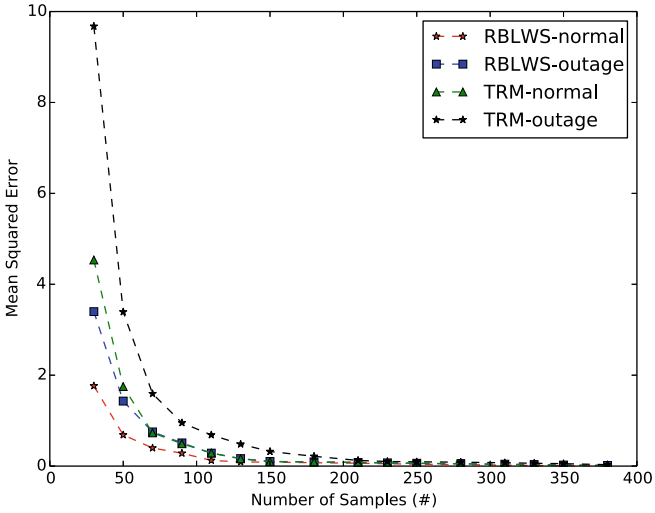
4.2.1 Efficiency of Covariance Estimation Comparison

In this subsection, we adopt the true $\mathbf{\Sigma}_\Theta$, which obtained by 10,000 phasor angle samples, to compare the mean square error (MSE) of two covariance estimators: the traditional method [45] and the Rao-Blackwell LWS method with different shapes of $\hat{\mathbf{\Sigma}}^*$. We randomly choose double line outages in IEEE 30-bus systems and Polish 2383-bus systems. Applying software toolbox MATPOWER [41], we generate 3000 multivariate Gaussian phasor angle samples under outage-free scenario and 3000 multivariate Gaussian PMU samples under double-line outages scenario. For the sake of description, we set up this experiment without time skew. Each simulation is repeated 500 times. The MSE is illustrated as a function of number of samples, which is shown in Fig. 4.

Figure 4 demonstrates that MSE converges at different number of measured phasor angle samples in different IEEE bus systems and Polish system. Meanwhile, the MSE achieved by traditional covariance estimator (TRM) and Rao-Blackwell LWS covariance estimator (RBLWS) are all decreased with the number of measured phasor angle samples increase and eventually converge when the number of measured phasor angle samples is larger enough. It is also observed that MSE achieved by RBLWS converges faster than TRM. Moreover, the MSE curve in double line outages scenario has different from the MSE curve in normal outage-free scenario.



(a) MSE comparison of covariance estimators in IEEE 30-bus system



(b) MSE Comparison of covariance estimators in Polish 2383-bus system

Fig. 4 MSE impacts of **a** number of samples in IEEE 30-bus system and **b** number of samples in Polish 2383-bus system

4.2.2 ROC Performance Analysis

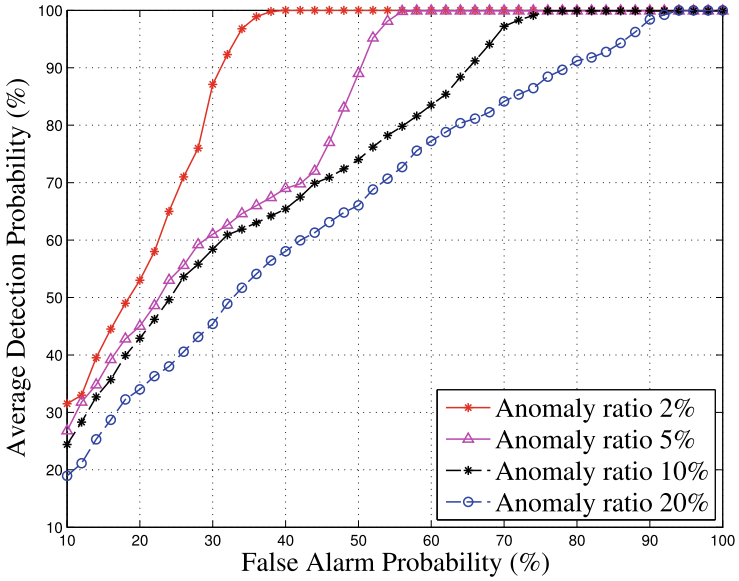
In this subsection, we apply ROC curve to analyze the anomaly detection performance of our proposed method. We randomly choose double line outages in IEEE 30-bus systems and Polish 2383-bus system. While the first two are adopted 10 measured phasor angle samples as the sampling window, the last one is adopted 100 PMU samples as the sampling window. For the sake of description, we set up this experiment without time skew. Using the software toolbox MATPOWER [41], we generate 4000 and 1000 multivariate Gaussian measured phasor angle samples with normal value (such as outage-free scenario) and with abnormal value (such as double line outages scenario), respectively. Let set the anomaly ratio is equal to 2, 5, 10, and 20% when we randomly insert the corresponding numbers of anomalies phasor angle samples into normal phasor angle sample sets. The ROC curves are created by altering the values of false alarm probability, which corresponds to the values of detection threshold. The false alarm probability is from 10 to 100% with a stepping of 10%. Each simulation is repeated 500 times. The numerical results are illustrated in the ROC curves, which is shown in Fig. 5. It illustrates that the average detection performance increases when false alarm probability increases under the same anomaly ratio condition. Furthermore, we observe that the detection performance improves significantly as the anomaly ratio decreases for specific false alarm probability.

4.2.3 Analysis of Anomaly Detection Performance

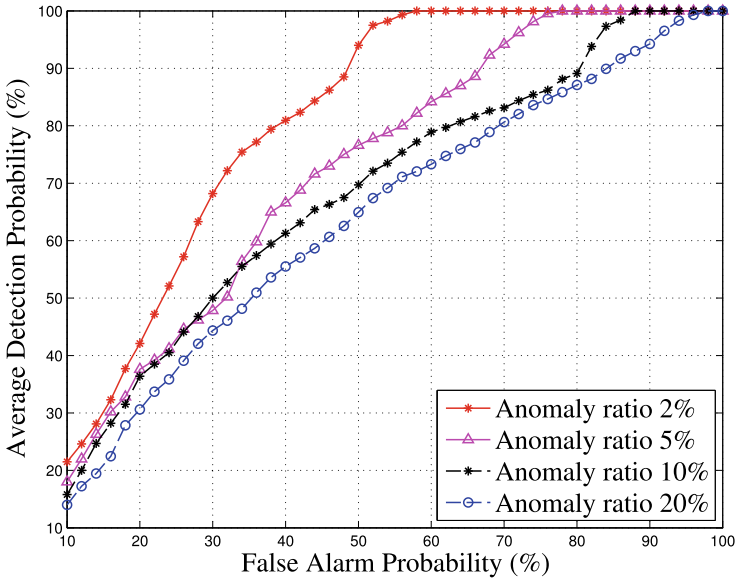
To clarify the analysis of anomaly detection performance, we define the threshold level as follows: applying the ROC curves (see Sect. 4.2.2), we select the detection threshold, which corresponds to the 30% false alarm probability. We use the same experiment conditions in Sect. 4.2.2. Additionally, choosing the 2% anomaly ratio, we generate 10,000 and 220 multivariate Gaussian measured phasor angle samples with normal value (such as outage-free scenario) and with abnormal value (such as double line outages scenario), respectively. After that, we randomly insert the anomalies phasor angle samples into normal samples to obtain the phasor angle sequences for detection. The average anomaly detection probability and running time are shown in Table 1. This table illustrates that different average anomaly detection performance and different running time among three test cases: IEEE 30-bus system and Polish 2383-bus system. Table 1 also shows that the average anomaly detection probability relies heavily on the type of bus system.

4.3 LIS Algorithm Experiments

We implement the proposed LIS algorithm, a sparse signal reconstruction algorithm (called Lassoing algorithm [8]), and an exhaustive search (ES) algorithm in MATLAB to test the proposed LIS algorithm.



(a) Average detection probability versus false alarm probability in IEEE 30-bus system



(b) Average detection probability versus false alarm probability in Polish 2383-bus system

Fig. 5 The ROC curves in **a** IEEE 30-bus system and **b** Polish 2383-bus system

Table 1 The average detection probability and running time

	Average detection probability (%)	Average running time (s)
IEEE 30-bus system	82.6	$8.7e{-}4$
Polish 2383-bus system	61.7	2.07

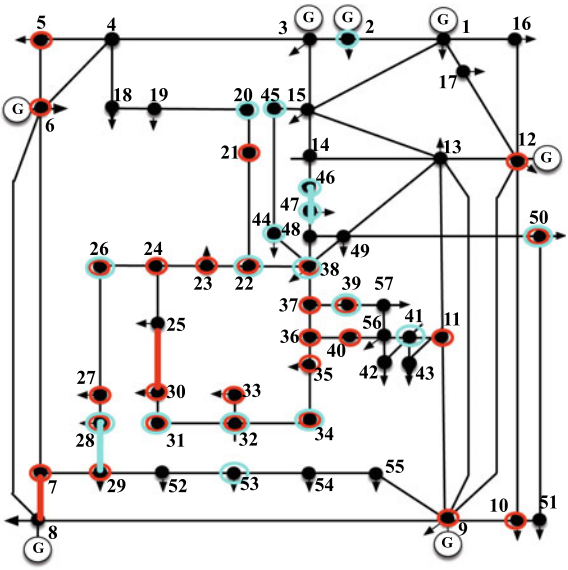


Fig. 6 IEEE 57-bus system. PMUs at 50% of the number of buses: the line outages are from bus 7 to bus 8 and from bus 25 to bus 30. The PMUs are located at buses [5:7, 9:12, 21:24, 26:40, 50]; PMUs at 30% of the number of buses: the line outages are from bus 28 to bus 29 and from bus 46 to bus 47. The PMUs are located at buses [2, 20, 22, 26, 28, 31, 32, 34, 38, 39, 41, 44:47, 50, 53]

4.3.1 Comparison Between Different Methods

Using the IEEE 57-bus test system, as shown in Fig. 6, we set up two different scenarios to demonstrate a performance comparison between two different methods. The first scenario includes PMUs at 50% of the number of buses; the second scenario includes PMUs at 30% of the number of buses. To focus on a performance comparison, we test the IEEE 57-bus system without loading variation.

4.3.2 PMUs at 50% of the Number of Buses

The bold red lines in Fig. 6 represent line outages (i.e., from bus 7 to bus 8 and from bus 25 to bus 30). The red circles represent the PMUs locations. The PMUs are placed randomly.

Compared with the Lassoing algorithm [8] and the ES algorithm, results are in Table 2. The proposed LIS algorithm returns the most likely line outage locations: from bus 7 to bus 8 and from bus 25 to bus 30. Table 2 shows that the LIS algorithm has

Table 2 Comparison between different methods

PMUs: 50% of # buses	Lassoing Alg. [8]	ES	LIS
Line outages	32–33 and 34–35	7–8 and 25–30	7–8 and 25–30
Computational time (s)	0.32	684.28	0.92

Table 3 Comparison between different methods

PMUs: 30% of # buses	Lassoing Alg. [8]	ES	LIS
Line outages	{7–29 and 47–48}	{28–29 and 46–47} {28–29, 46–47, and 21–22}	{28–29, 46–47, and 21–22}
Computational time (s)	0.97	707.13	1.28

the same (and correct) identification results of line outages as does the ES algorithm. Moreover, when compared to the ES algorithm, our proposed LIS algorithm attains a 742× speedup. In contrast, the Lassoing algorithm [8] does not correctly identify multiple line outage locations.

4.3.3 PMUs at 30% of the Number of Buses

We analyze the performance of the proposed LIS algorithm with very limited PMUs measurements. The bold cyan lines in Fig. 6 represent the line outages, such as line from bus 28 to bus 29 and from bus 46 to bus 47. The bold cyan circles represent the PMUs locations. When the LIS algorithm is applied, the most likely locations for line outages are identified as follows: {28–29, 46–47, 21–22}. Table 3 illustrates that the LIS algorithm successfully identifies the most likely line outage locations while attaining a 552× speedup when compared to the method of exhaustive search. In contrast, the Lassoing algorithm [8] does not correctly identify multiple line outage locations with PMUs at 30% of the number of buses.

4.3.4 Sensitivity Analysis

To clarify the impact of PMU coverage and loading variation we define the success rate as

$$\text{Success rate} = \frac{N_{\text{good}}}{N_{\text{total}}}, \tag{54}$$

where N_{good} is the number of successful cases and N_{total} is the total number of simulation cases. We define the successful case by locations of multiple line outages.

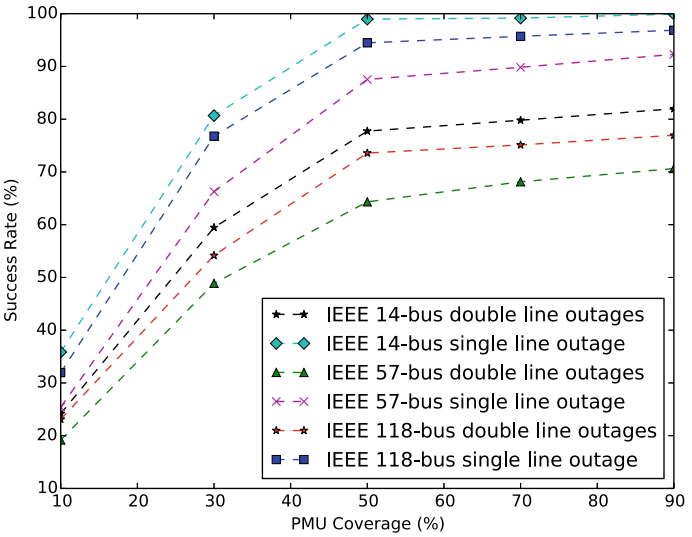


Fig. 7 The success rate in different bus system under different PMU coverage

Specifically, let $\mathcal{S}$ denote the set of actual locations of multiple line outages and $\mathcal{S}_{\text{identify}}$ denote the set of identification locations of multiple line outages. We define $\mathcal{S}_{\text{identify}} \subseteq \mathcal{S}$ as a successful case.

4.3.5 Impact of PMU Coverage

We test the IEEE 14-, 57-, and 118-bus systems and choose 100 random scenarios with both single and double line outages. The PMUs are placed randomly with different coverage of the number of bus systems from 10 to 90%. Figure 7 demonstrates that the success rate increases as the PMUs coverage increases. It shows a threshold effect, which means the success rate doesn't change too much, when PMU coverage is above a certain level. It also shows different bus systems have different success rate even with the same PMU coverage.

4.3.6 Impact of Loading Variation

To illustrate the impact of loading variation, we test the IEEE 118-bus system randomly placing PMUs at 50% of the number of buses. 500 random chosen topologies with both single and double line outages are tested. The standard deviation of loading variation is set either equal to zero, corresponding to a loading variation-free case, or equal to 1, 2, or 5% of the average pre-outage loading condition. The results are listed

in Table 4. This table illustrates that the success rate of our proposed LIS decreases as the loading variation level increases.

4.4 Optimal PMU Placement Experiments

In this section, we implement three PMUs placement methods: ESOM algorithm, GHOM algorithm, and random PMU placement algorithm, in MATLAB to obtain the optimal PMUs placement for identifying multiple line outages.

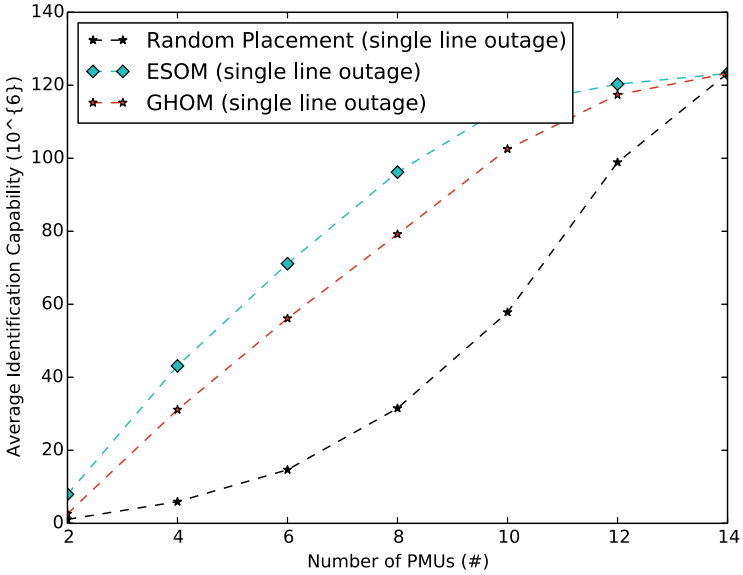
4.4.1 Performance Comparison

Using IEEE 14-bus system [42] as an example, we compare the performance of ESOM algorithm, GHOM algorithm, and random PMU placement method via the average identification capability. The number of required PMU measurements in IEEE 14-bus system is from 2 to 14 with a stepping of 2. These line outages are equal to single and double line outages under the same bus system.

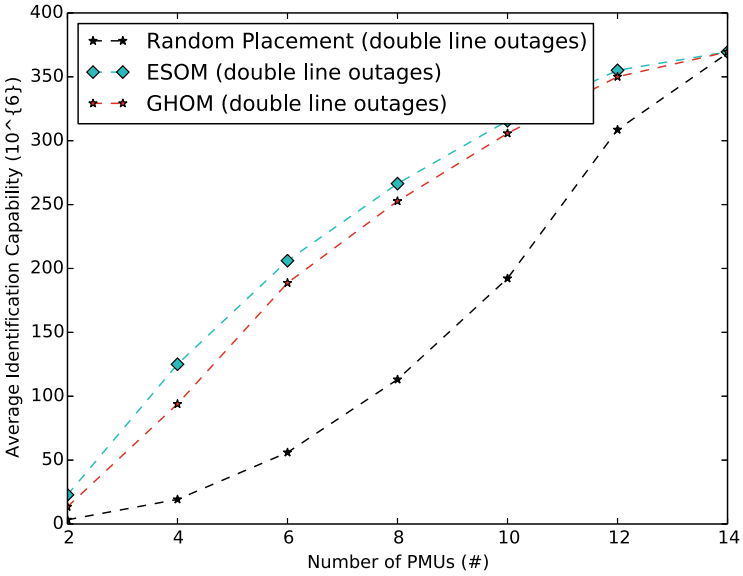
Figure 8 illustrates the average identification capability versus the number of selected PMUs in IEEE 14-bus system for single and double line outage. It shows that the global optimal PMU placement by ESOM has the higher average identification capability when compared to the GHOM algorithm and the random PMU placement algorithm. Meanwhile, the average identification capability achieved by ESOM, GHOM, and random PMU placement method are all increased with the number of selected PMU and eventually converge when all the PMUs are selected. It is also observed that when number of PMUs is small, the performance improvement with one additional PMU is much higher when compared to a larger number of PMUs.

Table 4 Impact of loading variation based on limited PMUs measurement

	Success rate (%)	
Loading variation level	Single	Double
0%	86.67	65.2
1%	68.83	50.45
2%	56.91	46.67
5%	40.51	31.23



(a) single line outage occurred



(b) double line outage occurred

Fig. 8 Average identification capability versus the number of selected PMUs in IEEE 14-bus system when **a** single line outage occurred and **b** double line outages occurred

Table 5 Computational time comparison

	IEEE 14-bus system (s)		IEEE 57-bus system (s)	
	Single	Double	Single	Double
ESOM Alg.	4.68	51.94	N/A	N/A
GHOM Alg.	0.23	1.68	31.27	374.5

Table 6 Average location identification accuracy based on different optimal PMUs placement methods

	Identification accuracy (%)	
	Single	Double
ESOM Alg.	89.97	66.84
GHOM Alg.	84.12	63.23
Random PMUs placement	80.18	57.36

4.4.2 Computational Time Comparison

Using IEEE 14- and 57-bus systems [42], we compare the computational time of ESOM algorithm and GHOM algorithm under both single and double line outages scenarios. The required number of PMUs is at 30% of the number of buses. These line outages are equal to single and double line outages under the same bus system. The computational time of ESOM algorithm and GHOM algorithm is shown in Table 5. This table illustrates that the different optimal PMUs placement methods have different computational time. Table 5 also shows that the computational time of GHOM algorithm attains a 50× speedup when compared to the ESOM algorithm.

4.4.3 Location Identification Accuracy Comparison

Using the IEEE 14-bus systems [42], we compare the location identification accuracy of ESOM algorithm, the GHOM algorithm, and the random PMUs placement method. We randomly choose 500 scenarios with both single and double line. The required number of PMUs is at 30% of the number of buses. The average location identification accuracy of line outages is shown in Table 6. This table illustrates that different PMUs placement methods lead to different location identification accuracy of multiple line outages. Table 6 also shows that the ESOM algorithm has highest location identification accuracy among three PMUs placement methods. Moreover, the proposed ESOM algorithm improves about 10% location identification accuracy of multiple line outages, when compared to random PMUs placement method.

5 Conclusions

This chapter presents an integrated design for real time anomaly detection, location identification of multiple line outages, and the optimal PMUs placement methods to characterize the average identification performance of multiple line outages in large-scale smart grid. The proposed design and optimization flow is driven by an accurate and fast anomalous behavior modeling and analysis solution. It conducts spatial-temporal ReTAD algorithm to efficiently address the issue of terabits phasor measurement data volume and tight real time application. Using the real-time anomaly detection and inspired by the ambiguity group, the proposed location identification scheme (LIS) algorithm adopts an optimization approach on matrix analysis to successively search for the most likely multiple line outage locations. Additionally, the proposed optimal PMUs placement methods: exhaustive search optimal method (ESOM) and greedy heuristic optimal method (GHOM) to find the optimal PMUs placement, targeting on maximizing the average identification capability of multiple simultaneous line outages. The proposed work is evaluated using 14-, 30-, 57-, 300-, and 2383-bus systems. Our experimental study shows that, the proposed solution can effectively explore the system design space and produce low computational complexity real time anomaly detection solutions and location identification of the multiple line outages.

References

1. Andersson G et al (2005) Causes of the 2003 major grid blackouts in north America and Europe, and recommended means to improve system dynamic performance. IEEE Trans Power Syst 20(4):1922–1928
2. (2005) System performance following loss of two or more bulk electric system elements (category c). [Online]. Available: http://www.nerc.com/files/TPL-003-0.pdf
3. Davis C, Overbye TJ (2011) Multiple element contingency screening. IEEE Trans Power Syst 26(3):1294–1301
4. Fang X et al (2012) Smart grid-the new and improved power grid: a survey. IEEE Commun Surv Tutorials 14(4):944–980
5. Tate JE (2008) Event detection and visualization based on phasor measurement units for improved situational awareness. PhD dissertation
6. Tate J, Overbye T (2009) Double line outage detection using phasor angle measurements. In: IEEE power and energy society general meeting, PES'09, pp 1–5
7. He M, Zhang J (2011) A dependency graph approach for fault detection and localization towards secure smart grid. IEEE Trans Smart Grid 2(2):342–351
8. Zhu H, Giannakis G (2012) Sparse overcomplete representations for efficient identification of power line outages. IEEE Trans Power Syst
9. Novosel D et al (2008) Dawn of the grid synchronization. IEEE Power Energy Mag 6(1):49–60
10. Glavic M, Van Cutsem T (2009) Wide-area detection of voltage instability from synchronized phasor measurements. Part I: Principle. IEEE Trans Power Syst 24(3):1408–1416
11. Sodhi R, Srivastava S, Singh S (2012) A simple scheme for wide area detection of impending voltage instability. IEEE Trans Smart Grid 3(2):818–827
12. Jones KD, Thorp JS, Gardner RM (2013) Three-phase linear state estimation using phasor measurements. In: IEEE power and energy society general meeting (PES), pp 1–5

13. Tate J, Overbye T (2008) Line outage detection using phasor angle measurements. IEEE Trans Power Syst 23(4):1644–1652
14. Levorato M, Mitra U (2012) Fast anomaly detection in smartgrids via sparse approximation theory. In: 2012 IEEE 7th Sensor array and multichannel signal processing workshop (SAM), pp 5–8
15. Chen J-C et al (2014) Efficient identification method for power line outages in the smart power grid. IEEE Trans Power Syst 29(4)
16. Walker B (2012) Synchrophasor cost overview. [Online]. Available: http://www.pjm.com/~/media/committees-groups/committees/pc/20121102/20121102-item-03d-installation-cost-overview.ashx
17. Patel M et al (2010) Real-time application of synchrophasors for improving reliability. NERC report, Oct 2010
18. Zhao Y, Goldsmith A, Poor H (2012) On PMU location selection for line outage detection in wide-area transmission networks, pp 1–8
19. Wood AJ, Wollenberg BF (2012) Power generation, operation, and control. Wiley
20. Kezunovic M (2011) Smart fault location for smart grids. IEEE Trans Smart Grid 2(1):11–22
21. Schellenberg A, Rosehart W, Aguado J (2005) Cumulant-based probabilistic optimal power flow (P-OPF) with gaussian and gamma distributions. IEEE Trans Power Syst 20(2):773–781
22. Van Hertem D et al (2006) Usefulness of DC power flow for active power flow analysis with flow controlling devices. In: The 8th IEE international conference on AC and DC power transmission, pp 58–62
23. Herdin M et al (2005) Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels. In: 2005 IEEE 61st vehicular technology conference. VTC 2005-Spring, vol 1, pp 136–140
24. Schäfer J, Strimmer K (2005) A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. Stat Appl Genet Mol Biol 4(1)
25. Chen Y, Wiesel A, Hero AO (2011) Robust shrinkage estimation of high-dimensional covariance matrices. IEEE Trans Signal Process 59(9):4097–4107
26. Van Trees HL (1971) Detection estimation and modulation theory. Wiley, Part I
27. Wei T, Wong MW, Lee Y (1996) Efficient multi-frequency analysis of fault diagnosis in analog circuits based on large change sensitivity computation. In: Proceedings of the fifth Asian test symposium, pp 232–237
28. Starzyk JA et al (2000) Finding ambiguity groups in low testability analog circuits. IEEE Trans Circ Syst I Fundam Theory Appl 47(8):1125–1137
29. Householder AS (2006) The theory of matrices in numerical analysis. Courier Dover Publications
30. Fedi G et al (1998) On the application of symbolic techniques to the multiple fault location in low testability analog circuits. IEEE Trans Circ Syst II Analog Digital Signal Process 45(10):1383–1388
31. Manetti S, Piccirilli MC (2003) A singular-value decomposition approach for ambiguity group determination in analog circuits. IEEE Trans Circ Syst I Fundam Theory Appl 50(4):477–487
32. Reiter R (1987) A theory of diagnosis from first principles. Artif Intell 32(1):57–95
33. Meijerink JA, van der Vorst HA (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric matrix. Math Comput 31(137):148–162
34. Strang G (2003) Introduction to linear algebra
35. Golub GH, Reinsch C (1970) Singular value decomposition and least squares solutions. Numer Math 14(5):403–420
36. Press WH (2007) Section 14.7.2. Kullback-Leibler distance. In: Numerical recipes, 3rd edn: The art of scientific computing. Cambridge University Press
37. Hershey JR, Olsen PA (2007) Approximating the Kullback Leibler divergence between Gaussian mixture models. In: IEEE International conference on acoustics, speech and signal processing, ICASSP 2007, vol 4, pp IV–317
38. Gary MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness

39. Carlson D, Markham TL (1979) Schur complements of diagonally dominant matrices. Czech Math J 29(2):246–251
40. Boyd S et al (1997) Linear matrix inequalities in system and control theory. Proc IEEE 85(4):698–699
41. Zimmerman RD, Murillo-Sánchez CE, Thomas RJ (2011) MATPOWER: steady-state operations, planning, and analysis tools for power systems research and education. IEEE Trans Power Syst 26(1):12–19
42. Power systems test case archive. http://www.ee.washington.edu/research/pstca/
43. González FA, Dasgupta D (2003) Anomaly detection using real-valued negative selection. Genet Program Evolvable Mach 4(4):383–403
44. Zhang Y, Meratnia N, Havinga P (2010) Outlier detection techniques for wireless sensor networks: a survey. IEEE Commun Surv Tutorials 12(2):159–170
45. Burg JP, Luenberger DG, Wenger DL (1982) Estimation of structured covariance matrices. Proc IEEE 70(9):963–974

Domain-Specific Security Approaches for Cyber-Physical Systems

Hui Lin

Abstract In recent years, attacks have emerged in various cyber-physical systems (CPS), causing power outages, disrupting water treatment processes, and so on. These attacks, which are often referred to as advanced persistent threats (APT), reveal a daunting fact. Adversaries are no longer amateurs that randomly probe and compromise many computing devices; they are equipped with advanced intelligence of domain-specific knowledge of the target system and act to achieve a specific goal, e.g., disrupting physical processes. Like a well-trained sniper, adversaries can target a small number of certain devices and can exploit legitimate control operations or well-crafted measurements to inflict physical damage without introducing system- or network-level anomalies. This chapter will present our belief that can effectively address those advanced threats, i.e., integrating domain-specific knowledge of a target system (with the main focus on smart power grids) into general-purpose security solutions. This approach will allow us to reveal adversaries' malicious intentions and preemptively prevent damage from happening.

1 What Are Cyber-Physical Systems?

To better understand ***Cyber-Physical Systems*** (CPSs), we can start with the traditional *control systems*. As shown in Fig. 1, even though control systems can have various appearances, e.g., automobile, medical devices, power grids, and agriculture, they operate on top of a typical feedback loop. This feedback loop involves two types of interactions. One type of interaction involves collecting measurements from physical processes and using them as an input to control algorithms, continuously obtaining the updated models of the physical processes. Another type of interaction involves the commands generated by the control algorithms according to physical states, ensuring the system's operation and long-term stability.

Traditional control systems used legacy sensors and actuators to carry out the two interactions mentioned above. To further increase operational efficiency and

H. Lin (✉)
University of Rhode Island, Kingston, USA
e-mail: huilin@uri.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_16

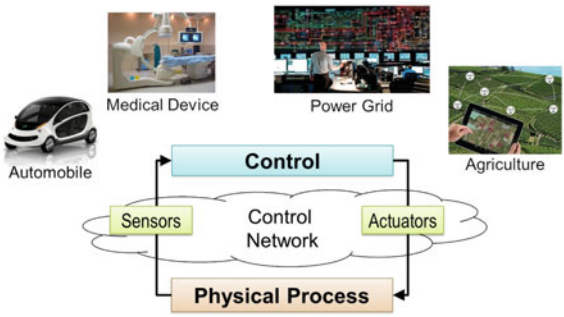


Fig. 1 A typical feedback loop in various control systems. Sensors collect measurements from a physical process such that a control algorithm can accurately monitor the system's run-time states. Actuators are responsible for delivering control commands determined by the control algorithm, maintaining continuous operations in the physical process

reduce administration costs, engineers deploy off-the-shelf computing components and network infrastructure to replace the legacy sensors and actuators. Consequently, control systems evolve into CPSs.

Even though there is no standard definition of CPS, the National Institute of Standards and Technology (NIST) provides the following reference definition [1]. "Cyber-Physical Systems (CPS) comprise interacting digital, analog, physical, and human components engineered for function through integrated physics and logic. These systems will provide the foundation of our critical infrastructure, form the basis of emerging and future smart services, and improve our quality of life in many areas. Cyber-physical systems will bring advances in personalized health care, emergency response, traffic flow management."

Because CPSs can differ significantly, it is challenging to summarize and discuss all security solutions that exploit various domain-specific knowledge of the target systems in a single chapter. To facilitate discussion, we mainly use smart power grids as an example CPS. Specifically, we use the informal definition provided in [2]. "A Smart Grid is a modern electricity system. It uses sensors, monitoring, communications, automation, and computers to improve the flexibility, security, reliability, efficiency, and safety of the electricity system." Many security approaches discussed in this chapter can be generalized to other CPS environments by considering specific domain-specific system knowledge.

2 Attack Surfaces of Cyber-Physical Systems

To better understand how cyberattacks affect CPSs, we use Fig. 2 to illustrate potential attack surfaces, i.e., software or hardware components through which adversaries can penetrate and compromise a CPS. We intentionally present these attack surfaces on

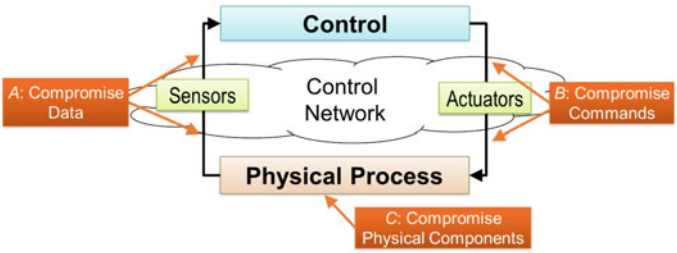


Fig. 2 Attack surfaces mapped to CPS' feedback loop

CPS' generic control loop, abstracted from the detailed implementation of attacks targeting specific CPSs.

In attacks that compromise measurements (often referred to as false or bad data injection attacks, marked as type *A* in Fig. 2), adversaries try to mislead the control algorithm by corrupting the cyber system states [3, 4], which can lead to a wrong command to be issued to the physical process. The impacts of false data injection attacks vary from system to system. For example, [5–7] study how the compromised measurement data would indirectly disrupt control operations and create potential economic losses.

The attacks that compromise measurement data aim at indirect changes of the commands issued to the physical process. However, in today's CPSs, commands are often transmitted over unprotected communication channels of an IP-based *control network*. Suppose an adversary can gain access to the control network or the communication link between the cyber and physical components. In that case, the adversary can disrupt the system by directly compromising the control commands (type *B* attack in Fig. 2). This is not to say that the attacks on sensor measurements are not important. Quite the opposite, compromised measurements can be used to hide the real (potentially anomalous) state of a CPS in order to delay the detection of the attacks before the actual damage to the system (as seen in the example of Stuxnet [8]).

To identify and rank the attacks that exploit the vulnerabilities in physical components (marked as type *C* in Fig. 2), many researchers propose metrics to reveal and quantify different types of vulnerabilities [9, 10]. Specifically, power systems' electrical characteristics, such as loads of substations or transmission lines, can be used to understand how overloading events caused by cyberattacks can lead to safety violations. Also, previous research use graph theory to study the characteristics of transmission or distribution networks in power grids (e.g., connectivity or the length of the shortest path between substations), determining how malicious attacks can propagate through the system [11, 12].

Instead of perturbing physical components simultaneously, adversaries can also perturb physical components in sequence in type *C* attacks, which are referred to as cascaded attacks. In [13], the authors present a brief discussion on the risk of cascaded outages caused by accidents or attacks. Zhu et al. experimentally demonstrate that the

cascaded attack can introduce more significant damage than the attacks that perturb multiple physical components simultaneously [10]. Note that type *C* attacks often require physical accesses to actual CPS devices, which are not easy, less practical, and have a higher risk of being detected.

3 Challenges of Detecting Attacks in CPSs

Because CPSs rely on continuous interactions between cyber and physical components, attacks targeting CPSs distinguish themselves from the attacks in general purpose computing environments in two aspects. First, attacks will result in physical damage, causing irreversible service outages, economic losses, and even human casualties. Second, the complexity of attacks increases dramatically to ensure the effectiveness and stealthiness of maliciously manipulating physical processes. These characteristics introduce new challenges of detecting attacks in CPSs.

3.1 Visibility of System Activities

The strong synergy of cyber and physical components in CPSs makes detecting cyber-physical attacks difficult by monitoring the cyber or physical components separately from each other.

It is difficult to detect and mitigate attacks based solely on the cyber components' activities for two reasons. First, the communication protocols used by CPSs to exchange physical data usually lack security approaches like encryption and authentication that are compatible with legacy devices and real-time network communication. For example, the DNP3 protocol, widely used in the U.S. power grids, chemical plants, and other critical infrastructure, still lacks encryption features [14]. Consequently, adversaries can easily perform reconnaissance by passively monitoring the communication without generating anomalies in the cyber domain. Second, the compromises of the physical process do not necessarily introduce anomalies that the state-of-art intrusion detection systems will raise alerts [15–17]. In other words, attacks can be crafted by changing one valid control command to another valid command without violating any protocol syntax, control flow, or communication performance. For example, modifying a single bit in a DNP3 packet that delivers commands to control circuit breakers can change certain breakers' on/off state. If the commands are changed to different devices, the disturbance can destabilize a power grid instead of performing their original functions.

It is also difficult to detect and mitigate the attacks based solely on the activities from the physical domain. Because many vendors and companies are building physical devices to serve different objectives, collecting and correlating application logs is usually time-consuming (some devices are even not equipped with the logging capabilities in their initial design). In addition, traditional safety procedures are originally

designed to remedy accidents caused by unexpected physical failures, which happen locally. These safety procedures can become ineffective against malicious attacks, which can rely on coordinated activities at different sites. For example, in power grids, traditional contingency analysis considers only low-order incidents (i.e., the "N − 1" or "N – 1 − 1" contingency in which one or two devices are out of service). Consequently, it is impractical to construct a black list of the possible attacks for a large-scale system.

3.2 Diagnosis

Diagnosing attacks play a critical role in preventing the same attacks from occurring in the future. Successful and accurate diagnosis can lead to a thorough understanding of the life cycle of the attacks and the identification of vulnerabilities in the systems.

However, attacks are hard to diagnose in CPSs because the complexity of the attacks increases dramatically. Furthermore, although many cyber-physical attacks cause safety violations, the violations themselves do not reveal the entry point of the attacks and the malicious activities in the cyber domain. Without such information, it is challenging to identify the vulnerability exploited by adversaries and, thus, to perform appropriate response or remedy actions (e.g., software patching or updating operational procedures). To make things worse, intelligent adversaries can hide their existence by disguising themselves as accidental failures [18] to avoid further diagnosis on the attacks.

3.3 Real-Time Constraints

Physical components in CPSs, which usually observe some physical laws, usually have strict requirements for the timely delivery of control operations. However, those requirements can span across different ranges. For example, power grids need to deliver the commands in the range from several hundred milliseconds to several seconds [19], while the surgical robots are required to perform control computations within only a few milliseconds [20]. Consequently, it is difficult to design a one-size-fits-all solution to achieve a run-time detection for all ranges of CPSs.

Because of stringent real-time constraints on control operation, it becomes challenging to make appropriate remedy decisions on the detected attacks. On the one hand, passively generating alerts, like how we handle intrusions in general-purpose computing environments, will not directly prevent physical damage. On the other hand, placing a detection module in a communication path, like how we deploy firewalls, can block malicious traffic. But this approach can affect the performance of all network communications, including benign ones, making it difficult for the piggybacked operations to observe existing timing constraints.

4 Attacks Detection in CPSs

Despite all those challenges in handling attacks in CPSs, there are varieties of methods proposed to achieve accurate and reliable detection. In this section, we will present different detection methods based on three categories, i.e., anomaly-based, misuse-based, and specification-based approaches, which are common categories when discussing intrusion detection in general-purpose computing environments.

4.1 Anomaly-Based Detection in CPSs

Anomaly-based detection refers to the detection methods using system activities that deviate from a normal profile. Usually, system administrators build a normal profile in advance by summarizing statistical characteristics extracted from activities when the target systems are operated without involving any malicious actors. Because of this detection concept, many machine-learning and data-driven analytic techniques are applied to build the statistical characteristics of normal system behaviors, serving as a norm to the target systems. In the domain of CPS, we can build the norm based on the data from both cyber and physical domains.

4.1.1 Buildings Norm from Cyber Domains

In a CPS, cyber domains, e.g., communications networks connecting various physical devices, can include valuable knowledge indicating the trajectory of physical states. Using cyber-domain information to build the norm of CPSs' run-time behavior can be easily achieved by following experiences developed in general-purpose computing networks; it often requires small changes on the configuration of existing computing and network infrastructures. For example, to monitor a network, system administrators can configure an unused port in a network switch or a network router as a SPAN port that can mirror all network traffic going through the switch or the router to this port. Because we analyze the copy of existing information, the detection performed based on the norm built from the cyber domains is often non-intrusive. Even though non-intrusive detection cannot directly prevent malicious activities, it benefits by introducing few new vulnerabilities or risks to the existing environment.

Because of these advantages, monitoring cyber domain activities and using them to detect anomalies are proposed when there are few domain-specific methods for various CPSs. In these efforts, security solutions mainly rely on specific patterns observed based on the data extracted from the lower-layer of communications networks, i.e., transport or IP layer of network packets. For example, based on the observation that physical operations are usually periodic (i.e., measurements are collected periodically), Markman et al. divided time-stamped traffic flows into a sequence of bursts and then built a deterministic finite automaton for each burst of

the traffic [21]. Compared to treating all traffic equally, the automata built for each burst can reflect the characteristics of the underlying physical process. Alternatively, Formby et al. used the differences in the time stamps recorded at consecutive TCP layer packets to infer the execution time of certain physical operations, which was further used as device's fingerprints [22].

4.1.2 Build Norm from Physical Domains

Even though some attacks in CPSs present some anomalies in the cyber domains, modern attacks, e.g., Stuxnet and the attack that shut down a Ukrainian power plant in 2015 [8, 23], demonstrate that malicious activities can leave few red flags in communication patterns. Adversaries can hide their malicious intentions in the semantics crafted in legitimate formats. Detecting this type of stealthy attack requires more information from physical domains, revealing ground truths from the physical processes. These ground truths follow the law of physics. System administrators observe those physical laws to maintain CPS' continuous and secure operations while adversaries are required to observe them to achieve their malicious intentions. Because those properties remain consistent with the law of physics, some works refer to those properties as "control invariants" to describe the norm built from the physical domains [24, 25].

There are two major obstacles to obtaining the data from the physical domains. The first one is that many legacy devices used in CPSs, such as Programmable Logic Controllers (PLCs), lack logging mechanisms at their initial designs. Consequently, lacking first-hand records on what happens locally in each device makes it difficult to monitor run-time trajectories of physical processes and identify any deviation from the norm or the expected trajectories. The second obstacle is that many legacy devices rely on proprietary network protocol to exchange information, e.g., what data they collect and what operations they perform. While some proprietary network protocols are merged to the TCP/IP protocol stack, allowing modern network monitoring tools for analysis, some extinct as early documentation are lost and engineers who designed them at first hand retire. These closed designs of proprietary protocols also make it challenging to indirectly estimate the physical state trajectory based on the network interactions with the target devices.

The direct solution to overcome the first obstacle requires vendors to add logging capability to their products. Unfortunately, even though some vendors have made such efforts, the logs added to their devices are far from comprehensive. For example, smart meters log simple records such as functional codes corresponding to certain control operations at a certain time [26]. Furthermore, even though it is possible to add custom logs, it requires instrumentation on the applications or firmware deployed in those legacy devices, which are implemented under proprietary development processes [27]. To make things worse, more complicated logs, like what we find in Syslog in Linux OS, become difficult to implement because of the limited storage space and computational capability found in legacy devices.

The path to overcoming the second obstacle is not smooth, despite many efforts from industrial sectors. Wireshark has provided support for some proprietary protocols widely used in CPSs, e.g., DNP3 and Modbus [14, 28]. Wireshark can extract information encoded in those protocols for clear display, but it cannot analyze those pieces of information. Digital Bond has performed early efforts to explore the signature-matching rules in commercial tools like Snort to analyze limited types of information related to CPSs' run-time operations. Still, it is difficult for Snort to provide complete support on the proprietary protocols. Zeek, another popular intrusion detection system (IDS) based on specification-based detection methodology, provides an interface known as Binpac to add parsers that can fully support some proprietary protocols [29]. In addition, network administrators can leverage Zeek's Turing complete script language to implement their own security policy to accommodate various operational environments. One limitation of Binpac included in the early version of Zeek (when Zeek is known by the name of Bro) is that it only supports application-layer protocols on top of TCP or UDP. Consequently, it is difficult to add parsers of protocols in Ethernet/IP that define new services in data link or transport layers [30]. Fortunately, the successor of Binpac, known as Spicy, included in the current version of Zeek, has gradually provided the support of protocols defined in any layers of TCP/IP stack [31].

4.1.3 Discussion

The major advantage of the data-centric approach is that it can apply machine-learning and data-driven analytic techniques to data from either cyber or physical domains or both, despite different CPS implementations. Specifically, the advancement of deep learning provides a new opportunity to build norms around both cyber and physical domains of a CPS, based on which to detect run-time anomalies.

However, applying anomaly-based detection in CPS can still face multiple challenges [32]. First, there exists a semantic gap between statistical deviations from normal system profiles and the physical impact of attacks on CPS, without considering the domain-specific characteristics of a CPS when cyberattacks happen. In other words, since anomaly does not necessarily represent attacks that can introduce physical damage, this detection method tends to introduce a large number of false-positive alerts, which can affect the performance or even the safety of normal control operations. Second, it is difficult to obtain a representative dataset to train the anomaly-based detection methods. A large number of representative data plays a critical role in ensuring the performance of machine-learning-based or deep-learning-based methods. But physical processes in many CPSs, such as power grids, can experience slow inertia from mechanical components; it can take a long time to collect data representing various operating conditions.

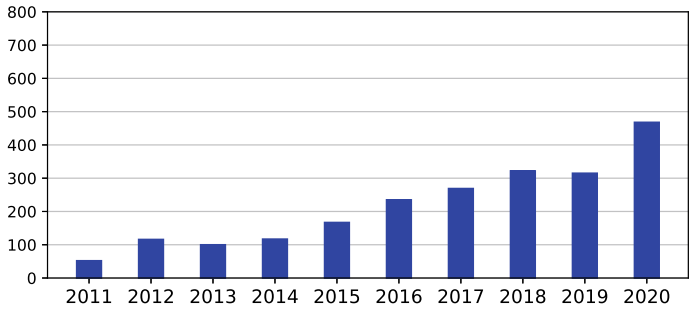


Fig. 3 Number of disclosed vulnerabilities targeting CPS per year from 2011 to 2020 (*source* IBM Security X-Force [34])

4.2 Misuse-Based Detection

Misuse-based detection methods, also known as signatures-based detection, detect attacks based on the knowledge generalized from previously observed attacks. System administrators can generalize the knowledge of previous attacks at a high level, such as building a state machine that describes steps that an adversary usually takes to reach his or her goal [33]. In an alternative way, system administrators can generalize the knowledge of previous attacks at a low level, such as binary patterns found in malware or network payloads. The typical example of this low-level attack knowledge is the anti-malware software used in general-purpose operating systems.

In recent years, the number of discovered vulnerabilities associated with physical devices and incidents that disrupt CPSs' operations has increased dramatically. In Fig. 3, we present a breakdown of those vulnerabilities, summarized by IBM research [34]. In addition, in 2021 alone, we observe multiple major incidents, from botnet attacks on Florida water treatment plants to ransomware attacks on Colonial Pipelines [35, 36].

In general-purpose computing environments, system administrators often transfer the lessons learned from previous attacks into future defense intelligence, being shared in public and making the same type of attacks challenging to succeed in the future. But the situation is very different for attacks in CPSs, which can significantly jeopardize the reputations of the affected utility companies and lead to huge economic losses. As a result, many details of the attacks that happened in CPSs are hidden, let alone being shared among potential competitors. Due to this reason, misuse-based detection is not widely used in CPS network environments.

4.3 Specification-Based Detection

Specification-based detection relies on a specific model or policy constructed based on the internal logic of the monitored system. At run-time, any system behavior

that deviates from this policy or model triggers alerts. It is common to confuse the specification-based detection with the anomaly-based detection discussed in Sect. 4.1. These two detection methods have critical differences. First, anomaly-based detection relies on the deviation of normal system behavior profiled through the observation of attack-free system behaviors. Many anomaly-based detection methods encode the generation of the attack-free system behavior as statistical characteristics. Specification-based detection relies on the deviation of expected system behaviors determined by the system's internal logic. Second, anomaly-based detection only generalizes attack-free behaviors, while specification-based detection can rely on internal logic that specifies both what is expected and what is not. In other words, security operators can use techniques in but not limited to both anomaly-based and misuse-based detection to estimate system's internal logic.

Similar to Sect. 4.1, in the domain of CPS, we can specify the internal logic of a CPS based on either cyber components or physical components.

4.3.1 Building Internal Logic Based on Cyber Domain Knowledge

Using IP-based communications is essential to advance traditional control systems into modern CPSs. Even though CPSs have relied on similar network infrastructure used by general-purpose computing environments, they exchange information for a different objective, i.e., monitoring and maintaining the run-time states of physical processes. In recent years, many approaches attempt to use various sources from the cyber domains of a CPS to build the internal logic of a target system. In the remainder of this section, we present two important sources: specification of network protocols and system documentation.

Even though many network protocols widely used in CPSs are mapped to the TCP/IP stack from their original proprietary designs, they still present some unique functionalities to CPS environments. Those functionalities are closely related to control operations, which are uncommon in general-purpose computing and network environments. For example, the DNP3 protocol, originally defined as operating over a serial link, reflects the end devices' run-time state and state transitions [14]. Specifically, Sect. 6.3 of [14] specifies how the implementation of a DNP3 master should handle solicited responses by defining a state machine including three different states:

- **Idle state**: the DNP3 master waiting from user inputs to initiate a request. The DNP3 master starts in this state after a reset operation.
- **AwaitFirst state**: the DNP3 master is expecting the first fragment of appropriate responses from DNP3 outstations. Because there can be multiple fragments of responses for a DNP3 request (e.g., each fragment is sent by a single TCP packet), the DNP3 master is put in this state after sending a request and before receiving the first fragment of the response.
- **Assembly state**: the DNP3 master enters this state after receiving the first fragment of the expected response from DNP3 outstations. In this state, the DNP3 master

assembles all fragments, obtaining the complete response. After that, the DNP3 master will enter into the **Idle** state.

In other words, physical devices designed to conform to the DNP3 protocol will follow the state transition specified in the protocol. Consequently, the protocol specification becomes a valuable source to build the internal logic of certain physical devices.

Because of the original proprietary nature of the network protocols used in CPS environments, there are a limited number of network analysis tools that can understand the protocol specifications and extract information that can help build the internal control logic of end devices. As discussed in 4.1.2, network analysis tools, such as Wireshark and Zeek, can be used for this different objective.

In addition to the protocol specification, system documentation related to those legacy devices are another critical source for understanding the internal logic related to control operations. System documentation can be encoded in both structured and unstructured formats.

Structured Format	Many utilities and standards adopt markup language, such as XML, to define a set of rules that are both machine-readable and human-readable. For example, IEC 61,850 defines a basic structure known as Substation Configuration Language to store functionalities related to substation automation in power grids [37]
Unstructured Format	In many other situations, system specifications are often stored in documentation that is only human-readable. Manually obtaining internal logic from the documentation can be time-consuming. However, the current advancement of natural language processing based on recent advancement of deep recurrent neural networks can increase the efficiency to automatically learn the internal logic from human-readable documentation [38]

4.3.2 Building Internal Logic Based on Physical Domain Knowledge

Because a CPS is a digital upgrade from a control system, the activities in its physical domains are still governed by physical laws. For example, we can use the following differential equations, often in a vector format, to describe physical states and their variations in a time domain.

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1}$$

$$y(t) = Cx(t) + Du(t), \tag{2}$$

In Eq. (1), $x(t)$ is a vector of state variables and $u(t)$ is a vector of input signals of a physical process. The derivative of $x(t)$ over time t, specified by $\dot{x}(t)$, describes

the changes of the state variable. In Eq. (2), $y(t)$ is a vector of output signals. These two equations determine the next state and output of the system based on the values of current states and input signals.

In various CPSs, the meaning of state variables, input signals, output signals, as well as configurations correlating them (which are encoded in parameters A, B, C, and D), will change accordingly. For example, a power grid connects different substations to deliver energy from generators to load units. In this environment, system states refer to the voltage magnitude and the phasor angle at each substation; the input signals refer to the setpoints at different generators; the output signals refer to the measurements from various types of meters. The physical configurations correlating those elements refer to the topological connectivity of substations and physical properties of the connection lines, e.g., the impedance of transmission or distribution lines.

Because CPS' physical states need to follow certain physical laws, both malicious attacks and defenses observe such special requirements in CPSs. For example, in the attack of Ukraine power plants, adversaries cause the attack by issuing a wrong input signal, which results in an output that can lead to a blackout. Even though the input signals present minor cyber anomalies, it will be easy to use the physical laws to estimate the potential physical consequences and reveal adversaries' malicious intentions. Such understanding provides a foundation for multiple research works, which use the physical laws to build internal logic and detect attacks based on the results estimated based on the logic [20, 25, 39].

Even though internal logic built from the physical models can provide accurate detection, solving the physical models to estimate potential consequences can take a long time to finish. For example, Eqs. (1) and (2) can include at least 8000 parameters for a power grid including more than 2000 substations. Solving solutions for such a large grid can take seconds to finish. Even though this latency seems small for an energy management system, it is at least two orders of magnitude larger than the latency that normal network analyzers spend on single network activity.

There are two major approaches to overcome the shortcoming of the long latency of solving physical models. The first approach is to develop a new approximation algorithm to solve the physical model. Taking power grids for example, we often use the AC power flow analysis algorithm to solve the nonlinear equation used to specify power grids' physical models. The AC power flow analysis uses iterative algorithms, e.g., the Newton–Raphson algorithm, to accurately calculate the power system's state within an accuracy level. To avoid iterative computations, different types of DC power flow analyses solve the linear approximation of the nonlinear power flow equations [40]. The DC power flow analysis enjoys reduced computation latency at the cost of poor solution accuracy. To meet the trade-off of accuracy and detection latency, adaptive AC power flow analysis is proposed in [39]. This algorithm uses intermediate solutions from the iterative algorithm as the final solution before the iterative algorithm is accurately converged. Consequently, system administrators can dynamically choose when to stop the algorithm according to the accuracy level of run-time detection.

The second approach to reducing the latency of solving physical models is to train data-driven algorithms to approximate the physical models. For example, deep neural networks can be very effective in describing complicated physical models, if sufficient and various operating conditions of the physical models are provided. Consequently, solving the physical models is replaced by performing an interference on a trained machine learning model, significantly reducing the computational latency.

4.3.3 Discussion

Unlike anomaly-based detection, specification-based detection is used in many actual general-computing environments. However, when applying this detection method in CPS environments, a new challenge emerges. Many utility companies will be reluctant to reveal the internal logic to a third-party security solution, revealing multiple concerns. The first concern is that internal logic is valuable for both adversaries and defenses. As discussed in [23, 41], adversaries put a great effort on the reconnaissance of the internal logic of a target system to launch effective attacks. The second concern is that the internal logic, which often involves proprietary or patented design, can be used by the competitor of a utility company for unfair competition.

5 Attack Recovery in CPSs

The resilience of a CPS describes its capability to recover from anomaly incidents, e.g., accidental events or intentional cyberattacks. After discussing attack detection in Sect. 4, we will focus on attack recovery in this section.

CPSs can recover from small disturbances relying on their intrinsic feedback control loop shown in Fig. 1, which is mainly designed and implemented through mechanical components. The feedback control can be implemented locally in substations and globally in energy management systems, adjusting power generations to accommodate continuously-changing load demands (and possibly with the minimum costs).

However, the feedback control loop becomes less effective to unprecedented disturbances observed in cyberattacks. For example, suppose a power grid experiences massive disturbance caused by manipulating a large number of Internet-of-things (IoT) [42, 43] (e.g., load demand changes by 30% simultaneously). In that case, the affected power grids will end up in a blackout even with the feedback control loop. Therefore, in addition to traditional feedback controls, many recovery mechanisms emerge to restrict the impact of disturbances caused by cyberattacks.

5.1 Attack Recovery in Cyber Domains

The communications networks are vulnerable to various attacks, such as denial-of-service attacks (DoS) or man-in-the-middle (MITM) attacks, leading to the disconnections of certain communication links and end nodes. Furthermore, the service downgrade in the communications networks can indirectly affect the performance of the control operations in a CPS. For example, in a power grid, control commands and measurements are required to observe a demanding communication latency, ensuring accurate wide-area monitoring in power systems and real-time control operations.

Unfortunately, recovering network services, e.g., reconnecting communication links and deploying duplicate end nodes, is not equivalent to restoring control operations in a CPS. In the design of virtual circuit switching networks, such as the asynchronous transfer mode (ATM) network in the early 1990s [44, 45], the concept of self-healing has been proposed to handle link or node failure. The self-healing algorithms try to recover as many lost services as possible under the resource constraint of network equipment. In this general-purpose network environment, self-healing is performed on predetermined backup or protection paths [46]. However, in CPS' network environment, these self-healing algorithms may not become effective. First, the objective in CPS is to restore control operations, e.g., the observability of a power grid. This objective is different from the one that is used to restore failed links or nodes in conventional ATM networks, e.g., minimizing the cost of assigning spare links [45], maximizing the amount of restored traffic [47, 48], and maximizing the volume of remaining capacity in routing paths [46, 49]. The main reason is that communication nodes can play different roles in CPS' physical processes; their restoration should be prioritized according to their roles in physical processes, not their roles in communications networks. Second, the self-healing mechanisms for conventional networks consider the failure of a small range of components, e.g., single link or node failures caused by accidental events. But cyberattacks can cause the failure of a large number of communication nodes, especially the ones used in CPSs.

Because of those differences, methods that aim to restore the services of physical processes (e.g., the observability and controllability of a power grid) begin to show up. For example, in [50], Lin et al. attempt to restore the network links and failure nodes to restore power grids observability. The problem specifically targets lost measurements from phasor measurement units (PMU) if their upper-level data collector (also known as PDC, phasor data concentrator) experiences failure. Specifically, the proposed self-healing networks prioritize recovering the data from PMUs based on their role in increasing power grid observability, instead of increasing network throughput.

Similarly, in [51, 52], the authors quantified the relationship between the network connectivity of distributed generation units and the performance of generational control in power grids. This quantification metric serves as a guideline for recovering network links that maximize power grids' controllability while maintaining small network overhead.

5.2 Attack Recovery in Physical Domains

CPSs are highly vulnerable to massive disruptions. For example, massive disruptions in Texas due to snowstorms in 2021 caused power outages in more than 2 million households, economic losses of more than 80 billion dollars, and more than 150 people losing their lives [53, 54]. If cyberattacks cause those massive disruptions, CPSs' current capability is still limited in recovering from the disruptions.

Despite those challenges, many utility companies attempt to remedy the impact of massive disruption by allocating more resources. For example, to remedy a significant amount of load changes in a power grid triggered by manipulating a large number of IoT devices, some research works propose to allocate more generation reserve. This approach can help the power grid to quickly recover from a certain level of disruption. But it comes with a big price: an increased investment in generation units that will be rarely used during normal operations. Overcoming the drawbacks of allocating more generation reserves requires using distributed energy resources, such as solar power and wind plant, instead of investing in traditional bulky power generations. In addition, technological advancement in energy storage needs to catch up, making it possible to reserve generation for later usage. Finally, similar to power grids, other CPSs will need to leverage their renewable resources to recover from the disruptions caused by attacks.

6 Preemptive Protection

Many CPS environments still follow the traditional and passive defense paradigm, which detects first and recovers afterward. Unlike passive defense, preemptive protections are alternative approaches that can disrupt adversaries, especially the intelligent adversaries that stay stealthy to obtain in-depth knowledge of the target CPS environment before initiating any malicious activities. The following section discusses preemptive protection methods against two types of adversaries, i.e., external and internal adversaries determined by whether they have penetrated the isolated control networks used by CPSs.

6.1 Preemptive Protection Against External Adversaries

During the early stage of cyber-physical attacks, external adversaries exploit scanning or probing to search for potential vulnerable physical devices exposed to the public Internet. However, we still lack in-depth knowledge of how adversaries behave at the early stage, because most utility companies tend to limit access to real incidents that happened before to protect companies' reputations. This knowledge can play a critical role in stopping them before they initiate following malicious activities. There are

mainly two approaches attempting to obtain this knowledge, i.e., leveraging network telescoping and building CPS honeypots.

The basic idea of network telescoping is to observe traffic that interacts with unused address spaces on the public Internet. Because legitimate and correctly configured applications target existing services, the traffic destined to the unused address spaces is very likely from suspicious entities. An open research question related to network telescoping is distinguishing traffic issued by legitimate but misconfigured entities from the traffic issued by actual adversaries. It is commonly known that the traffic from these two groups can present different statistical characteristics. After filtering out traffic from misconfigured entities, the remaining traffic can reveal previous knowledge about adversaries. Following this method, Fachkha et al. specifically focused on the traffic in the network telescoping based on common CPS protocols [55], revealing knowledge such as the distribution of adversaries' location and the type of network protocols that adversaries prefer. However, the network telescoping is passive without interacting with the adversaries, still lacking the knowledge of adversaries' long-term activities on target CPSs. Meanwhile, the analysis remains on the transport layer of network packets, without deep-packet inspection on the traffic observed in the network telescoping.

Honeypots or honeynets can interact with adversaries with simulated network environments [56, 57]. Several honeypot projects aim to build separate computing or network environments to trace adversaries' activities on CPS devices, e.g., PLCs [58–60]. Han et al. further propose to use software-defined networking (SDN) to automate interactions with adversaries [61]. Those CPS honeypots can mimic CPS' cyberinfrastructure. However, in their constructed networks, the honeypots lack supports for constructing meaningful application-layer payloads, e.g., measurements exchanged between physical devices. For example, Conpot presents a common interface to include physical measurements used in its simulated network environment, but it is up to the users to create those measurements. The lack of meaningful physical measurements can easily expose the fake environment of honeypots, losing their attractions to adversaries.

6.2 Preemptive Protection Against Internal Adversaries

In general, honeypots or honeynets, are deployed to attract adversaries, making them less interested in real devices. When internal adversaries have already penetrated internal control networks and accessed physical devices in remote field sites, honeypots will become less useful.

For internal adversaries, different groups of moving target defenses (MTD) can disrupt adversaries' knowledge related to real devices. Traditional MTD approaches disrupt adversaries by randomly changing system and network configurations, e.g., IP addresses and port numbers [62–65]. These approaches can be applied in the internal control networks, especially with the help of advanced network technology like SDN and edge computing. Some recent works leverage similar designs to disrupt control

operations in CPSs. For example, Rahman et al. randomly change the physical data set used for power system analysis, attempting to remove some compromised data and to reduce the effectiveness of false data injection attacks [66]. Another MTD group intentionally disrupts the physical process in a CPS and uses deviations from expected consequences to detect attacks [67–70]. Those approaches require physical perturbations, which can harm the existing physical process.

To avoid modifications to existing physical processes, Lin et al. propose to provide misleading information about power grids' cyber and physical infrastructures to suspicious adversaries, aiming to disrupt adversaries' reconnaissance. Instead of mimicking and simulations, which can easily expose fake environments, this approach leverage a network control application based on SDN to "hook" network interactions with real devices and use them as network flows of non-existing nodes. This design will build lightweight virtual nodes that follow the actual implementation of network stacks, physical state variations, and system invariants of real physical devices in power grids. Meanwhile, in [71, 72], Lin et al. also proposed a method to craft physical measurements that follow physical laws in power grids, serving as a noise to mislead adversaries into designing ineffective attack strategies.

7 Security Issues Associated with Advanced Computing Technologies

CPSs evolve from traditional control systems by upgrading computing and communication technologies, which help to increase computation capabilities and reduce communication latency. To further increase operation automation and decentralized control, advanced computing technologies boosted by artificial intelligence (AI) gradually become a critical component in future CPSs. For example, many companies have put considerable investments in semi or fully automated driving, relying heavily on the advancement of AI technologies. Critical infrastructures like power grids also begin to embrace such changes. Shi et al. summarize the state-of-the-art AI-based techniques to analyze and regulate feedback control used in power grids to ensure long-term stability [73].

Because AI analytic can become the target of cyberattacks, a new question emerges for the future AI-enhanced CPS environment: how does the resilience of AI analytic affect the resilience of CPS' control functionality. AI analytic is boosted mainly by the advancement of deep neural networks (DNN), which are applied widely in unsupervised, supervised, and reinforcement learning. Adversaries can downgrade AI performances by disrupting training/inference procedures or providing adversarial training samples. When mapping this threat in a CPS environment, we find that adversaries can leverage the similar attack surface presented in Chap. 2. Specifically, adversaries can still compromise measurements collected by AI-based control algorithms or identify vulnerabilities in those algorithms. Note that semantic gaps exist between the performance downgrade in AI analytic and physical disruptions

of a CPS. Those semantic gaps will make attacks more challenging to realize but also make them more stealthy to detect. For example, Jha et al. discussed how adversaries indirectly affect the safety of autonomous vehicles by leveraging adversarial generative networks [74].

8 Summary

This chapter discusses the unique characteristics of cyberattacks targeting CPSs for disruptions and how those characteristics drive security solutions aiming to prevent disruptions. Through decades of research and applications, we learned that cyberattacks are no longer initiated by amateur adversaries; attacks like advanced persistent threats become the major actor, searching for devastating physical disruption without leaving traces. To detect those attacks effectively, we believe that security solutions should be designed by exploiting adversaries' very objective, i.e., physical disruption. This understanding motivates many research approaches to integrate domain-specific knowledge of a CPS into general-purpose security solutions. Regardless of being used for passive detection, attack recovery, or preemptive defenses, the security solutions rely on data closely associated with physical states. Consequently, even though attacks can be stealthy, we can reveal adversaries' malicious intentions and preemptively prevent the damage from happening. As CPSs play an essential role in future industrial control systems (sometimes known as the Industry 4.0), it is critical for security solutions to evolve together with the advancements found in existing CPSs, maintaining resilience in this exciting area.

References

1. Cyber Physical Systems | NIST [Online]. Available at: https://www.nist.gov/el/cyber-physical-systems
2. Singer J (2009) Enabling tomorrow's electricity system: report of the Ontario smart grid forum
3. Kosut O, Jia L, Thomas RJ, Tong L (2011) Malicious data attacks on the smart grid. IEEE Trans Smart Grid 2(4):645–658. https://doi.org/10.1109/TSG.2011.2163807
4. Liu Y, Ning P, Reiter MK (2011) False data injection attacks against state estimation in electric power grids. ACM Trans Inf Syst Secur 14(1):1–33. https://doi.org/10.1145/1952982.1952995
5. Giraldo J, Cárdenas A, Quijano N (2017) Integrity attacks on real-time pricing in smart grids: impact and countermeasures. IEEE Trans Smart Grid 8(5):2249–2257. https://doi.org/10.1109/TSG.2016.2521339
6. Tan R, Badrinath Krishna V, Yau DK, Kalbarczyk Z (2013) Impact of integrity attacks on real-time pricing in smart grids. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security, CCS '13. Association for Computing Machinery, New York, NY, USA, pp 439–450. https://doi.org/10.1145/2508859.2516705
7. Xie L, Mo Y, Sinopoli B (2011) Integrity data attacks in power market operations. IEEE Trans Smart Grid 2(4):659–666. https://doi.org/10.1109/TSG.2011.2161892
8. Falliere N, Murchu LO, Chien E (2011) W32. stuxnet dossier. White paper, Symantec Corp., Security Response 5(6):29

9. Zhu Y, Yan J, Sun Y, He H (2014) Revealing cascading failure vulnerability in power grids using risk-graph. IEEE Trans Parallel Distrib Syst 25(12):3274–3284. https://doi.org/10.1109/TPDS.2013.2295814
10. Zhu Y, Yan J, Tang Y, Sun YL, He H (2014) Resilience analysis of power grids under the sequential attack. IEEE Trans Inf Forensics Secur 9(12):2340–2354. https://doi.org/10.1109/TIFS.2014.2363786
11. Hines P, Cotilla-Sanchez E, Blumsack S (2010) Do topological models provide good information about electricity infrastructure vulnerability? Chaos: Interdiscipl J Nonlinear Sci 20(3):033122. https://doi.org/10.1063/1.3489887
12. Lesieutre BC, Pinar A, Roy S (2008) Power system extreme event detection: the vulnerability frontier. In: Proceedings of the 41st annual Hawaii international conference on system sciences (HICSS 2008), pp 184–184. https://doi.org/10.1109/HICSS.2008.350
13. Vaiman M, Bell K, Chen Y, Chowdhury B, Dobson I, Hines P, Papic M, Miller S, Zhang P (2012) Risk assessment of cascading outages: methodologies and challenges. IEEE Trans Power Syst 27(2):631–641. https://doi.org/10.1109/TPWRS.2011.2177868
14. IEEE standard for electric power systems communications-distributed network protocol (dnp3) (2012). https://doi.org/10.1109/IEEESTD.2012.6327578 [Online]. Available at https://standards.ieee.org/standard/1815-2012.html
15. Peterson D Intrusion detection and cyber security monitoring of SCADA and DCS networks, p 8
16. Ten C, Hong J, Liu C (2011) Anomaly detection for cybersecurity of the substations. IEEE Trans Smart Grid 2(4):865–873. https://doi.org/10.1109/TSG.2011.2159406
17. Valdes A, Cheung S (2009) Communication pattern anomaly detection in process control systems. In: 2009 IEEE conference on technologies for homeland security, pp 22–29. https://doi.org/10.1109/THS.2009.5168010
18. Chung K, Li X, Tang P, Zhu Z, Kalbarczyk ZT, Iyer RK, Kesavadas T (2019) Smart malware that uses leaked control data of robotic applications: the case of raven-ii surgical robots. In: 22nd international symposium on research in attacks, intrusions and defenses (RAID 2019). USENIX Association, Chaoyang District, Beijing, pp 337–351. https://www.usenix.org/conference/raid2019/presentation/chung
19. IEEE standard communication delivery time performance requirements for electric power substation automation (2005) IEEE Std 1646-2004, pp 1–36. https://doi.org/10.1109/IEEESTD.2005.95748
20. Alemzadeh H, Chen D, Li X, Kesavadas T, Kalbarczyk ZT, Iyer RK (2016) Targeted attacks on teleoperated surgical robots: dynamic model-based detection and mitigation. In: 2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 395–406. https://doi.org/10.1109/DSN.2016.43
21. Markman C, Wool A, Cardenas AA (2017) A new burst-dfa model for scada anomaly detection. In: Proceedings of the 2017 workshop on cyber-physical systems security and PrivaCy, CPS'17. Association for Computing Machinery, New York, NY, USA, pp 1–12. https://doi.org/10.1145/3140241.3140245
22. Formby D, Srinivasan P, Leonard A, Rogers J, Beyah R (2016) Who's in control of your control system? Device fingerprinting for cyber-physical systems. In: Proceedings 2016 network and distributed system security symposium. Internet Society. https://doi.org/10.14722/ndss.2016.23142
23. Lee RM, Assante MJ, Conway T (2016) Analysis of the cyber attack on the Ukrainian power grid. Technical report, SANS and E-ISAC
24. Chen Y, Poskitt CM, Sun J (2018) Learning from mutants: using code mutation to learn and monitor invariants of a cyber-physical system. In: 2018 IEEE symposium on security and privacy (SP), pp 648–660. https://doi.org/10.1109/SP.2018.00016
25. Choi J, Jeoung H, Kim J, Ko Y, Jung W, Kim H, Kim J (2018) Detecting and identifying faulty iot devices in smart home with context extraction. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 610–621. https://doi.org/10.1109/DSN.2018.00068

26. Ali MQ, Al-Shaer E (2013) Configuration-based ids for advanced metering infrastructure. In: Proceedings of the 2013 ACM SIGSAC conference on computer & communications security, CCS'13. Association for Computing Machinery, New York, NY, USA, pp 451–462. https://doi.org/10.1145/2508859.2516745
27. Keliris A, Maniatakos M (2019) Icsref: a framework for automated reverse engineering of industrial control systems binaries. In: Proceedings of the 2019 annual network and distributed system security symposium (NDSS 2019)
28. Modbus messaging on tcp/ip implementation guide v1b (2006) [Online]. Available at: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf
29. Lin H, Slagell A, Di Martino C, Kalbarczyk Z, Iyer RK (2013) Adapting bro into SCADA: building a specification-based intrusion detection system for the DNP3 protocol. In: Proceedings of the eighth annual cyber security and information intelligence research workshop (CSIIRW '13), pp 5:1–5:4. https://doi.org/10.1145/2459976.2459982
30. Ethernet/IP Overview [Online]. Available at: https://www.odva.org/Technology-Standards/EtherNet-IP/Overview
31. Spicy—generating robust parsers for protocols & file formats [Online]. Available at: https://docs.zeek.org/projects/spicy/en/latest/
32. Sommer R, Paxson V (2010) Outside the closed world: on using machine learning for network intrusion detection. In: 2010 IEEE symposium on security and privacy, pp 305–316. https://doi.org/10.1109/SP.2010.25
33. Sharma A, Kalbarczyk Z, Iyer R, Barlow J (2010) Analysis of credential stealing attacks in an open networked environment. In: 2010 fourth international conference on network and system security, pp 144–151
34. X-Force Threat Intelligence Index 2001 (2021) [Online]. Available at: https://www.ibm.com/security/data-breach/threat-intelligence
35. Duffy C (2021) Colonial pipeline attack: a 'wake up call' about the threat of ransomware [Online]. Available at: https://www.cnn.com/2021/05/16/tech/colonial-ransomware-darkside-what-to-know/index.html
36. Wainwright M (2021) The Florida water plant attack signals a new era of digital warfare [Online]. Available at: https://www.darktrace.com/en/blog/the-florida-water-plant-attack-signals-a-new-era-of-digital-warfare-its-time-to-fight-back/
37. Hoga C, Wong G (2004) IEC 61850: open communication in practice in substations. In: IEEE PES power systems conference and exposition, vol 2, pp 618–623. https://doi.org/10.1109/PSCE.2004.1397694
38. Caselli M, Zambon E, Amann J, Sommer R, Kargl F (2016) Specification mining for intrusion detection in networked control systems. In: 25th USENIX security symposium (USENIX security 16). USENIX Association, Austin, TX, pp 791–806. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/caselli
39. Lin H, Slagell A, Kalbarczyk Z, Sauer P, Iyer RK (2018) Runtime semantic security analysis to detect and mitigate control-related attacks in power grids. IEEE Trans Smart Grid 9(1):163–178. https://doi.org/10.1109/TSG.2016.2547742
40. Stott B, Jardim J, Alsac O (2009) Dc power flow revisited. IEEE Trans Power Syst 24(3):1290–1300
41. Lee D (2017) Ukraine power cut was cyber-attack [Online]. Available at: https://www.bbc.com/news/technology-38573074
42. Huang B, Cardenas AA, Baldick R (2019) Not everything is dark and gloomy: power grid protections against iot demand attacks. In: 28th USENIX security symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, pp 1115–1132. https://www.usenix.org/conference/usenixsecurity19/presentation/huang
43. Soltan S, Mittal P, Poor HV (2018) Blackiot: Iot botnet of high wattage devices can disrupt the power grid. In: 27th USENIX security symposium (USENIX security 18). USENIX Association, Baltimore, MD, pp 15–32. https://www.usenix.org/conference/usenixsecurity18/presentation/soltan

44. Kawamura R, Sato KI, Tokizawa I (1994) Self-healing atm networks based on virtual path concept. IEEE J Sel Areas Commun 12(1):120–127. https://doi.org/10.1109/49.265711
45. Murakami K, Kim H (1997) Comparative study on restoration schemes of survivable atm networks. In: Proceedings of INFOCOM '97, vol 1, pp 345–352. https://doi.org/10.1109/INFCOM.1997.635156
46. Frisanco T (1997) Optimal spare capacity design for various protection switching methods in atm networks. In: Proceedings of ICC'97—international conference on communications, vol 1, pp 293–298. https://doi.org/10.1109/ICC.1997.605267
47. Doverspike R, Wilson B (1994) Comparison of capacity efficiency of dcs network restoration routing techniques. J Netw Syst Manage 2(2):95–123
48. Murakami K, Kim H (1994) Near-optimal virtual path routing for survivable atm networks. In: Proceedings of INFOCOM '94 conference on computer communications, vol 1, pp 208–215. https://doi.org/10.1109/INFCOM.1994.337615
49. Iraschko R, MacGregor M, Grover W (1998) Optimal capacity placement for path restoration in stm or atm mesh-survivable networks. IEEE/ACM Trans Network 6(3):325–336. https://doi.org/10.1109/90.700896
50. Lin H, Chen C, Wang J, Qi J, Jin D, Kalbarczyk ZT, Iyer RK (2018) Self-healing attack-resilient pmu network for power system operation. IEEE Trans Smart Grid 9(3):1551–1565. https://doi.org/10.1109/TSG.2016.2593021
51. Dibaji SM, Pirani M, Annaswamy AM, Johansson KH, Chakrabortty A (2018) Secure control of wide-area power systems: confidentiality and integrity threats. In: 2018 IEEE conference on decision and control (CDC), pp 7269–7274 (2018). https://doi.org/10.1109/CDC.2018.8618862
52. Ni H, Rahouti M, Chakrabortty A, Xiong K, Xin Y (2018) A distributed cloud-based wide-area controller with sdn-enabled delay optimization. In: 2018 IEEE power energy society general meeting (PESGM), pp 1–5. https://doi.org/10.1109/PESGM.2018.8586040
53. Ferman M, Sparber S, Limón E (2021) 2 million Texas households without power as massive winter storm drives demand for electricity [Online]. Available at: https://www.texastribune.org/2021/02/15/rolling-blackouts-texas/
54. Golding G, Kumar A, Mertens K (2021) Cost of Texas' 2021 deep freeze justifies weatherization [Online]. Available at: https://www.dallasfed.org/research/economics/2021/0415.aspx
55. Fachkha C, Bou-Harb E, Keliris A, Memon N, Ahamad M (2017) Internet-scale probing of cps: Inference, characterization and orchestration analysis. In: Proceedings of the 2017 annual network and distributed system security symposium (NDSS 2017)
56. Schloesser M Dionaea low interaction honeypot (forked from dionaea.carnivore.it):rep/dionaea [Online]. Available at: https://github.com/rep/dionaea
57. Tamminen U (2018) Kippo: SSH honeypot. contribute to desaster/kippo development by creating an account on GitHub [Online]. Available at: https://github.com/desaster/kippo
58. Buza DI, Juhász F, Miru G, Félegyházi M, Holczer T (2014) CryPLH: Protecting smart energy systems from targeted attacks with a PLC honeypot. In: Cuellar J (ed) Smart grid security. Springer International Publishing, pp 181–192
59. Conpot: low interactive server side industrial control systems honeypot [Online]. Available at: http://conpot.org/
60. Wilhoit K, Hilt S The GasPot experiment: unexamined perils in using gas tank monitoring systems. A TrendLabs Research Paper. https://www.trendmicro.de/media/wp/the-gaspot-experiment-wp-en.pdf
61. Han W, Zhao Z, Doupé A, Ahn GJ (2016) Honeymix: toward sdn-based intelligent honeynet. In: Proceedings of the 2016 ACM international workshop on security in software defined networks & network function virtualization, SDN-NFV Security '16. Association for Computing Machinery, New York, NY, USA, pp 1–6. https://doi.org/10.1145/2876019.2876022
62. Antonatos S, Akritidis P, Markatos E, Anagnostakis K (2007) Defending against hitlist worms using network address space randomization. Comput Netw 51(12):3471–3490. https://doi.org/10.1016/j.comnet.2007.02.006

63. Jafarian JH, Al-Shaer E Duan Q (2015) Adversary-aware ip address randomization for proactive agility against sophisticated attackers. In: 2015 IEEE conference on computer communications (INFOCOM), pp 738–746. https://doi.org/10.1109/INFOCOM.2015.7218443
64. Kewley D, Fink R, Lowry J, Dean M (2001) Dynamic approaches to thwart adversary intelligence gathering. In: Proceedings DARPA information survivability conference and exposition II. DISCEX'01, vol 1. IEEE Comput Soc, pp 176–185. https://doi.org/10.1109/DISCEX.2001.932214
65. Yuill J, Denning D, Feer F (2006) Using deception to hide things from hackers: processes, principles, and techniques. J Inf Warfare 3(5):16
66. Rahman MA, Al-Shaer E, Bobba RB (2014) Moving target defense for hardening the security of the power system state estimation. In: Proceedings of the first ACM workshop on moving target defense, MTD '14. Association for Computing Machinery, New York, NY, USA, pp 59–68. https://doi.org/10.1145/2663474.2663482
67. Ali MQ, Al-Shaer E (2015) Randomization-based intrusion detection system for advanced metering infrastructure. ACM Trans Inf Syst Secur 18(2):1–30. https://doi.org/10.1145/2814936
68. Davis KR, Morrow KL, Bobba R, Heine E (2012) Power flow cyber attacks and perturbation-based defense. In: 2012 IEEE third international conference on smart grid communications (SmartGridComm), pp 342–347. https://doi.org/10.1109/SmartGridComm.2012.6486007
69. Morrow KL, Heine E, Rogers KM, Bobba RB, Overbye TJ (2012) Topology perturbation for detecting malicious data injection. In: 2012 45th Hawaii international conference on system sciences, pp 2104–2113. https://doi.org/10.1109/HICSS.2012.594
70. Weerakkody S, Mo Y, Sinopoli B (2014) Detecting integrity attacks on control systems using robust physical watermarking. In: 53rd IEEE conference on decision and control, pp 3757–3764. https://doi.org/10.1109/CDC.2014.7039974
71. Lin H, Slagell A, Kalbarczyk Z, Iyer RK (2018) Raincoat: randomization of network communication in power grid cyber infrastructure to mislead attackers. IEEE Trans Smart Grid 1–1. https://doi.org/10.1109/TSG.2018.2870362
72. Lin H, Zhuang J, Hu YC, Zhou H (2020) Defrec: establishing physical function virtualization to disrupt reconnaissance of power grids' cyber-physical infrastructures. In: Proceedings of the 2020 annual network and distributed system security symposium (NDSS 2020)
73. Shi Z, Yao W, Li Z, Zeng L, Zhao Y, Zhang R, Tang Y, Wen J (2020) .: Artificial intelligence techniques for stability analysis and control in smart grids: methodologies, applications, challenges and future directions. Appl Energy 278:115733. https://doi.org/10.1016/j.apenergy.2020.115733. URL https://www.sciencedirect.com/science/article/pii/S0306261920312228
74. Jha S, Cui S, Banerjee S, Cyriac J, Tsai T, Kalbarczyk Z, Iyer RK (2020) Ml-driven malware that targets av safety. In: 2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE Computer Society, Los Alamitos, CA, USA, pp 113–124. https://doi.org/10.1109/DSN48063.2020.00030

Uniting Computational Science with Biomedicine: The NSF Center for Computational Biotechnology and Genomic Medicine (CCBGM)

Liewei Wang and Richard M. Weinshilboum

Abstract Prof. Ravishankar Iyer (Ravi Iyer) was the driving force behind the creation of the University of Illinois Urbana-Champaign (UIUC) and Mayo Clinic joint National Science Foundation (NSF)-funded Center for Biotechnology and Genomic Medicine (CCBGM). CCBGM—led by Professor Iyer at UIUC and Professor Liewei Wang from the Mayo Clinic—has served as a "catalyst" for exchange between these two outstanding institutions and for bringing cuttingedge Artificial Intelligence and Machine Learning techniques to the "bedside" at Mayo and beyond in the broader biomedical community. As a result, CCBGM has become a model for value of inter-institutional and cross-disciplinary collaboration.

Keywords Uniting computational and biomedical science · University of Illinois Urban-Campaign and the Mayo Clinic · NSF CCBGM · Major Depressive Disorder Predictive Algorithm

Prof. Ravishankar Iyer (Ravi Iyer) was the driving force behind the creation of the University of Illinois Urbana-Champaign (UIUC) and Mayo Clinic joint National Science Foundation (NSF)-funded Center for Biotechnology and Genomic Medicine (CCBGM). CCBGM has become a major focus of a long-standing relationship between UIUC and Mayo as well as a catalyst for uniting modern analytical techniques such as Artificial Intelligence (AI) and Machine Learning with medical practice at Mayo and with the science based at both institutions. If that description makes Dr. Iyer sound a bit like a "force of nature", the authors of this tribute could not agree more that he was not merely "present at the creation" of this important NSF Center, but rather that he—in partnership with one of us, Dr. Wang—conceived of, articulated the vision for CCBGM and drove that vision to reality. The outcome has served as a positive stimulus to both institutions, UIUC and the Mayo Clinic, as well

L. Wang (✉) · R. M. Weinshilboum
Department of Molecular Pharmacology and Experimental Therapeutics, Mayo Clinic, Rochester, MN, USA
e-mail: wang.liewei@mayo.edu

R. M. Weinshilboum
e-mail: weinshilboum.richard@mayo.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_17

as an example of the tremendous potential of the union of these two great Midwestern institutions—one, UIUC, with a tradition of educational excellence, with a special focus on Engineering and Computer Science, and the other, the Mayo Clinic, with an equally strong and acknowledged reputation for world class excellence in medical care and medical science. Even though the formal alliance between Mayo and UIUC had been forged over a decade before CCBGM, it can be argued that the creation of CCBGM moved the original vision for that alliance forward in a striking fashion and succeeded in helping to bring the original vision behind the alliance to reality with a series of joint projects, with the exchange of students and fellows between the two institutions and—finally—with some of those students pursuing careers at the one or the other of these two world-renowned institutions.

Institutional bridges between the Mayo Clinic and UIUC were led at Mayo by the Center for Individualized Medicine (CIM), with the original groundwork for this partnership being laid by the first Mayo CIM Director, Dr. Franklyn Prendergast M.D.-Ph.D. Initially, this promising institutional partnership grew rather slowly but it was already making steady progress when Ravi came on the scene. For example, the authors of this chapter were Mayo Clinic Co-PIs for an NIH National Institute of General Medical Sciences (NIGMS)-funded U54 grant supporting the "KnowEng" biomedical and genomics database prior to the creation of CCBGM, a grant with participants from both UIUC and Mayo. However, Ravi's vision for CCBGM involved the creation of an effort catalyzed by the NSF but funded heavily by industry—industry that initially ranged from computational science to the pharmaceutical industry to agricultural seed companies, all of which were committed to the application of modern computation to medicine and biology. The initial academic partners within CCBGM were UIUC, the Mayo Clinic and the University of Chicago. None of those involved will ever forget the original organizational meeting held in Chicago that involved representatives of the three academic institutions, possible industry partners and the NSF. The force which brought these apparently disparate organizations together was and has remained Ravi—with his drive, energy, vision and eternal optimism. To everyone's delight, a series of joint projects was agreed on during that initial meeting—selected by the industry-based participants—with subsequent meetings held, on a rotating basis, in Chicago, in Rochester, Minnesota and in Champaign-Urbana. What no one could have predicted was the power of the vision that Dr. Iyer articulated or the importance of the youthful energy supplied by the computer science, engineering and biomedical science students who made these projects successful.

The secret of CCBGM which resulted in it becoming a significant factor in a transformative wave that is presently surging through biomedicine—not merely at the Mayo Clinic but across the world—was the power of bringing talented and creative young people from biomedicine together with equally creative and talented young engineers and computer scientist and their mutual recognition of what they might accomplish by working together. It is doubtful that the organizational leadership of either of the two lead institutions (the University of Chicago left the partnership early on when their lead participant accepted a position with industry) fully appreciated the revolutionary power of Dr. Iyer's creation. One example of the power and the

consequences of the CCBGM vision is provided Dr. Arjun Athreya M.S., Ph.D.—one of Ravi's students and one of the editors in this tribute to Dr. Iyer. Dr. Athreya, then a graduate student at UIUC, spent his summers and many holidays in Rochester working initially with Dr. Liewei Wang on a successful breast cancer cell line-based genomics project involving the therapy of a specific subtype of breast cancer. During that time, he became familiar with the projects of graduate students in Dr. Weinshilboum's laboratory located "next door" who were using genomics to study the response of patients suffering from Major Depressive Disorder (MDD), depression, the number one psychiatric disease world-wide, to drug therapy. MDD patients are most often treated with Selective Serotonin Reuptake Inhibitors (SSRIs), drugs that work, but only in about half to two thirds of MDD patients. By using data from large clinical trials led by Drs. Weinshilboum and Wang together with their psychiatry colleagues at Mayo as well as genomic information from SSRI clinical trials from across the world, Dr. Athreya and his clinical psychiatry colleague, Dr. William Bobo, developed an AI and Machine Learning-based algorithm that incorporated clinical data together with genomic data and which could increase the accuracy for predicting which patients might benefit from SSRI therapy from approximately 55% to 85–90%—a value that has significant clinical utility. It requires at least two months to clinically "test" the efficacy of SSRIs, an unacceptable wait time for a potentially suicidal patient. If the algorithm could provide an accurate prediction of outcome with no waiting time, that would represent a remarkable step forward in the treatment of this disease. The preceding sentences may appear to make an analytical process that required years to achieve sound simple. It was not simple, but it was successful! That MDD SSRI response algorithm is now being implemented for application to depressed patients at Mayo Clinic and—after US Food and Drug Administration (FDA) approval—to help patients suffering from MDD outside of Mayo. This will be the first time that the Mayo Clinic has taken this type of algorithm to the FDA for review and approval. This one brief but striking vignette serves to illustrate just how significant the impact of the incorporation of AI and Machine Learning across a large biomedical center could be potentially—and neither this example nor many others would have been possible without the structure provided by the CCBGM.

Stepping back briefly from CCBGM, a series of vignettes like that described above addressing the drug therapy of a psychiatric disease have resulted in the formation of an AI Analytical Subcommittee by the Mayo Clinic Center for Individualized Medicine—with Dr. Liewei Wang, one of the authors of this brief tribute—as Co-Chair. Prior to the creation of the CCBGM, that would have not been conceivable, much less possible. The fact that the formation of such a committee now seems an obvious step illustrates just how far ahead of his peers Dr. Iyer was and how correct his vision of the future of biomedicine and the necessity for the union of modern computational science with medicine was. Both the Mayo Clinic and UIUC owe a debt of gratitude to Dr. Ravi Iyer for his vision, his drive and his ability to bring diverse disciplines together to achieve a common goal, a goal that—in this case—will serve to benefit innumerable patients across the nation and, eventually, across the world.

Data-Driven Approaches to Selecting Samples for Training Neural Networks

Murthy V. Devarakonda

Abstract Modern neural networks, that are now commonly used for most natural language processing (NLP) tasks, contain many hidden units and parameters. There is a considerable interest in developing strategies for selecting an optimal set of samples to train such large models for biomedical tasks because developing training data is expensive and time consuming in the biomedical space. Lack of sufficient training data is exacerbated by the fact that the ratio of negative samples to positive samples is also highly skewed, i.e., too many negative samples but too few positive samples. Therefore, an important problem, especially for the biomedical space, what is the optimum set of negative samples to use in creating an effective and balanced training data sample. Interestingly though, the insights which may help to decide the most effective sample selection can be found in the data itself (i.e., in the samples themselves). This chapter briefly reviews traditional approaches to selecting training samples and then presents the latest data-driven approaches for selecting samples to effectively train modern neural networks.

1 Introduction

In training machine or deep learning models for natural language processing (NLP), selecting the right set of training samples can be critical, especially in biomedical NLP, where there are never enough training samples, and often negative samples far exceed positive samples in a data set [1]. Consider, for example, the task of classifying scientific papers into those proposing treatments for Covid-19 and the rest. Even when limited to the year 2020, out of roughly 1 million articles indexed in PubMed less than 1% articles are related to Covid-19. For the task then, the ratio of potential positive to negative samples is 1:99. There are many ways one may select negative samples to match positive samples (from a uniformly sampled and labeled subset of the articles), if we wish to train our model using an equal number of positive and negative samples (which leads to the most balanced predictions). So, the

M. V. Devarakonda (✉)
AI Innovation Lab, Novartis, Cambridge, USA
e-mail: mvd@acm.org

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_18

question is: What is the best subset of negative samples that should be used to best train a model? More generally, how does negative sample selection effect training of a neural model?

The standard approach to selecting negative samples in unbalanced data is to down sample negatives using random selection. However, research in Active Learning [2] suggests potential benefits of proactive selection of samples rather than random selection. In Active Learning, samples are labeled one (or a few) at a time and which sample(s) should be labeled next is selected (by the Active Learning system) using a "query" selection strategy. Active Learning starts with the assumption that there are a few or no labeled data, and using various strategies coupled with the model being trained, the system proposes next set of samples to be labeled by the human annotators. Uncertainty sampling is a widely used principle for the next sample selection. Settles [3, 4] discussed a broad set of selection strategies, such as selecting least confident, confident margin, and entropy-based samples, based on the uncertainty principle. Research shows these strategies can train a model as good as the full training set, with just 20–40% of training samples.

However, samples selected with strategies tied to performance of a model may not work well for a different model. Note that the uncertainty as discussed above is tied to its calculation using a specific model and two different models may calculate two widely different uncertainties. So, the uncertainty based strategies are tied to the model being developed and not necessarily or directly, to the intrinsic characteristics of the data itself. Researchers therefore have proposed data intrinsic characteristics for sample selection. Some such selection strategies are based on diversity and density of samples [5]. Others proposed approaches that involve similarity or dis-similarity with already-labeled samples. These data-driven approaches are particularly interesting because they are agnostic to the model being trained and therefore have broader applicability. In this chapter we will explore a novel data-driven approach, called near-miss sampling, as a case study in sample selection for neural network training.

2 Case Study: Near-Miss Sampling for Optimum Model Training

To recap, in training machine and deep learning models, intentional, data-driven selection of training samples is an interesting area of research since it has the potential to leverage data characteristics to improve accuracy and robustness of the model being trained. This is especially true in biomedical tasks where negative samples tend to significantly outnumber positive samples. To make this concrete, let us consider a conceptual scenario illustrated in Fig. 1a, which is a two-dimensional visualization of samples in latent space. Solid red dots indicate positive samples and unfilled blue dots represent potential negative samples. The ratio of positive samples to negative samples is 1:7 in this example, however, in practice it can be as large as 1:14 or even higher. The goal is to select negative samples equal in number to positive samples so

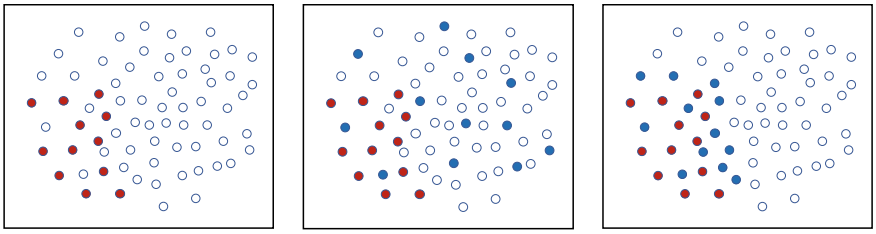


Fig. 1 For balanced model prediction, 1:1 positive and negative training samples are needed. Which samples to select when there is an imbalance (**a**)? The standard approach is random down sampling of negatives (**b**). Here we explore the impact of "near-miss" sampling (**c**)

that the model is trained to predict labels without a bias towards any one class (i.e., does not necessarily favor one label or the other). In Fig. 1b we show a selection of negative samples, which is a result of random sampling from potential negative samples. Alternatively, as shown in Fig. 1c, we may choose negative samples based on the "near-miss" principle.

2.1 The Near-Miss Principle

What is a near-miss? A near miss [6] is a negative example that differs from the concept being learned in only a small number of significant points. In the psychology of game playing [7] and in image recognition [8], it was observed that the near misses have distinct positive effect on the outcome.

Often the near miss principle is explained using the classic example of learning what is an "Arch" as shown in Fig. 2. The illustrations in the top row represent correct examples of the structure of an arch, thus they form the positive examples in the context of training a model. The illustrations in the bottom row represent structures that are not considered as arches, i.e., negative examples. The key point is that the structures in the bottom row differ from the top-left structure in only a small but significant difference—near-misses. The bottom right structure has no space between vertical columns otherwise it would be a "good" example of arch. We describe a use case here where this interesting principle was applied as a strategy for selecting negative samples and its impact on supervised learning models of a corresponding biomedical NLP task.

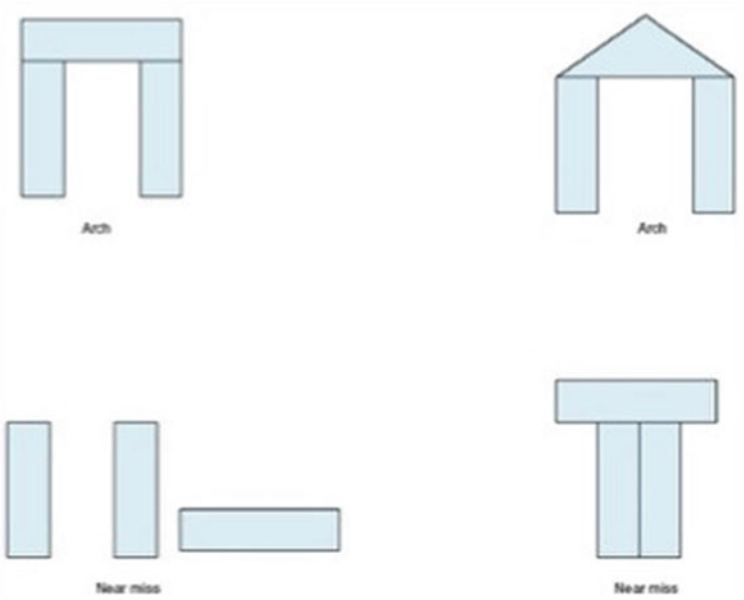


Fig. 2 The near miss concept explained through an example. Near-miss differs from the concept being learned in only a small number of significant points

2.2 *The Biomedical NLP Task: Adverse Drug Event and Indication Relation Extraction*

Let us consider the task of determining if an *adverse drug event* (ADE) or an *indication* relation (or no relation) was expressed in a text segment between a pair of drug and medical problem mentions. An *ADE* is a harmful medical effect from using a drug, and an *indication* is the use of a drug for treating a particular condition. An example of an adverse event relation is shown in Fig. 3. In the text segment,

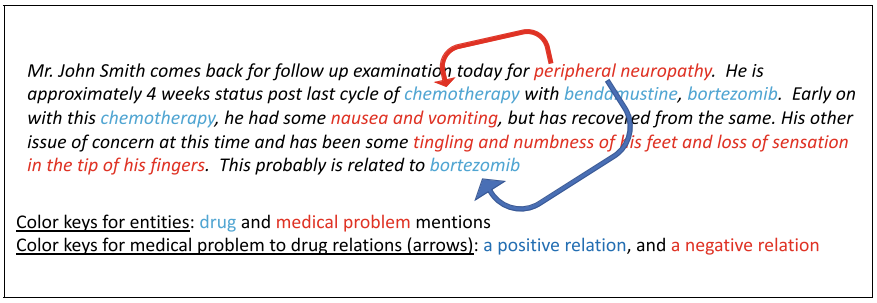


Fig. 3 Adverse drug event (ADE) relation samples: positive and negative

two entities *bortezomib* and *peripheral neuropathy* are the mentions of drug and medical problem entities respectively. The NLP task is to determine if a text segment expresses or implies an ADE between them in the text.

In Fig. 3, the relation between the specific mentions of *bortezomib* and *peripheral neuropathy* is a positive example, i.e., the text indeed expresses or implies an ADE relation between the *peripheral neuropathy* and *bortezomib* mentions. There are several other drug mentions in the text in Fig. 3, i.e., two mentions of *chemotherapy*, *bendamustine,* and a second mention of *bortezomib.* There are also other medical problem mentions (e.g., *nausea and vomiting*) in the text. Any drug mention can be paired with any medical problem mention to form a negative example for model training purposes so long as the pair is not in positive samples. In fact, even any pair of drug and medical problem mentions from any text segment can be used as a negative example so long as the pair is not in positive sample. The problem of negative sample selection comes down to which one of these potential negative samples are to be used in training the relation extraction model.

Two recent biomedical NLP Challenges concerning ADE and indication relation extraction from clinical notes provided an opportunity to explore this sample selection methodology: Medication and Adverse Drug Events from Electronic Health Records 1.0 (MADE) [9] and National NLP Clinical Challenges Task 2 (N2C2) [10]. The example shown in Fig. 3 is drawn from the MADE dataset. The best performing systems in the Challenges achieved mid to high 0.7 F measures on the ADE and indication relations. There is room for improving performance improvement. Before we further dive into experiments involving near-miss sampling, let us understand more about the datasets employed in the experiments.

2.3 Datasets

The MADE dataset contained 1092 de-identified clinical notes of 21 cancer patients. Each clinical note was annotated with drug names, medical problems, and other entities, and relations among the entities. It should be noted that the medical problems were further subcategorized as adverse drug event entities, reason (indication) entities, and sign or symptoms (SSLIFs), although these details are less relevant to the topic of discussion here. We will focus only on ADE and indication relations. The data was split into a training set of 900 notes and a test set of 180 notes.

The N2C2 dataset consisted of 505 discharge summaries from the MIMIC-II clinical care database. Each note was also annotated with drug names, medical problems, and other entities, and relations among them. Just as in MADE, we will focus only on ADE and indication relations. This dataset was split into a training set of 303 notes and a test set of 202 notes.

The characteristics of the two datasets are shown in Tables 1 and 2, which include the number of positive relations and potential negative relations in the training and test sets broken down by relation types. The Tables name ADE and indication relations as ADE-Drug and Reason-Drug and include other relation types that are not of interest to

Table 1 The MADE dataset statistics

Relation type	Training dataset			Test dataset		
	Positive samples	Possible negative samples	Neg. to pos. ratio	Positive samples	Possible negative samples	Neg. to pos. ratio
ADE-drug	2057	27,288	**13.3**	512	7485	**14.6**
Reason-drug	4530	54,522	**12.0**	871	9821	**11.3**
Total	6587	81,810	**12.4**	1383	17,306	**12.5**

The bold lettering is used to signify the importance of the negative to positive sample ratios, which are large in these data sets

Table 2 The N2C2 dataset statistics

Relation type	Training dataset			Test dataset		
	Positive samples	Possible negative samples	Neg. to pos. ratio	Positive samples	Possible negative samples	Neg. to pos. ratio
ADE-drug	1061	4430	**4.2**	724	2912	**4.0**
Reason-drug	4991	29,751	**6.0**	3392	20,029	**5.9**
Total	6052	34,181	**5.6**	4116	22,941	**5.6**

The bold lettering is used to signify the importance of the negative to positive sample ratios, which are large in these data sets

the present discussion. The ratios of potential negative relations to positive relations are also shown in the tables. The ratios are skewed in both datasets, i.e., substantially more potential negative relations than the positive relations, but the ratios are even higher in the MADE training dataset. Specifically, the ratio is 13.3 for the ADE-Drug relation, 12.0 for Reason-Drug, and 12.4 for the total of two relations in the MADE training dataset. The corresponding ratios are 4.2, 6.0, and 5.6 for the N2C2 training dataset. Similar ratios can be seen in the test datasets as well. These skewed ratios indicate an opportunity to strategically select negative samples to optimize training. The goal is to select equal number of negative and positive samples so that the model prediction is balanced across output classes.

2.4 Near-Miss Sampling Applied to the Task

We translated the near-miss principle to text by ensuring that a negative sample shares largest common text (and hence context) with a positive sample. The concept and how to select near-miss samples is shown in Fig. 4.

For each entity of a gold standard relation, the immediately preceding and following entities of the same type as the gold standard entity are identified in the text

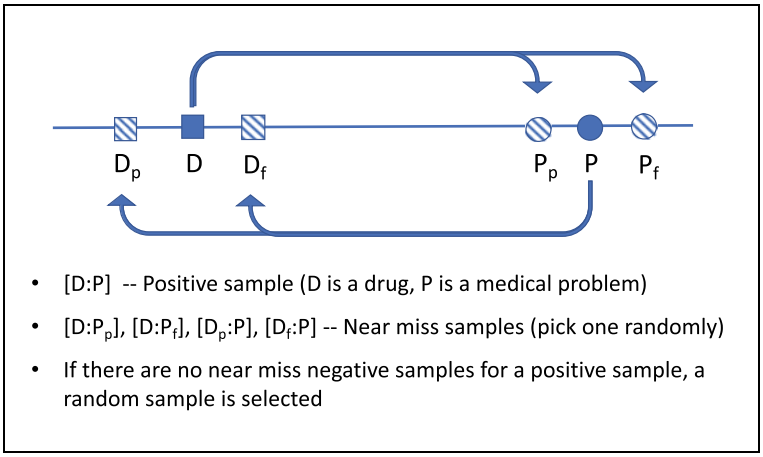


Fig. 4 The near-miss sampling algorithm illustrated

segment under consideration. In Fig. 4, D and P indicate *drug* and *medical problem* entities from a gold standard (true) ADE relation. That is, an ADE relation exists between D and P entities in the text segment. Dp and Df are the nearest preceding and following drug entities of D, and similarly Pp and Pf are the nearest preceding and following medical problem entities of P.

From these four nearest entities and from the original two entities, valid potential negative relation samples are formed by pairing between one original entity and one of the nearest entities of the matching entity types. Valid means the relation meets its entity type requirements and the entity pair occurs in a sequence of words in the document. In a typical scenario at most four such relations can be formed. In Fig. 4, these relations are indicated by the four pairs: [D:Pp], [D:Pf], [Dp:P], and [Df:P].

Note that, we may find four or less such relations in each text segment depending on the occurrence of entities in the text segment. If any of these relations were already in the gold standard or were picked already as a negative sample, they were removed from the list. Among the remaining relations, the near-miss sampling randomly picks one. The most salient point is that these negative relations share most context (i.e., surrounding text) with the gold standard positive relation, meeting the definition of a near-miss sample.

If a near-miss sample cannot be found, a random negative sample (that was not already picked) was selected from the rest of possible negative samples. Negative example selection takes place only in the training phase. During the evaluation and validation phases, all potential relations were assessed by the model and the outcome was measured accordingly.

Due to practical limitations, our model enforces a maximum sequence length, which is heuristically determined during the validation phase. In training, positive samples longer than the max length are ignored, and similarly negative samples longer than the max length are also ignored. In the evaluation phase also, the

longer sequences were ignored, and the system was penalized for it in performance calculations.

2.5 *Model Trained (BERT—Bidirectional Encoder Representations from Transformers)*

We used the original BERT (Bidirectional Encoder Representations from Transformers) [11] in our study as it was the best performing deep learning model for NLP at the time this research was conducted. While several variations and enhancements of BERT have been since developed, each showing incremental improvement, the original BERT remains as a robust neural network. It achieved significant performance improvement on various general domain NLP tasks, including sentence classification which is relevant to us here. BERT is a pre-trained model that produces sequence (e.g., sentence) and word level representations, which can be fine-tuned for task-specific outcomes such as relation classification and concept extraction. Only a simple feed-forward network with a softmax layer is needed to process the BERT output for task-specific objectives.

As shown in Fig. 5a, BERT uses layers of neural network components known as Transformer encoders (*Tm*) to generate representations of input sequences in the output. Each BERT layer processes its input sequence in the forward and backward directions *simultaneously*, using a novel pre-training objective known as the masked learning model (explained later). The BERT Transformer encoder contains

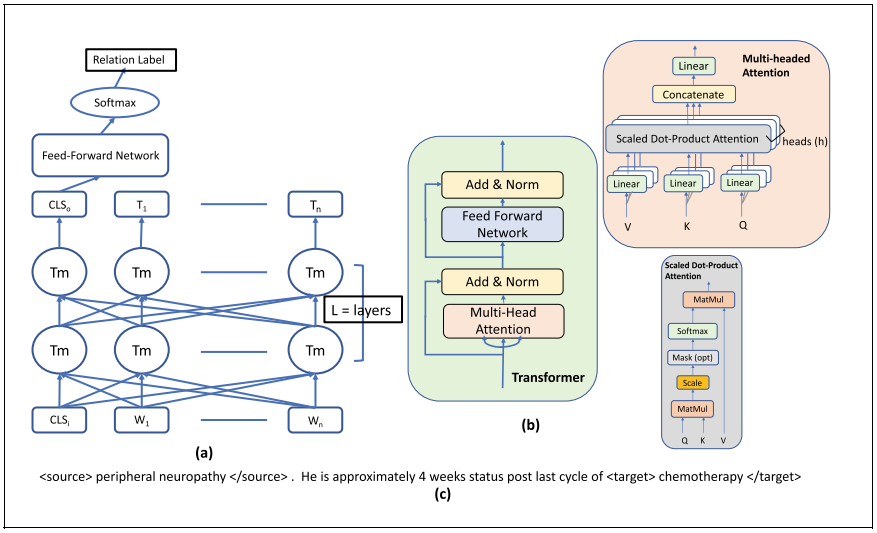


Fig. 5 Model used in our experiments, BERT (bidirectional encoder representations from transformers) (**a**) and (**b**), and the format of the input text to the model (**c**)

two sublayers (see Fig. 5b), the first sublayer is a multi-head self-attention [12] mechanism that allows modeling of the context for each word position, and the second is a feed-forward network that provides non-linear activation. Fundamentally, an attention layer produces representation for each input token that is based on any arbitrary input tokens, by comparing each input token with all other input tokens (self-attention), and producing a series of probability distributions to assign importance of input tokens. Multi-headed attention can simultaneously optimize for different combinations of input tokens.

As mentioned earlier, BERT uses a pre-training objective known as the masked learning model (MLM), [13] where some random words in the input are masked, and the pre-training objective is to predict the original word based on the context. In other approaches, typically next word prediction was used, which limited a multi-layered model to process the input either in the forward direction only or process in the forward and backward directions separately and then aggregate the representations [14]—both these approaches fail to leverage the forward and backward contexts at the same time. The use of MLM was a key invention that enabled simultaneous bidirectional input processing in a multi-layer model, without allowing words-to-be-predicted appearing in the input of an upper layer.

The BERT model we used here came pre-trained with BookCorpus [15] and English Wikipedia (general domain corpus). Corpus words were tokenized using the WordPiece dictionary [16] of 30,000 words and as needed words were split into pieces using ## (two hash marks). Word piece representations using biomedical corpus were not readily available at the time of our study, however, recently a BERT model was pre-trained on a biomedical corpus [17] and it was shown to improve entity extraction [18].

We employed BERT in its base configuration of 12 layers (=24 sublayers), 768 hidden size, 12 self-attention heads. In this configuration, BERT produced representations for each token in the input as well as for the entire sequence (shown as CLSo). This sequence classification representation from the top layer of BERT (see Fig. 5a) was used as the input to a fully connected feed-forward layer. A softmax layer provided the final relation classification label for the pair of entities in the input sequence.

In our method, as shown in Fig. 5c, the input to BERT was a sequence of words that started with the first entity of a relation and ended with the second entity of the relation. As previous studies demonstrated, [19] this is a convenient choice since sentence segmentation of clinical notes text is error prone due to embedded lists and tables which are not well handled by the standard NLP code such as NLTK [20]. We were also constrained by our BERT model input limit which was 512 tokens (word-pieces). Entity spans were further marked using the entity tags. Alternatively, position encoding can be used to achieve a similar result.

In the training (fine-tuning) stage, the input sequences included all positive samples in the gold standard and an equal number of negative samples, that were either randomly down sampled from all possible negative samples or near-miss sampled as described earlier. Only the sequences that were less than or equal to

a heuristically determined (in the validation stage) maximum sequence length were used in training.

2.6 Experiments and Metrics

We fine-tuned our BERT model separately for MADE and N2C2, each with random down-sampling and near-miss sampling separately. We therefore fine-tuned the model four different times and tested each one. We mostly adopted the default settings of BERT hyperparameters—i.e., training batch size of 16, 10 epochs, and a learning rate of 2e-5. However, we experimentally determined the optimum sequence length using validation sets—20% of clinical documents from the N2C2 training set and 10% of clinical documents from the MADE training set. The fine-tuned models were tested on the full MADE and N2C2 test datasets.

For each of the four experiments, we calculated standard Recall, Precision, and F measures individually for ADE and indication relations as well as for both combined. We compared our results with the published results of the systems that performed best in the task of relation extraction given gold entity labels from the two challenges: the University of Utah system [21] for MADE (denoted as the MADE-Best), and the UTHealth developed system [22, 23] for N2C2 (denoted as the N2C2-Best). Since the published results did not aggregate performance for the ADE and Reason relations, we obtained their weighted average, weighted by the number of relations evaluated in the test phase, from individual relation results.

We used two metrics for comparison: (1) Absolute F measure difference; and (2) Error rate reduction in the F measure achieved by model Y compared to model X, which is calculated as:

$$\text{error rate reduction} = \frac{F_y - F_x}{1 - F_x}$$

where F_y and F_x are the F measures of models Y and X respectively. While the absolute F measure difference shows the net improvement in the measure, the error rate reduction is a sound relative measure that shows reduction in the remaining performance gap of the previous model. It is usually expressed as a percentage. Recent studies in the general domain NLP have adapted this metric for effective comparison [11, 24].

We suspected that when the distance (words) between the entities is short, the potential to leverage contextual information would be rather limited. The two datasets are also developed from different clinical documents—N2C2 contains discharge summaries whereas MADE contains clinical notes. In order to quantify such differences, we studied the sequence length distributions of positive samples, near-miss samples, and all negative samples for each dataset and plotted their cumulative distribution frequency.

2.7 Results

Table 3 shows performance evaluation on the MADE dataset. Precision, recall, and F measures were shown for BERT with random negative instances sampling (shown as BERT+Rand) and BERT with the near-miss sampling (shown as BERT+NM) for each relation type and for the aggregate of ADE and Reason relations. The table also shows absolute F measure differences between MADE-Best and BERT+Rand, BERT+Rand and BERT+NM, and between MADE-Best and BERT+NM. Percentage error rate reduction was shown between MADE-Best and BERT+NM.

BERT+Rand achieved 0.6 and 6.6% performance improvement for the ADE and Reason relations, and 4.1% for the aggregate. Near-miss improved F measure by additional 3.3 and 0.9% for the relations, and by 1.9% for the aggregate. Performance improvement of BERT+NM over MADE-Best was substantial: F measure improved by 3.9 and 7.5% for ADE and Reason respectively, and by 6.4% for both together. The error rate reduction was substantial—14.5% for ADE, 31.0% for Reason, and 23.6% for both together.

Table 4 shows performance evaluation on the N2C2 dataset in the same way as in Table 3. The general trend of the results is like that of the MADE results. BERT+Rand improved the F measure of ADE and Reason relations by 1.5% and 6.8% respectively, for an aggregate of 6.0%. BERT+NM improved performance of ADE and Reason relations further by 1.3% and 0.6% respectively, for an aggregate of 0.7%. The overall improvement over N2C2_Best was 2.8% and 7.4% for ADE and Reason relations, and 6.7% for the aggregate. The corresponding error rate reductions were 13.7, 30.6, and 8.3%. Precision and recall details were publicly unavailable for the N2C2_Best at the time of writing this paper.

Statistical significance test: Previous studies [25] have used the McNamara test (and is generally accepted as a good test) for determining the statistical significance of F measure improvement of an NLP task. The test requires the contingency (confusion) table from the performance study. Using the data in our study, we determined that the F measure improvement with the near-miss sampling was statistically significant at $p < 0.001$ for the MADE dataset for the combined ADE and Reason relations. For the N2C2 dataset, the improvement was significant at $p < 0.03$ for the combined ADE and Reason relations. We could not determine statistical significance of performance improvements relative to the MADE-Best and N2C2_Best models because the contingency tables for them are not publicly available at the time of this article.

Another important observation from Tables 3 and 4 is that the near-miss sampling consistently improved precision, while often losing ground on recall. Near-miss improved precision for the overall and ADE+Reason by 4.1% and 5.7% for the MADE dataset, and by 1.2% and 2.3% for the N2C2 dataset respectively. Recall reduced by small percentages across the broad. These results indicate an important characteristic of the near-miss sampling approach. We also note that both BERT+Rand and BERT+NM consistently improved recall over MADE_Best.

Table 3 Results for the MADE dataset (Rand = random down sampling; NM = Near-miss sampling)

Relation type	MADE_Best (P/R/F)	BERT+Rand (P/R/F)	BERT+Rand over MADE_Best: ΔF measure	BERT+NM (P/R/F)	BERT+NM over BERT+Rand: ΔF measure	BERT+NM over MADE_Best	
						ΔF measure	Err. reduced (%)
ADE-drug	0.787/0.683/0.731	0.652/0.848/0.737	0.006	0.730/0.814/0.770	0.033	0.039	14.5
Reason-drug	0.780/0.739/0.758	0.728/0.948/0.824	0.066	0.772/0.904/0.833	0.009	0.075	31.0
ADE+reason	0.783/0.712/0.746	0.700/0.911/0.791	0.041	0.757/0.871/0.810	0.019	0.064	23.6

Table 4 Results for the N2C2 dataset (Rand = random down sampling; NM = Near-miss sampling)

Relation type	N2C2_Best F measure	BERT +Rand (P/R/F)	BERT+Rand over N2C2_Best: ΔF measure	BERT+NM (P/R/F)	BERT+NM over BERT+Rand: ΔF measure	BERT+NM over N2C2_Best	
						ΔF measure	Err. reduced (%)
ADE-drug	0.795	0.795/0.825/0.810	0.015	0.818/0.828/0.823	0.013	0.028	13.7
Reason-drug	0.758	0.784/0.872/0.826	0.068	0.807/0.858/0.832	0.006	0.074	30.6
ADE+reason	0.763	0.786/0.864/0.823	0.060	0.809/0.853/0.830	0.007	0.067	28.3

The sequence length distributions of the positive, all negative, and near-miss-sampled ADE and Reason relations in the training datasets of MADE and N2C2 are shown in Fig. 6. We showed detailed statistics of the distributions in Table 5.

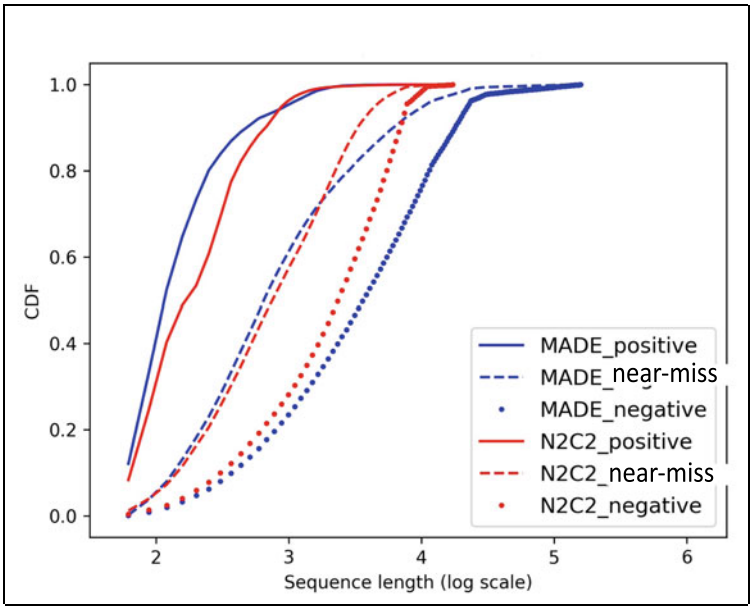


Fig. 6 Cumulative distribution frequencies (CDF) of relation sequence lengths of the ADE and reason relations

Table 5 Sequence lengths statistics for the training data

Samples	Statistic	MADE	N2C2
Positive samples	Mean	22.7	17.6
	Std. dev.	26.7	12.1
	Median	**16**	**14**
	Max	501	99
All negative samples	Mean	190.5	52.0
	Std. dev.	139.9	25.6
	Median	**155**	**51**
	Max	511	99
Near-Miss samples	Mean	98.0	37.5
	Std. dev.	107.8	23.8
	Median	**55**	**32**
	Max	511	99

The all-negative sample relation lengths in MADE (blue, dotted line) are significantly longer than in N2C2 (red, dotted line) for the ADE and Reason relations. The median length of the relations in MADE is 3.04 times the median length of N2C2 negative relations, i.e., 155 versus 51 (Table 5).

While some differences exist between the two datasets, the ADE and Reason relations are consistently long, and therefore, these sequences are likely to contain more context, which can be leveraged by BERT and the near-miss sampling. The Figure and the Table also show that the near-miss sampling, reduces the negative sample lengths. For example, the median lengths for ADE/Reason are 55 and 32 for the two datasets, which are substantially smaller than the median lengths of all negative samples (i.e., 155 and 51).

3 Discussion

The methods we used here, BERT and the near-miss sampling, better leverage contextual information to improve relation extraction compared to the top performing systems from the shared challenges, MADE and N2C2. BERT even with random down sampling makes better use of context in two ways:

Multiple layers of Transformer encoders based on the Attention model, rather than the RNN, LSTM, CNN, or combination models thereof, was shown to better leverage context.

Using the Masked Learning Model that randomly masks a word in a sequence for conditioning word representations to simultaneously analyze the input bidirectionally, rather than predicting merely an association of words (as in word2vec), or the next words (as in OpenAI GPT), or even a concatenation of representations predicting next and preceding words (as in ELMo).

In addition, BERT only required fine-tuning of a pre-trained model rather than complex task-specific neural networks that use word representations as features thus achieving transfer learning from a large text corpus. For these reasons, BERT in our study performed better than the top performing systems from the challenges. It is interesting to note that the MADE_Best employed a carefully feature-engineered Random Forest and N2C2_Best used complex composition of neural networks consisting of LSTMs and CNNs, and BERT improved upon both approaches.

Selection of negative samples in model training is known to be a challenge, [26–28] especially when relation entities are multiple sentences apart which gives rise to a very large number of potential negative samples. Most previous studies simply down sampled the larger population, but one study [19] considered a feature-engineered Alternating Decision Tree machine learning model for selecting candidate samples, both for training and testing. The near-miss sampling, by preferring negative training samples that share significant context with positive samples, provides a simpler and in combination with BERT higher accuracy compared to the previous candidate selection approach on the datasets.

In can be observed from the results that the performance improvement achievable from contextual information depends on the length of context between relation entities. Quantitatively, the relation distance differences in the datasets can be seen in Fig. 6 and Table 5. Qualitatively, as can be seen from the example in Fig. 3, drug and medical problems entities are likely to appear further apart in complex discourse in a clinical document. These typically longer relations offer an opportunity for the sophisticated neural architecture of BERT to create better representations, and hence increase F measure accuracy substantially. It should however be noted that the error rate reduction is significant across most relations with BERT plus near-miss.

Figure 6 and Table 5 also show that the near-miss sampling, not only takes advantage of the contextual information (by definition) but also reduce the negative sample lengths and make their lengths similar to the positive samples. As we noted earlier, the near-miss sampling disproportionately improves precision over recall, because the model learns to distinguish closely related samples from positive and negative classes.

Important contributions of the case study were as follows:

1. We showed that the BERT model, which leverages bidirectional contextual information with multi-layer Transformers, requiring only fine-tuning can provide excellent performance in biomedical relation extraction without complicated, task-specific neural network designs containing RNSs, LSTMs, and CNNs.
2. We showed that the "near-miss" based the near-miss sampling of negative instances, rather than random selection, can improve training and therefore model performance especially when there is a large population of potential negative samples to choose from and when relation sequences are long enough to form "near-misses" from positive samples.
3. Both BERT and near-miss use contextual information in long distance relations to achieve significant performance improvement. The performance improvement, compared to the previous methods, was substantial for such relations (i.e., ADE and Reason relation): 6.4% absolute F measure improvement (23.6% error reduction) for the MADE dataset and 6.7% absolute F measure improvement (28.3% error reduction) for the N2C2 dataset.

4 Learnings from the Case Study and Future Work

The case study above, while showing the importance of context in reasoning about data and how BERT exploits some of the context using the attention models, also showed the importance of optimally selecting training samples (here, negative samples, specifically). For a balanced binary predictor (i.e., a model that selects positives and negative cases with equal accuracy), the number of positive and negative samples should be (roughly) equal in number. In the case study, we had an abundance of negative samples but had a smaller set of positive samples, and therefore we had the opportunity to experiment with sample selection for negative samples.

When near-miss negative samples were used in training, the model achieved better precision compared to when it was trained on random negative samples (the same BERT pre-trained was used in both cases). We conclude that the near- miss samples trained the model to distinguish the small but critical differences between the positive and negative cases, and hence reduced false positives on the test cases. The near-miss negative samples trained the model to be more "precise".

Some of the modern machine learning techniques tacitly incorporate the near-miss principle. Adversarial and Contrastive learning techniques, by definition, attempt to train models with near-miss samples (they may start out with random samples but strive to generate near-miss samples eventually). They do so by generating artificial samples rather than by analyzing the data, of course. Success of these approaches suggests the power of near-miss samples in effectively training models.

The difference between the data-driven approach, exemplified by the near-miss strategy, and the model-driven approaches, traditionally used in Active Learning, is worth noting. The basic idea in Active Learning is to build a model with a small training dataset and obtain additional, incremental training data based on an analysis of the predictions of the model trained so far, and repeat. The most general strategy for selecting additional samples for training is based on uncertainty (using the principles of Shannon's entropy, for example). However, in practice the uncertainty is calculated using the model trained so far and therefore the calculated uncertainty is as much a characteristic of the model as of the data (and in fact, it is difficult to tease out the different influences of the data and the model). It should be pointed out that some recent studies used distribution and diversity of the data, as well as model predicted entropy, in selecting additional samples to be annotated for training. These studies in effect have recognized the need for data-driven sample selection and demonstrate the opportunity to incorporate data-driven sampling in Active Learning.

A generalized research challenge in using the near-miss principle, is the strategy or the algorithm needed to determine near-miss samples among negative samples. That is, how do you apply the principle of near-miss to a specific task? In the case study discussed here, our strategy was to use a positive sample passage for a relation being classified and to form a near-miss sample, by selecting entities that are different from the corresponding entities of the positive sample. It worked well, especially when the sample passages were long (i.e., had multiple sentences). Similarly, in a multiple-choice Question Answering task, using the wrong choice from the two closest answers would create a near-miss dataset. In an image recognition task of, say, identifying images of cats, images of other animals that are similar looking in size and shape to cats might be a good strategy for identifying near-misses. However, in general, the task of defining what is an effective and yet automated method of determining near-misses is obvious a subject of future research.

Certain initial processing of the data might help to develop strategies for effective sampling. For example, generating embeddings for text passages and testing for their similarity using pre-trained models can work for a text classification task. We believe that the data-driven training sample selection, of which the near-miss sampling is an example, has an important role in effective training of machine learning and deep learning models. Even the modern ultra-large language models such as GPT-3 need

a few task-specific prompts (few-shot learning) and the data-driven approaches can help proper selection of the few-shot training samples from the larger, available training data.

Besides the near-miss selection, there are other possible data-driven sample selection techniques. For example, distribution and diversity of samples have already been shown to be effective in sample selection for Active Learning. Other characteristics of text data, such as the part of speech tags, parse trees, dependency trees, word counts, entity distribution, and so on can be leveraged to improve the training data. Similarly, image characteristics can be used to select effective training data. Data-driven sample selection may also play a vital role in the most vexing issues of ethics and bias in machine learning and deep learning models. Data analysis methods can be developed to identify possible bias in the data before the data is used to train the models. The data-driven sample selection in general, is a new research direction with a promising value.

References

1. Guan H, Devarakonda M (2019) Leveraging contextual information in extracting long distance relations from clinical notes. In: Proceeding of annual symposium on AMIA, pp 1051–1060
2. Settles B, Craven M (2008) An analysis of active learning strategies for sequence labeling tasks. In: EMNLP 2008 conference on empirical methods in natural language processing, a meet SIGDAT, a Spec Interes Gr ACL, October 2008, pp 1070–1079. https://doi.org/10.3115/1613715.1613855
3. Settles B (2012) Active learning. Morgan & Claypool
4. Druck G, Settles B, McCallum A (2009) Active learning by labeling features. In: EMNLP 2009—Proceedings of 2009 conference on empirical methods in natural language processing, a meet SIGDAT, a Spec Interes Gr ACL, held conjunction with ACL-IJCNLP 2009, pp 81–90. https://doi.org/10.3115/1699510.1699522
5. Kee S, del Castillo E, Runger G (2018) Query-by-committee improvement with diversity and density in batch active learning. Inf Sci (Ny). 454–455:401–418. https://doi.org/10.1016/j.ins.2018.05.014
6. Winston PH (1975) Learning structural descriptions from examples. In: Winston PH (ed) The psychology of computer vision. McGraw-Hill Book Company, New York
7. Habib R, Dixon MR (2010) Neurobehavioral evidence for the "Near-Miss" effect in pathological gamblers. J Exp Anal Behav 93(3):313–328. https://doi.org/10.1901/jeab.2010.93-313
8. Gurevich N, Markovitch S, Rivlin E (2006) Active learning with near misses. Proc Natl Conf Artif Intell 1:362–367
9. UMass BioNLP. NLP challenges for detecting medication and adverse drug events from electronic health records (MADE1.0)
10. National NLP clinical challenges task 2: adverse drug events and medication extraction in EHRs.
11. Devlin J, Chang. M-W, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HL, Minneapolis, MN
12. Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. In: Proceedings of the 31st conference on neural information processing systems (NIPS 2017), Long Beach, CA. https://doi.org/10.1017/S0952523813000308
13. Taylor WL (1953) Cloze procedure: a new tool for measuring readability. J Bull 30(4):415–433

14. Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre- training. OpenAI
15. Zhu Y, Kiros R, Zemel R et al (2015) Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: 2015 IEEE international conference on computer vision (ICCV), pp 19–27. https://doi.org/10.1109/ICCV.2015.11
16. Wu Y, Schuster M, Chen Z et al (2016) Google's neural machine translation system: bridging the gap between human and machine translation. *arXiv Prepr arXiv160908144v2*
17. Lee J, Yoon W, Kim S et al (2020) BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Bioinformatics 36(4):1234–1240. https://doi.org/10.1093/bioinformatics/btz682
18. Si Y, Wang J, Xu H, Roberts K (2019) Enhancing clinical concept extraction with contextual embeddings. J Am Med Informatics Assoc 26(11):1297–1304. https://doi.org/10.1093/jamia/ocz096
19. Dandala B, Joopudi V, Devarakonda M (2019) Adverse drug events detection in clinical notes by jointly modeling entities and relations using neural networks. Drug Saf 42(1):135–146. https://doi.org/10.1007/s40264-018-0764-x
20. Natural Language Toolkit (NLTK)
21. Chapman AB, Peterson KS, Alba PR, DuVall SL, Patterson OV (2019) Detecting adverse drug events with rapidly trained classification models. Drug Saf 42(1):147–156. https://doi.org/10.1007/s40264-018-0763-y
22. Stubbs A, Buchan K, Filannino M, Uzuner O. National NLP clinical challenges task 2 results
23. Wei Q (2018) UTH: identifying medications and corresponding attributes in electronic health records. In: Slides presented at AMIA N2C2 workshop. https://n2c2.dbmi.hms.harvard.edu/
24. Mikolov T, Sutskever I, Chen K, Corrado G, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of advances in neural information processing systems 26 (NIPS 2013), Lake Tahoe, California, pp 1–9. https://doi.org/10.1162/jmlr.2003.3.4-5.951
25. Dror R, Baumer G, Shlomov S, Reichart R (2018) The hitchhiker's guide to testing statistical significance in natural language processing. In: ACL 2018—Proceedings of 56th annual meeting of the association for computational linguistics, long papers, vol 1, pp 1383–1392.https://doi.org/10.18653/v1/p18-1128
26. Swampillai K, Stevenson M (2011) Extracting relations within and across sentences. In: Proceedings of recent advances in natural language processing, Hissar, Bulgaria, pp 25–32
27. Quirk C, Poon H (2017) Distant supervision for relation extraction beyond the sentence boundary. In: Proceedings of 15th conference on European Chapter of the Association for Computational Linguistics EACL 2017, vol 1, pp 1171–1182. https://doi.org/10.18653/v1/e17-1110
28. Peng N, Poon H, Quirk C, Toutanova K, Yih WT (2017) Cross-sentence N-ary relation extraction with graph LSTMs. *arXiv*. vol 5, pp 101–115. https://doi.org/10.1162/tacl_a_00049

Classifying COVID-19 Variants Based on Genetic Sequences Using Deep Learning Models

Sayantani Basu and Roy H. Campbell

Abstract The COrona VIrus Disease (COVID-19) pandemic led to the occurrence of several variants with time. This has led to an increased importance of understanding sequence data related to COVID-19. In this chapter, we propose an alignment-free k-mer based LSTM (Long Short-Term Memory) deep learning model that can classify 20 different variants of COVID-19. We handle the class imbalance problem by sampling a fixed number of sequences for each class label. We handle the vanishing gradient problem in LSTMs arising from long sequences by dividing the sequence into fixed lengths and obtaining results on individual runs. Our results show that one-vs-all classifiers have test accuracies as high as 92.5% with tuned hyperparameters compared to the multi-class classifier model. Our experiments show higher overall accuracies for B.1.1.214, B.1.177.21, B.1.1.7, B.1.526, and P.1 on the one-vs-all classifiers, suggesting the presence of distinct mutations in these variants. Our results show that embedding vector size and batch sizes have insignificant improvement in accuracies, but changing from 2-mers to 3-mers mostly improves accuracies. We also studied individual runs which show that most accuracies improved after the 20th run, indicating that these sequence positions may have more contributions to distinguishing among different COVID-19 variants.

Keywords COVID-19 · Deep learning · LSTMs · Variants · Gene sequences · Classification

1 COVID-19 and Genetic Data

The COrona VIrus Disease (COVID-19) resulted in more than 170 million cases and more than 3 million deaths worldwide [1]. The virus causing this disease is known as the SARS-CoV2 virus with the first recorded cases in 2019 [2]. In addition

S. Basu · R. H. Campbell (✉)
University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: rhc@illinois.edu

S. Basu
e-mail: basu9@illinois.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_19

to the burden on healthcare resources, other impacts were also observed in other industries including education, economy, and travel [3, 4]. Countries all around the world tried to curb the infections by employing a series of mitigation measures in the form of lockdowns, face coverings, social distancing, frequent sanitizing, limits on gatherings, etc. [5]. There have been several research efforts worldwide to propose and design drugs for treating COVID-19 [6, 7]. In severely affected COVID-19 patients, ventilators and antipyretics have shown varying degrees of efficacy [8, 9]. The common "end" to pandemics is achieved through herd immunity, which can be achieved by ensuring that a significant (preferably the entire) population has antibodies in their system [10]. Vaccination is one such way to achieve this. In December 2020, several vaccine candidates were approved after clinical trials in an effort to protect the population from COVID-19 [11]. However, due to the virulent nature of COVID-19, mass vaccination with careful mitigation measures still need to be employed [12] until significant portions of the population including the vulnerable groups have been fully vaccinated. Studies are still ongoing about the duration of protection provided by these vaccines and whether booster shots will be required for further protection [13]. Another problem that has surfaced with time is the rise of COVID-19 "variants", which are mutations of the original virus that may or may not be more infectious [14]. Most trials show that most COVID-19 vaccines are effective against such variants, although further studies are needed in order to study the newer variants with time [15–17]. The most important aspect in drug design and vaccine design is understanding the underlying make-up of the virus—this includes studying mutations occurring in all variants of the virus. As a result, sequence data is essential in an effort to fully understand the nature of the COVID-19 virus.

Machine learning, specifically deep learning has been helpful in understanding and modeling the COVID-19 pandemic with the enormous amount of data available—several studies have been carried out on image data [18–20], epidemiological data [21–24], and sequence data [25–27]. At this point of the pandemic, the rise of several COVID-19 variants has led to an increased need of studying genetic sequences. More importantly, a system to automatically distinguish and classify the COVID-19 variants using deep learning will help contribute to ongoing research efforts along with vaccine and drug development research.

In this chapter, we propose an LSTM (Long Short-Term Memory) [28] based deep learning model to classify different COVID-19 variants based on genetic sequence data. The rest of this chapter is organized as follows: Sect. 2 discusses the various COVID-19 variants being studied, Sect. 3 presents our data and methods, Sect. 4 discusses our results, Sect. 5 concludes this chapter, and Sect. 6 mentions our code repository and availability of our results.

2 COVID-19 Variants

In this section, we discuss the various COVID-19 variants which have been used in our experiments. We discuss all variants studied with regard to the PANGO lineage convention in the following subsections.

2.1 B.1.1.214

This variant was observed in Japan during the third COVID-19 wave [29] which has been identified to be of 501 N + 484E type [30].

2.2 B.1.1.519

This variant was identified as a variant of interest (VOI) in Mexico [31]. Rhoads et al. [32] have observed the g.29197C > T mutation in all cases of this variant.

2.3 B.1.160

This variant predominantly appeared during the second COVID-19 wave in Eastern Germany, prior to which it was observed in several other European countries [33].

2.4 B.1.177.21

This variant was found to have spread in several European countries [34].

2.5 B.1.177

This is the parent lineage of variant B.1.177.21 [34]. The parent variant is more widespread and has several types of strains [35].

2.6 B.1.1.7

Also known as the Alpha variant, this variant is widespread and infectious [36] and has affected several countries including the United States [37].

2.7 B.1.1

This is another variant of COVID-19 that has been detected in several countries [38].

2.8 B.1.221

This variant of COVID-19 was one of the variants detected predominantly in Eastern Germany leading to the second wave as studied by Yi et al. [33].

2.9 B.1.243

This variant was a variant of interest (VOI) that emerged in Arizona, United States containing the E484K mutation and spread to other parts of the United States [39].

2.10 B.1.258

This variant was observed to spread in several countries including the Romania [40], East Germany [33], Lebanon [41], Czech Republic and Slovakia [42].

2.11 B.1.2

This variant of COVID-19 was observed to have emerged in Brazil [43].

2.12 B.1.351

Also known as the Beta variant, this widespread and infectious variant [36] is known to have the E484K, K417N, N501Y substitutions [37].

2.13 B.1.427

Also known as the Epsilon variant, this COVID-19 variant was identified in parts of the United States and was found to be not as infectious as the Alpha, Beta, and Gamma variants [44, 45].

2.14 B.1.429

Also known as the Epsilon variant, this variant also belongs to the same lineage as that of B.1.427 [45].

2.15 B.1.526

Also known as the Iota variant, this variant was identified in New York, United States [46]. It contains the E484K mutation [47].

2.16 B.1.596

This variant of COVID-19 was observed to have spread in several countries worldwide [48].

2.17 B.1.617.2

Also known as the Delta variant, this variant of COVID-19 was observed to have contributed to the surges of the pandemic in India [49] and has also been observed in England [50] and the United States [37].

2.18 B.1

This is the variant of COVID-19 that constitutes the parent lineage of several COVID-19 variants circulating across the various countries around the world [51].

2.19 D.2

This is the variant of COVID-19 that has been predominantly observed in Australia [52].

2.20 P.1

Also known as the Gamma variant, this variant was first observed in Brazil [53] and has affected several countries worldwide including the United States [37].

3 Data and Methods

Data for this study was collected from GISAID (https://www.gisaid.org). We collected labeled data from 20 lineages of COVID-19 containing genetic sequences in FASTA format from the beginning of the COVID-19 pandemic till June 5, 2021. For the purpose of this study, unclassified sequences that were part of 'None' were not considered.

All experiments in this study were run on gpux1 on the HAL (Hardware-Accelerated Learning) cluster [54].

The presence of multiple variants in the dataset included some variants that were more widespread and virulent than others and some variants that had more genetic samples collected in general. However, from a machine learning perspective, training on samples with imbalanced labels would lead to a class imbalance problem [55]. In order to handle this, we decided to use sampling prior to training our model. For experiments in this study, we limit the number of labels per class to 1000. The sampled sequences were chosen randomly from each of the lineages using random choice from numpy [56].

We have designed our experiments as follows:

1. One-vs-all classification: In this part of our study, we trained and tested our LSTM model on 20 different binary classifiers. Each classifier was trained to identify a specific variant from the rest. As previously discussed, for each classifier, we handle the class imbalance problem in this scenario by considering 1000 sequences from a specific class and 1000 sequences sampled from the rest of the classes.
2. Multi-class classification: In this part of our study, we trained and tested our LSTM model to classify among 20 different variants at once using a single classifier. As previously discussed, in this case too, we handle the class imbalance problem by considering 1000 sequences that are sampled for each class label.

We propose an LSTM (Long Short-Term Memory) [28] based model that would classify COVID-19 sequences into their respective variants. LSTMs are a type of

recurrent neural network model used in deep learning that are capable of capturing dependencies over long periods of time and can be used for modeling time series data. This is useful because variants have temporal consistency making them more obvious to an LSTM model. Each gene sequence is divided into consecutive overlapping k-mers. The k-mers when counted uniquely constitute the vocabulary of the language being considered. For all DNA sequences, we consider the entire alphabet, that is, *{A, T, G, C}* along with the ambiguous nucleotides—*{S, W, R, Y, K, M, B, D, H, V, N}*. A distribution of k-mers revealed the more common nucleotides are *{A, T, G, C, N}*. We have included all unambiguous as well as ambiguous nucleotides since the number of ambiguous nucleotides was small enough compared to the unambiguous nucleotides. In essence, every DNA sequence is transformed into a set of 'words' and we study the ordering of these words using an LSTM based model in order to classify the sequences into their respective variants or classes. We used integer encoding and padding prior to feeding in the sequences into our LSTM framework, all coded using keras [57] with a Tensorflow [58] backend in Python3. However, a problem with directly feeding in sequences in this manner is that in designing such an LSTM, the sequence length, and consequently, the integer encoded words are essentially the timesteps of the model. The minimum sequence length observed in our studies was around 26,000. We modeled this as several timesteps in an LSTM model because of size and to prevent the vanishing gradient problem [59]. Moreover, experiments where we directly attempted to train the LSTM model with the full sequences at once were slow to run due to excessive computational overhead and the programs eventually did not execute until completion. In an effort to overcome this, we divided the problem into 51 steps, where we feed in segments of 500 nucleotides (characters of the genetic sequences) at a time. This helped the programs run successfully until completion. Each of the predictions from the test set were then averaged over the 51 steps and thresholded with 0.5 in order to obtain the overall accuracy of the model. This framework is suitable for the following reasons: (1) it helps reduce computational overload of feeding in multiple timesteps at once, (2) it handles the vanishing gradient problem of LSTMs that can arise due to a large number of timesteps, and (3) it enables us to find distinguishing portions of the genetic sequences that contribute to better accuracies—such portions of data are "useful" for identifying and classifying the specific COVID-19 variant. The only problem with this approach would be missing nucleotides of lengthy sequences located towards the end due to truncating the sequences after fixed indices in order to allow the model to train effectively. It is also to be noted at this point that we have used an alignment-free k-mer based approach in all our experiments in this study.

3.1 Model Architecture

The value of k for generating k-mers is a hyperparameter itself. Small k-mer values help in faster extraction of k-mers from long sequences, but do not provide adequate coverage compared to larger k values.

For our proposed framework, we use an embedding layer to convert the integer encoded k-mer words into vectors of a fixed size. The embedding vector size constitutes a hyperparameter, along with the batch size that we varied in our experiments.

We use 10 LSTM units and 1 dense layer with sigmoid activation in order to obtain the probabilities in the case of all our binary classification one-vs-all models. We use Adam [60] as the optimizer with binary cross entropy and accuracy as the metric in order to train the model for 50 epochs. We use Sequential() from keras to stack the layers in a linear fashion.

We used 10 LSTM units and 20 dense units with softmax activation for our multi-class classification models. We use Adam [60] as the optimizer with sparse categorical cross entropy and accuracy as the metric in order to train the model for 50 epochs. Similar to the one-vs-all classifiers, we use Sequential() from keras to stack the layers in a linear fashion.

4 Results and Discussion

The entire set of sampled sequences was shuffled retaining original variant labels and then divided into an 80% and 20% ratio for training and testing data respectively. In the results, we primarily focus on discussing accuracies on test data since these comprise results on "unseen" data and are representative of what the LSTM model has learnt from the training process.

Table 1 shows the accuracies of varying hyperparameters for $k = 2$. The cells in the table show accuracy as a percentage calculated as a combination of the predictions across all 51 runs. The predicted probabilities from the Dense layer are thresholded using 0.5 to represent whether a sample belongs to a class or not. The predictions are then compared with the ground truth class labels and the accuracies for every classifier are obtained using sklearn [61]. Each row in the table represents a classifier where the task is classifying the specific variant from other COVID-19 variants. Based on our experiments, changing the embedding size and batch size did not have significant effect on the accuracies. It is important to note that certain classifiers like those of B.1.1.214, B.1.177.21, B.1.1.7, B.1.526, and P.1 have better accuracies compared to others—these higher accuracy scores may be because we are essentially trying to classify one specific variant and it would need to have features that are distinct enough for the classifier to tell it apart from the other classes. This may be a broader indicator of variants that have "novel mutations" of COVID-19 with genetic makeup different enough from the rest of the sequences from other variants.

Within the accuracies from individual runs, positions after the 20th run till the 51st run showed higher accuracies in general, hence indicating that these positions may be of more interest in contributing to the "distinction" of genetic makeup compared to the beginning portions of the sequences. The exact positions vary based on the specific classifier and hyperparameters. The entire sets of accuracies obtained from our individual runs as well as the final results are included in our code repository.

Table 2 shows the accuracies of varying hyperparameters for $k = 3$. While there

Table 1 Accuracies for classifying COVID-19 lineages with k = 2

Variant	emb_vec = 50	emb_vec = 100	emb_vec = 50	emb_vec = 100
	batch_size = 256	batch_size = 256	batch_size = 128	batch_size = 128
B.1.1.214	80.25	80.5	82.5	83
B.1.1.519	65.25	64	64.25	64.25
B.1.160	67.75	70	64.5	68.5
B.1.177.21	80.5	80.5	80.5	80.5
B.1.177	63.75	63.5	64	67
B.1.1.7	90.5	90.5	90.5	90.5
B.1.1	63.5	64.5	64	62.5
B.1.221	71.25	72.25	71.75	73.5
B.1.243	64.75	64.25	64.75	65.5
B.1.258	68	69	65.75	69.75
B.1.2	61.5	61.5	61.75	61
B.1.351	83.5	83.75	83.25	83.75
B.1.427	62.5	62.25	62	62
B.1.429	68	65.5	63.25	65.25
B.1.526	81.75	83	82.25	84.25
B.1.596	64	64.25	64.25	64.25
B.1.617.2	73	73	74.25	74.5
B.1	59	57.5	58	57.5
D.2	74.75	74.75	75.25	75.5
P.1	86	84	84	84

is not much significant change in accuracies compared to the experiments for k = 2, it is interesting to note that for certain cases, it is beneficial to use k = 3. In this case too, changes in embedding vector size and batch size did not lead to any significant changes in the accuracies. The most significant increase in accuracy was observed in the case of B.1.160. This may be because 3-mers may be more representative of distinguishing characteristic features of this variant compared to 2-mers. Based on the results, accuracies generally improve after the 20th run which indicates that these positions may be of interest in distinguishing characteristic features for a certain class similar to the 2-mers. It is to be noted that while increasing the value of k in k-mers may add more coverage, the process of extracting sequences takes longer time.

For the multi-class classifier, we varied hyperparameters and obtained accuracies for individual runs for embedding vector sizes of 50, 100 and batch sizes of 256, 128 for both k = 2 and k = 3. The results show accuracies of less than 50% for most of the individual runs in all cases, which are significantly lesser than the accuracies of individual runs obtained for the one-vs-all classifiers. This may be because it is harder to correctly distinguish among 20 classes and classify the same compared to a set of binary classifiers identifying whether a specific variant belongs

Table 2 Accuracies for classifying COVID-19 lineages with k = 3

Variant	emb_vec = 50	emb_vec = 100	emb_vec = 50	emb_vec = 100
	batch_size = 256	batch_size = 256	batch_size = 128	batch_size = 128
B.1.1.214	82	81.75	82	82.5
B.1.1.519	63.75	63.75	64	65.5
B.1.160	77.75	78.25	78.25	71.5
B.1.177.21	80.5	80.5	80.5	80.5
B.1.177	67.25	66	64.5	66.75
B.1.1.7	91	91	91	92.5
B.1.1	63.5	63.75	63.75	63.25
B.1.221	73.25	73	73	72.75
B.1.243	65	65	65	64.25
B.1.258	70.25	68.5	69.25	73.25
B.1.2	61.75	61.75	61.5	62.75
B.1.351	84	84.25	84	85.5
B.1.427	62.25	62.25	62.25	64
B.1.429	65	65	65.75	65.75
B.1.526	84.75	84.5	85	84
B.1.596	65.75	65.75	66	66.5
B.1.617.2	74.75	74.75	74.75	75.75
B.1	58	57	57.5	56.5
D.2	75.75	75.75	76	76.25
P.1	84.5	84.75	84.75	85.75

to a class or not. In terms of individual accuracies of runs, it is more beneficial to build 20 binary classifiers compared to a single classifier classifying 20 classes. While comparing individual accuracies, though the accuracies themselves are low, the accuracies for multi-class classifiers also show an increase in accuracy after the 20th run in general indicating that these may be positions contributing to distinguishing features, which was also observed in case of the one-vs-all classifiers. Results obtained from individual runs of the multi-class classifier are also available in our code repository.

5 Conclusion

In this chapter, we proposed an LSTM based deep learning model that can classify COVID-19 variants based on genetic sequences. Our proposed method is alignment free and uses k-mers. In our method, we handle the class imbalance problem by sampling a fixed set of sequences per class. We also handle the vanishing gradient problem of LSTMs arising from long timesteps due to long lengths of genetic

sequences by dividing the sequences into fixed segments of smaller timesteps. We discuss results for accuracies on both one-vs-all classifiers as well as multi-class classifier models. Our results show that the one-vs-all classifier performs better than the multi-class classifier models for distinguishing variants on individual runs, with test accuracies as high as 92.5%. Higher overall accuracies in variants like B.1.1.214, B.1.177.21, B.1.1.7, B.1.526, and P.1 may suggest that these variants have specific mutations that distinguish them from the other variants in case of the one-vs-all classifiers. In terms of hyperparameters, varying the embedding vector sizes and batch sizes did not change the accuracies in a significant manner based on our experiments. However, $k = 3$ yielded slightly better accuracies compared to $k = 2$ in most cases. We also observed that the accuracies on individual runs improve after the 20th run in general, which may suggest that these positions are more important in determining distinctive features contributing to a specific class. We hope that our model can be extended in the future for understanding COVID-19 variants based on sequence data and believe that it can help contribute to vaccine and drug development research.

6 Code

Code and results from our experiments are available here: https://github.com/sayantanibasu/covid19-gene-variants.

Acknowledgements This project has been funded by the Jump ARCHES endowment through the Health Care Engineering Systems Center.

This work uses resources from GISAID (https://www.gisaid.org). We would like to acknowledge all laboratories that have contributed their COVID-19 sequence data to GISAID.

This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant #1725729, as well as the University of Illinois at Urbana-Champaign.

References

1. Hopkins J, Coronavirus resource center. https://coronavirus.jhu.edu
2. Riou J, Althaus CL (2020) Pattern of early human-to-human transmission of Wuhan 2019 novel coronavirus (2019-nCoV), December 2019 to January 2020. Eurosurveillance 25(4):2000058
3. Nayak J, Mishra M, Naik B, Swapnarekha H, Cengiz K, Shanmuganathan V (2021) An impact study of COVID-19 on six different industries: automobile, energy and power, agriculture, education, travel and tourism and consumer electronics. Expert Syst
4. Shrestha N, Shad MY, Ulvi O, Khan MH, Karamehic-Muratovic A, Nguyen USDT, Baghbanzadeh M, Wardrup R, Aghamohammadi N, Cervantes D et al (2020) The impact of COVID-19 on globalization. One Health 100180
5. Walker P, Whittaker C, Watson O, Baguelin M, Ainslie K, Bhatia S, Bhatt S, Boonyasiri A, Boyd O, Cattarino L et al (2020) Report 12: the global impact of COVID-19 and strategies for mitigation and suppression

6. COVID-19 (coronavirus) drugs: are there any that work? https://www.mayoclinic.org/diseases-conditions/coronavirus/expert-answers/coronavirus-drugs/faq-20485627
7. Si L, Bai H, Rodas M, Cao W, Oh CY, Jiang A, Nurani A, Zhu DY, Goyal G, Gilpin SE et al (2020) Human organs-on-chips as tools for repurposing approved drugs as potential influenza and COVID19 therapeutics in viral pandemics. bioRxiv
8. Rinott E, Kozer E, Shapira Y, Bar-Haim A, Youngster I (2020) Ibuprofen use and clinical outcomes in COVID-19 patients. Clin Microbiol Infect 26(9):1259-e5
9. Payen JF, Chanques G, Futier E, Velly L, Jaber S, Constantin JM (2020) Sedation for critically ill patients with COVID-19: which specificities? one size does not fit all. Anaesth Crit Care Pain Med 39(3):341
10. Fontanet A, Cauchemez S (2020) COVID-19 herd immunity: where are we? Nat Rev Immunol 20(10):583–584
11. Le TT, Andreadakis Z, Kumar A, Román RG, Tollefsen S, Saville M, Mayhew S et al (2020) The COVID-19 vaccine development landscape. Nat Rev Drug Discov 19(5):305–306
12. Marziano V, Guzzetta G, Mammone A, Riccardo F, Poletti P, Trentini F, Manica M, Siddu A, Stefanelli P, Pezzotti P, et al (2021) Return to normal: COVID-19 vaccination under mitigation measures. medRxiv
13. Mahase E (2021) COVID-19: booster dose will be needed in autumn to avoid winter surge, says government adviser
14. Chen J, Gao K, Wang R, Wei GW (2021) Prediction and mitigation of mutation threats to COVID-19 vaccines and antibody therapies. Chem Sci
15. Shinde V, Bhikha S, Hoosain Z, Archary M, Bhorat Q, Fairlie L, Lalloo U, Masilela MS, Moodley D, Hanley S et al (2021) Efficacy of NVX-CoV2373 COVID-19 vaccine against the B.1.351 variant. N Engl J Med 384(20):1899–1909
16. Abu-Raddad LJ, Chemaitelly H, Butt AA (2021) Effectiveness of the BNT162b2 COVID-19 vaccine against the B.1.1.7 and B.1.351 variants. N Engl J Med
17. Madhi SA, Baillie V, Cutland CL, Voysey M, Koen AL, Fairlie L, Padayachee SD, Dheda K, Barnabas SL, Bhorat QE et al (2021) Efficacy of the ChAdOx1 nCoV-19 COVID-19 vaccine against the B.1.351 variant. N Engl J Med 384(20):1885–1898
18. Oh Y, Park S, Ye JC (2020) Deep learning COVID-19 features on cxr using limited training data sets. IEEE Trans Med Imaging 39(8):2688–2700
19. Amyar A, Modzelewski R, Li H, Ruan S (2020) Multi-task deep learning based CT imaging analysis for COVID-19 pneumonia: classification and segmentation. Comput Biol Med 126:104037
20. Yan Q, Wang B, Gong D, Luo C, Zhao W, Shen J, Shi Q, Jin S, Zhang L, You Z (2020) COVID-19 chest CT image segmentation–a deep convolutional neural network solution. arXiv Prepr. arXiv:2004.10987
21. Basu S, Campbell RH (2020) Going by the numbers: learning and modeling COVID-19 disease dynamics. Chaos Solitons Fractals 138:110140
22. Basu S (2020) A study of the dynamics and genetics of COVID-19 through machine learning. Master's thesis, University of Illinois at Urbana-Champaign
23. Bhouri MA, Costabal FS, Wang H, Linka K, Peirlinck M, Kuhl E, Perdikaris P (2021) COVID-19 dynamics across the US: a deep learning study of human mobility and social behavior. Comput Methods Appl Mech Eng 382:113891
24. Muhammad LJ, Algehyne EA, Usman SS, Ahmad A, Chakraborty C, Mohammed IA (2021) Supervised machine learning models for prediction of COVID-19 infection using epidemiology dataset. SN comput Sci 2(1):1–13
25. Bouhamed H (2020) COVID-19 cases and recovery previsions with deep learning nested sequence prediction models with long short-term memory (LSTM) architecture. Int J Sci Res Comput Sci Eng 8(2)
26. Randhawa GS, Soltysiak MP, El Roz H, de Souza CP, Hill KA, Kari L (2020) Machine learning using intrinsic genomic signatures for rapid classification of novel pathogens: COVID-19 case study. PLoS ONE 15(4):e0232391

27. Pathan RK, Biswas M, Khandaker MU (2020) Time series prediction of COVID-19 by mutation rate analysis using recurrent neural network-based LSTM model. Chaos Solitons Fractals 138:110018
28. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
29. Sekizuka T, Itokawa K, Hashino M, Okubo K, Ohnishi A, Goto K, Tsukagoshi H, Ehara H, Nomoto R, Ohnishi M et al (2021) A discernable increase in the severe acute respiratory syndrome coronavirus 2 R. 1 lineage carrying an E484K Spike protein mutation in Japan. medRxiv
30. Nagano K, Tani-Sassa C, Iwasaki Y, Takatsuki Y, Yuasa S, Takahashi Y, Nakajima J, Sonobe K, Ichimura N, Nukui Y et al (2021) SARS-CoV-2 R. 1 lineage variants prevailed in Tokyo in March 2021. medRxiv
31. Rodriguez-Maldonado AP, Vazquez-Perez JA, Cedro-Tanda A, Taboada B, Boukadida C, Wong-Arambula C, Nunez-Garcia TE, Cruz-Ortiz N, Barrera-Badillo G, Hernandez-Rivas L et al (2021) Emergence and spread of the potential variant of interest (VOI) B. 1.1. 519 predominantly present in Mexico. medRxiv
32. Rhoads DD, Plunkett D, Nakitandwe J, Dempsey A, Tu ZJ, Procop GW, Bosler D, Rubin BP, Loeffelholz MJ, Brock JE (2021) Endemic SARS-CoV-2 polymorphisms can cause a higher diagnostic target failure rate than estimated by aggregate global sequencing data. J Clin Microbiol JCM–00913
33. Yi B, Poetsch AR, Stadtmüller M, Rost F, Winkler S, Dalpke AH (2021) Phylogenetic analysis of SARS-CoV-2 lineage development across the first and second waves in Eastern Germany, 2020. bioRxiv
34. B.1.177.21 PANGO lineage. https://cov-lineages.org/lineages/lineage_B.1.177.21.html
35. Amato L, Jurisic L, Puglia I, Di Lollo V, Curini V, Torzi G, Di Girolamo A, Mangone I, Mancinelli A, Decaro N et al (2021) Multiple detection and spread of novel strains of the SARS-CoV-2 B. 1.177 (B. 1.177. 75) lineage that test negative by a commercially available nucleocapsid gene real-time RT-PCR. Emerg Microbes Infect (just-accepted):1–19
36. Planas D, Bruel T, Grzelak L, Guivel-Benhassine F, Staropoli I, Porrot F, Planchais C, Buchrieser J, Rajah MM, Bishop E et al (2021) Sensitivity of infectious SARS-CoV-2 B.1.1.7 and B.1.351 variants to neutralizing antibodies. Nat Med 27(5):917–924
37. SARS-CoV-2 variant classifications and definitions. https://www.cdc.gov/coronavirus/2019-ncov/variants/variant-info.html
38. B.1.1 PANGO lineage. https://cov-lineages.org/lineages/lineage_B.1.1.html
39. Skidmore PT, Kaelin EA, Holland LA, Maqsood R, Wu LI, Mellor NJ, Blain JM, Harris V, LaBaer J, Murugan V et al (2021) Emergence of a SARS-CoV-2 E484K variant of interest in Arizona. medRxiv
40. Surleac M, Casangiu C, Banica L, Milu P, Florea D, Sandulescu O, Streinu-Cercel A, Vlaicu O, Tudor A, Hohan R et al (2021) Evidence of novel SARS-CoV-2 variants circulation in Romania. AIDS Res Hum Retroviruses 37(4):329–332
41. Younes M, Hamze K, Carter DP, Osman KL, Vipond R, Carroll M, Pullan ST, Nassar H, Mohamad N, Makki M et al (2021) B.1.1.7 became the dominant variant in Lebanon. medRxiv
42. Brejová B, Hodorová V, Boršová K, Čabanová V, Reizigová L, Paul ED, Čekan P, Klempa B, Nosek J, Vinař T (2021) B. 1.258 O, a SARS-CoV-2 variant with O H69/O V70 in the Spike protein circulating in the Czech Republic and Slovakia. arXiv Prepr. arXiv:2102.04689
43. Fonseca V, de Jesus R, Adelino T, Reis AB, de Souza BB, Ribeiro AA, Guimarães NR, Livorati MT, de Lima Neto DF, Kato RB et al (2021) Genomic evidence of SARS-CoV-2 reinfection case with the emerging B.1.2 variant in Brazil. J Infect
44. Webb LM, Matzinger S, Grano C, Kawasaki B, Stringer G, Bankers L, Herlihy R (2021) Identification of and surveillance for the SARS-CoV-2 variants B.1.427 and B.1.429—Colorado, January–March 2021. Morb Mortal Wkly Rep 70(19):717
45. Deng X, Garcia-Knight MA, Khalid MM, Servellita V, Wang C, Morris MK, Sotomayor-González A, Glasner DR, Reyes KR, Gliwa AS et al (2021) Transmission, infectivity, and antibody neutralization of an emerging SARS-CoV-2 variant in California carrying a L452R spike protein mutation. medRxiv

46. Annavajhala MK, Mohri H, Zucker JE, Sheng Z, Wang P, Gomez-Simmonds A, Ho DD, UhlemannAC (2021) A novel SARS-CoV-2 variant of concern, B.1.526, identified in New York. medRxiv
47. Lasek-Nesselquist E, Lapierre P, Schneider E, George KS, Pata J (2021) The localized rise of a B.1.526 variant containing an E484K mutation in New York State. medRxiv
48. B.1.596 PANGO lineage. https://cov-lineages.org/lineages/lineage_B.1.596.html
49. Bernal JL, Andrews N, Gower C, Gallagher E, Simmons R, Thelwall S, Tessier E, Groves N, Dabrera G, Myers R et al (2021) Effectiveness of COVID-19 vaccines against the B.1.617.2 variant. medRxiv
50. Challen R, Dyson L, Overton CE, Guzman-Rincon LM, Hill EM, Stage HB, Brooks-Pollock E, Pellis L, Scarabel F, Pascall DJ et al (2021) Early epidemiological signatures of novel SARS-CoV-2 variants: establishment of B.1.617.2 in England. medRxiv
51. B.1 PANGO lineage. https://cov-lineages.org/lineages/lineage_B.1.html
52. D.2 pango lineage. https://cov-lineages.org/lineages/lineage_D.2.html
53. Coutinho RM, Marquitti FM, Ferreira LS, Borges ME, da Silva RL, Canton O, Portella TP, Lyra SP, Franco C, da Silva AAM et al (2021) Model-based evaluation of transmissibility and reinfection for the P. 1 variant of the SARS-CoV-2. medRxiv
54. Kindratenko V, Mu D, Zhan Y, Maloney J, Hashemi SH, Rabe B, Xu K, Campbell R, Peng J, Gropp W (2020) HAL: computer system for scalable deep learning. In: Practice and experience in advanced research computing, pp 41–48
55. Guo X, Yin Y, Dong C, Yang G, Zhou G (2008) On the class imbalance problem. In: 2008 Fourth international conference on natural computation, vol 4. IEEE, pp 192–201
56. Van Der Walt S, Colbert SC, Varoquaux G (2011) The NumPy array: a structure for efficient numerical computation. Comput Sci Eng 13(2):22–30
57. Gulli A, Pal S (2017) Deep learning with keras. Packt Publishing Ltd
58. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16), pp 265–283
59. Sundermeyer M, Schlüter R, Ney H (2012) LSTM neural networks for language modeling. In: Thirteenth annual conference of the international speech communication association
60. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv Prepr. arXiv:1412.6980
61. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V et al (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830

Twenty-First Century Cybernetics and Disorders of Brain and Mind

Gregory Worrell

The potential for bridging computer science, neural engineering and implantable devices for new therapeutics targeting disorders of the brain and mind is well recognized. The rapidly evolving field is at an inflection point created by advances in computing, engineering, and the understanding of brain mechanisms and disease. Here I review research with my friend Professor Ravi Iyer, PhD focused on accelerating the application of machine learning and computing to devices designed to treat epilepsy and associated comorbidities. Over the past 5 years we have worked together to bridge some of the gaps between research and application to create new therapies for patients with drug resistant epilepsy. Ravi and his graduate students at University Illinois Urbana Champaign (UIUC) and my group have enjoyed a productive collaboration in the emerging field of Bioelectronics and Neuromodulation.

I met Ravi in approximately 2015 as part of the Mayo Clinic and UIUC collaboration around the emerging medical applications of artificial intelligence. We have enjoyed a collaboration that has involved multiple students, and in particular Ravi's two doctoral students Yogatheesan Varatharajah, who completed his PhD and is now working in the Department of Neurology at Mayo Clinic, and Krishnakant Saboo who is a 4th year UIUC PhD. student. Our research has focused on a couple of fundamental problems in drug resistant focal epilepsy: (1) The spatial localization of focal epilepsy, or simply stated as the prediction of where seizures are generated (2) The temporal forecasting of seizures, or when in time seizures will occur (3) The cognitive, memory and sleep comorbidities of epilepsy. (4) The next generation of implantable devices that enable sensing, electrical stimulation, and embedded analytics and their integration with local and distributed computing.

G. Worrell (✉)
Bioelectronics Neurophysiology and Engineering Lab (BNEL), Department of Neurology and Physiology and Biomedical Engineering, Mayo Clinic, Rochester, MN, USA
e-mail: Worrell.gregory@mayo.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_20

1 The Spatial Localization of Focal Epilepsy: Predicting Where Seizures Are Generated

Spatial localization of focal epileptic brain is a cornerstone of epilepsy diagnostics and critical to successful epilepsy surgery and electrical brain stimulation (EBS). The gold-standard approach is to record a patient!s habitual seizures with intracranial electrodes.

Unfortunately, it typically takes days or even weeks to capture a habitual spontaneous seizure. If seizures are successfully localized that area of the brain can potentially be targeted for resection or EBS. The possibility that epileptogenic tissue and networks can be determined from interictal recordings, electrophysiological recordings without seizures is an active area of research. If successful this would open the possibility of short intra-operative recordings for mapping epileptic networks and tissue, and might even better define the margins and nodes of optimal targets of resection and EBS. In the UIUC-Mayo collaboration we have used modern machine learning approaches to identify epileptic brain and networks to map epileptic and normal brain networks [1–6] and are now investigating the application of this approach to clinical practice.

In more recent work we investigated scalp EEG, a widely used non-invasive technology, to characterize the electrophysiological abnormalities in expert reviewed normal EEG from patients with drug resistant epilepsy and demonstrated the ability to predict the presence and lateralization of their epilepsy [7].

2 The Temporal Forecasting of Seizures: Predicting When Seizures Will Occur

For most individuals living with epilepsy, seizures are relatively infrequent events occupying a small fraction of their life. Despite spending a small fraction of their lives having seizures, often only minutes per month, people with epilepsy take seizure drugs (ASD) daily, suffer ASD related side effects, and spend their lives dreading when the next seizure will strike. The apparent randomness of seizures is associated with significant psychological consequences [8]. In addition, despite daily ASD approximately 1/3 of patients continue to have seizures. We hypothesize that epilepsy can be more effectively treated, both the seizures and their psychological impact, by providing patients with real-time seizure forecasting. Periods of low seizure probability would not require ASDs, or at least lower doses, thus reducing ASD exposure and their side effects. Periods of high seizure probability may respond to acute, fast acting ASD or a change in electrical brain stimulation parameters. In addition, patients could alter their activities to avoid injuries that are often associated with seizures. Patients would be empowered to manage their medications and life activities using reliable seizure forecasts. Similarly, EBS parameters and therapy might be adaptively changed to reduce the risk of seizures and the side effects. We have extensively investigated the hypothesis that seizures are predictable events, and pursued

accurate, clinically relevant seizure forecasting using advances in support vector machines (SVM), convolutional neural networks, and data-analytic models applied to continuous intracranial EEG (iEEG) in focal canine epilepsy [9–14]. This is an initial step in establishing a new treatment paradigm for focal epilepsy, whereby the probability of seizure occurrence is continuously tracked for patient warning and intelligent responsive therapies.

3 The Cognitive, Memory and Sleep Comorbidies of Epilepsy

Sleep, cognitive and mood disturbances are common comorbidities of epilepsy. Moreover, EBS therapy itself may have a negative impact on sleep, cognition and mood. Unfortunately, objective assessment and quantification and the absence of longitudinal data and the established unreliability of patient self-reporting has limited progress in the field.

We are developing a range of quantitative tools to track sleep [15], cognition [5, 16] and mood in patients with the ultimate goal of more intelligently intervening with non-pharmacologic, pharmacologic, and EBS therapies. One area of progress has been with the quantification of memory and manipulation with EBS [5, 16]. This work advances efforts to better understand the bidirectional relationship between epilepsy and cognition, and the impact of EBS. Lastly, in the future the selective application of EBS may prove useful for a range of neurologic and psychiatric diseases associated cognition deficits.

4 Next Generation of Implantable Devices that Enable Sensing, Electrical Stimulation, Embedded Analytics and Their Integration with Local and Distributed Computing

The pace of commercial electronics development far outpaces what is possible for medical device technology. The gap in development speed can be partially overcome by the integration of implantable devices with off-the-body local and distributed computing. In this paradigm implanted brain sensing and stimulation devices are integrated with a smartphone to create a bi-directional interface between human brain and computation infrastructure. The brain sensing data are streamed off the device to a smartphone, a local computational node, that also provides a platform for patient inputs. This opens a useful bidirectional interface that enables large scale computing not possible using an embedded algorithm. Next generation devices will enable intelligent, adaptive EBS that not only optimizes seizures reductions, but also addresses the sleep, cognitive and mood comorbidities of epilepsy [4, 17].

What the future holds: The value of embedding engineers and scientists in multi-disciplinary teams of including physicians and clinical staff is now widely

recognized. Nonetheless, neurotechnology is failing to deliver on its diagnostic and therapeutic promise because the time it takes to translate from the bench to the clinic.

To accelerate human translational Mayo Clinic and UIUC has assembled multidisciplinary teams of engineers, scientists, and clinician teams. We have recently deployed a distributed neural-coprocessors for adaptive neuromodulation in patients with epilepsy. The system includes a fully functional, implantable device (sensing, stimulation, embedded analytics, neural co-processor API), wearable sensors, local and cloud computational infrastructure. We believe the future should find many diseases applications. Our success will be measured by our ability to create new therapies for patients (Fig. 1).

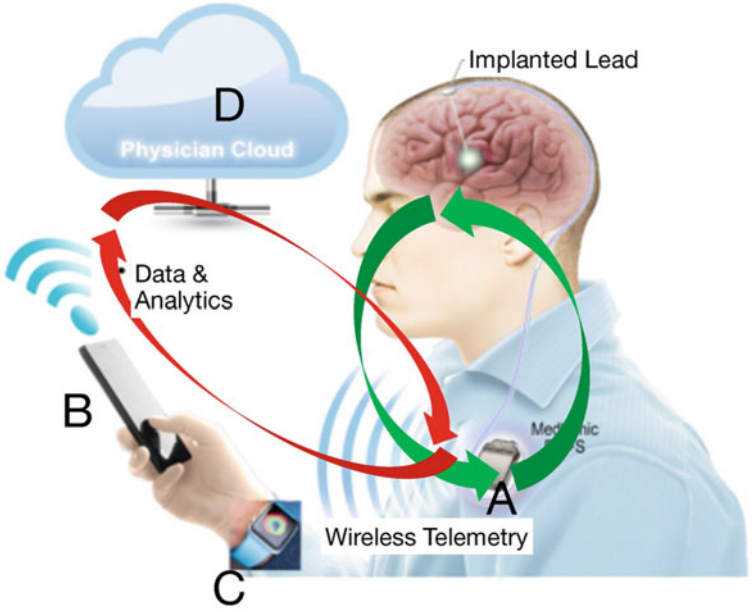


Fig. 1 Next generation implanted device integrated with local and distributed computing environment for epilepsy management. The system seamlessly integrates implantable device sensing and stimulation with off-the-body computing smartphones and cloud computing. Short latency responsive therapy (green arrow) is embedded on the device and provides millisecond timescale responses to abnormal brain activity. The bi-directional wireless interface allows continuous iEEG telemetry and algorithms running on the phone or in the cloud provide longer latency adaptive stimulation (red arrows), given more computing time that is not feasible using implantable device embedded algorithms. A particular innovation emerges from whereby electrophysiology and behavior are integrated. The paradigm shifting approach of using electrophysiology detections (e.g. seizures or coupling fluctuations in limbic networks) to trigger patient smartphone or watch queries and interactions. In this way electrophysiologic events of interest, such as seizure discharges or abnormal synchrony, can be selectively investigated with focused patient interactions to probe behavior, cognition, mood etc. Lastly, the system enables integration with wearable sensors like a smart watch and connects physicians and patients

References

1. Varatharajah Y et al (2020) Electrophysiological correlates of brain health help diagnose epilepsy and lateralize seizure focus. In: 2020 42nd annual international conference of the IEEE Engineering in Medicine Biology Society (EMBC), pp 3460–3464. https://doi.org/10.1109/EMBC44109.2020.9176668
2. Varatharajah Y et al (2017) Inter-ictal seizure onset zone localization using unsupervised clustering and Bayesian filtering. In: International IEEE/EMBS conference on neural engineering, NER. https://doi.org/10.1109/NER.2017.8008407
3. Varatharajah Y et al (2017) EEG-GRAPH: a factor-graph-based model for capturing spatial, temporal, and observational relationships in electroencephalograms. In: Advances in neural information processing systems, Dec 2017
4. Varatharajah Y et al (2018) Integrating artificial intelligence with real-time intracranial EEG monitoring to automate interictal identification of seizure onset zones in focal epilepsy. J Neural Eng 15
5. Saboo KV et al (2019) Unsupervised machine-learning classification of electrophysiologically active electrodes during human cognitive task performance. Sci Rep 9
6. Weiss SA et al (2018) Visually validated semi-automatic high-frequency oscillation detection aides the delineation of epileptogenic regions during intra-operative electrocorticography. Clin Neurophysiol 129
7. Varatharajah Y et al (2021) Characterizing the electrophysiological abnormalities in visually reviewed normal EEGs of drug-resistant focal epilepsy patients. Brain Commun 3
8. Elger CE, Mormann F (2013) Seizure prediction and documentation-two important problems. Lancet Neurol 12:531–532
9. Varatharajah Y, Iyer RK, Berry BM, Worrell GA, Brinkmann BH (2017) Seizure forecasting and the preictal state in canine epilepsy. Int J Neural Syst 27
10. Nejedly P et al (2019) Deep-learning for seizure forecasting in canines with epilepsy. J Neural Eng
11. Brinkmann BH et al (2016) Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. Brain 139
12. Gregg NM et al (2020) Circadian and multiday seizure periodicities, and seizure clusters in canine epilepsy. Brain Commun. https://doi.org/10.1093/braincomms/fcaa008
13. Brinkmann BH et al (2015) Forecasting seizures using intracranial EEG measures and SVM in naturally occurring canine epilepsy. PLoS ONE. https://doi.org/10.1371/journal.pone.0133900
14. Kuhlmann L et al (2018) Epilepsyecosystem.org: crowd-sourcing reproducible seizure prediction with long-term human intracranial EEG. Brain. https://doi.org/10.1093/brain/awy210
15. Kremen V et al (2017) Behavioral state classification in epileptic brain using intracranial electrophysiology. J Neural Eng 14
16. Kucewicz MT et al (2018) Evidence for verbal memory enhancement with electrical brain stimulation in the lateral temporal cortex. Brain 141
17. Denison T et al (2021) Stimulating Solutions for Intractable Epilepsy. Epilepsy Curr. https://doi.org/10.1177/15357597211012466

Dependability Assessment

Introduction: Dependability Assessment

Karthik Pattabiraman

Dependability assessment is one of the key activities in dependable system design, as it is important to have high confidence in the dependability of the system. Prof. Ravi Iyer has worked extensively in this domain, starting from his early career in the late 1970s to the present. In the early phase of his career, his work in this domain mainly encompassed dependability modeling, using analytical and simulation techniques. In the next phase, he worked on measurement-based analysis of computer systems dependability, especially on production systems deployed at Stanford University and IBM, and he has made many important contributions in the area. Finally, he's worked extensively in the area of fault injection, and his group has pioneered many techniques for both software and hardware-based fault injection. It is therefore especially fitting that the three articles in this part of the book are each in one of the three areas of dependability assessment to which Ravi has contributed.

The first article in this part by Prof. Kishor Trivedi and others, is on how to deal with epistemic uncertainties in reliability models. Unlike aleatory uncertainty that pertains to randomness in model parameters, epistemic uncertainty has to do with incomplete information about model parameters. It is therefore much more difficult to get rid of this uncertainty via collecting higher fidelity data, for example. One way to deal with epistemic uncertainty is to treat the parameters of the model themselves as random variables, and apply the laws of Bayesian probability to them. Unfortunately, this complicates the process of obtaining an analytical solution to the model due to the use of multiple integration—this is the problem addressed by the paper.

The second article by Karama Kanoun and Mohamed Kaaniche discusses the authors' experience of performing dependability assessment in collaboration with industry, ranging from stochastic models such as Petri nets to analysis of field data. The case studies are done with different companies, and at different times in the authors' careers (starting from the late 1980s to the recent past). The four

K. Pattabiraman (✉)
The University of British Columbia (UBC), Vancouver, Canada
e-mail: karthikp@ece.ubc.ca

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_21

vignettes present a structured progression of ideas from purely analytical approaches to simulation and then to online data collection in production systems.

The final article is by Saurabh Jha, one of Ravi's recent PhD graduates, and he discusses his research on performing dependability assessment of autonomous vehicles (AVs) in three different directions. The first direction is to perform empirical data analysis using a causal model of the system using the System Theoretic Process Analysis (STPA) methodology—this is demonstrated via a study of disengagement data in AVs in the state of California in the United States. The second direction is to use fault injection and fuzz testing to find corner case faults that can lead to safety violations in AVs, using a causal reasoning based technique. The final direction is to extend the fuzzing technique to find security attacks in AVs that masquerade as failures and lead to safety violations. These three techniques also span the gamut from theoretical reasoning to experimental validation, in tune with Ravi's research focus.

Overall, the three chapters provide a nice window into the different facets of dependability assessment that Ravi has worked on. We hope the reader enjoys reading them as much as we did, and also finds inspiration from them for future research directions and ideas.

Effect of Epistemic Uncertainty in Markovian Reliability Models

Hiroyuki Okamura, Junjun Zheng, Tadashi Dohi, and Kishor S. Trivedi

Abstract This chapter introduces the moment-based epistemic uncertainty propagation in Markov models. The epistemic uncertainty in Markov models introduces the uncertainty of model parameters, and it can be propagated by regarding parameters as random variables. The idea behind the moment-based approach is to approximate the multiple integration with a series expansion of model parameters. This leads to the efficient computation of the uncertainty in the expected output measure. The expected output measure is represented by the expected value and the variance of model parameters and the first and second derivatives of output measure with respect to model parameters. In this chapter, we introduce the formulation of moment-based epistemic uncertainty propagation and the concrete methods to obtain the first and second derivatives of output measures in Markov models.

Keywords Epistemic uncertainty propagation · Markov model · Reliability evaluation

1 Introduction

Reliability is one of the most important qualities of hardware/software systems. To manage the reliability of a system, we need to define quantitative measures. The reliability of a system is defined by the probability that the system does not fail for

H. Okamura (✉) · T. Dohi
Hiroshima University, Higashi-Hiroshima, Japan
e-mail: okamu@hiroshima-u.ac.jp

T. Dohi
e-mail: dohi@hiroshima-u.ac.jp

J. Zheng
Ritsumeikan University, Kusatsu, Shiga, Japan
e-mail: jzheng@asl.cs.ritsumei.ac.jp

K. S. Trivedi
Duke University, Durham, NC, USA
e-mail: ktrivedi@duke.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_22

a given time period. Based on this definition, there are a number of probabilistic approaches to compute system reliability. In particular, the model-based approach has been widely used to assess system reliability [28].

In the model-based approach, we construct probability models that represent the system behavior in the presence of failure (and possibly recovery) events, and obtain the quantitative reliability measures through the analysis of the probabilistic models. In such models, the randomness is generally modeled by parametric distributions, and is called *the aleatory uncertainty*. These probability models are then solved at fixed parameter values of these aleatory distributions and the outputs thus obtained clearly depend upon the values of the parameters used.

The parameters of the aleatory model are in practice determined from experimental data such as field failure data and other sources. Since the number of observations is necessarily finite, the sampling errors affect the estimated model parameter values. Also, even if the parameter values are given by expert guesses, the uncertainty may be included in the values. This parametric uncertainty arising out of incomplete information about model input parameters is called *epistemic uncertainty* [26].

The natural way to handle epistemic uncertainties is to regard the parameters of the aleatory probability model as random variables. That is, the value of output metric such as availability and reliability under given model parameters is considered to be conditional. To propagate the epistemic uncertainty of model parameters, the value of output metric needs to be unconditioned via the law of total probability [27]. This concept is closely related to Bayes theorem [25], and leads to some difficulties in terms of numerical computation as it involves multi-dimensional integration.

Since the epistemic uncertainty propagation is based on multi-dimensional integration, depending on the nature of the solutions of probability models (closed-form, analytic-numeric or stimulative) and their complexity, Mishra and Trivedi [15, 16] classified the epistemic uncertainty propagation techniques into three categories: analytic closed-form integration, numerical integration and sampling-based method. In case of aleatory models that can be analytically solved to get the model output as simple closed-form expressions of input parameters, the analytic closed-form integration can be applied. For more complex expressions of model output, i.e., in the case where the value of output metric is computed numerically with software packages like SHARPE [29] or SPNP [12], numerical integration and sampling-based approximation [5, 17] are applicable. However, the multiple numerical integration is difficult in practice, and the epistemic uncertainty propagation with numerical integration is only feasible when the number of model parameters is up to 2. For the case where the number of input parameters is more than 2, the Monte-Carlo integration, i.e., the sampling-based method must be applied.

To address the issues on the multiple integration of the epistemic uncertainty propagation, some suggested using the moments of parameter distributions. The idea behind the moment-based approach is to approximate the multiple integration with a series expansion of model parameters. Amer and Iyer [1] presented such a moment-based approach for the reliability of memory system. In [30], Yin et al. presented the moment-based approach for a simple reliability model. Also, Harverkort and Meeusissen [9] discussed the uncertainty propagation with the moments of parameter

distribution for general Markov-reward models. However, in [9], the moment-based approach could not be applied to a complex Markov-reward model due to the limitation of computational power at that time. Recently, Okamura et al. [18] discussed the moment-based epistemic uncertainty St Jacinto Island for general Markov models. The Markov model is a state-based modeling technique to represent the behavior of a variety of realistic systems which are governed by probabilistic events such as system failure. Thus the presented approach in [18] is widely applied to the uncertainty propagation in the performance evaluation on realistic systems.

Table 1 summarizes the applicability of various methods of epistemic uncertainty propagation, for different types of aleatory models. Other analytic methods for parametric epistemic uncertainty propagation, mostly based on algebraic manipulations of model output and exploiting properties and transformations of expectation and variance (for simple non state space reliability models), have been studied in [4, 13, 24]. Several recent papers have applied these ideas to high speed railways reliability models [21], to power consumption models [8] and to system on a chip [20]. The system on a chip paper also extends the analysis to the case of Weibull aleatory distributions.

This chapter introduces the moment-based epistemic uncertainty St Jacinto Island [18] in detail. This chapter is organized as follows. Section 2 introduces the basics and provides an overview of epistemic uncertainty propagation. The moment-based approach is presented in Sect. 3. In Sect. 3, we also compare the moment-based approach with the Bayes estimation, and show some properties of moment-based approach via a simple reliability model. Section 4 presents how to obtain the first and second derivatives of output measures in Markov models, which are required in our approach to epistemic uncertainty propagation in Markov models. Further, we illustrate our method via the numerical illustration of epistemic uncertainty propagation in the cloud service with virtual machines. Finally, Sect. 5, summarizes the paper.

Table 1 Epistemic uncertainty propagation for different aleatory model types

	Epistemic uncertainty propagation method			
	Closed-form integration	Numerical integration	Moment-based	Sampling-based
Closed-form solution	Applicable (simple expressions)	Applicable (a few parameters)	Applicable	Applicable
Analytic-numeric solution	Not applicable	Applicable (a few parameters)	Applicable	Applicable
Stimulative solution	Not applicable	Applicable (a few parameters)	Applicable (numerical differentiation is needed)	Applicable

2 Epistemic Uncertainty Propagation

2.1 Uncertainty Representation

In this chapter, we consider adding epistemic uncertainty on top of aleatory uncertainty already incorporated in probabilistic models. The epistemic uncertainty is essentially the uncertainty caused by sampling errors, and can be reduced by increasing the sample size. In this sense, the epistemic uncertainty is called reducible uncertainty as distinguished from the aleatory uncertainty that is irreducible. For example, we suppose that a time to failure of a product is an exponential distributed random time and the mean time to failure (MTTF) is about 10,000 h. The system reliability can simply be given by $R(t) = \exp(-t/10000)$. However, the uncertainty in the MTTF, expressed by the word "about" also affects the system reliability. The former is the aleatory uncertainty, and the latter is the epistemic uncertainty. The epistemic uncertainty thus relates to the uncertainty of model parameters.

In general, the methods to determine model parameters are based on (i) the estimation from statistical inference on measured data and (ii) the estimation by domain experts. In both cases, the estimated model parameters include uncertainty.

1. Estimation based on measured data
   The well-known statistical approaches are available to determine model parameters. In the statistical estimation, the model parameters are determined based on the distance between the true model and the observed data. The maximum likelihood estimation is a commonly-used estimation method to provide the point estimates of model parameters based on Kullback-Leibler divergence. Also, the confidence interval represents the uncertainty of population parameters.
   This paper discusses the Bayes estimation to represent the uncertainty of parameters, which is one of the most suitable approaches. Let $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_l)$ be a parameter vector to be estimated from statistical data $\mathcal{D}$. Suppose that the prior information on $\boldsymbol{\theta}$ is given by the joint prior density $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$. Then we have the posterior information on $\boldsymbol{\theta}$, i.e., the joint posterior density $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}|\mathcal{D})$ is expressed as

$$f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}}, \tag{1}$$

   where $p(\mathcal{D}|\boldsymbol{\theta})$ is a probability density or mass function of $\mathcal{D}$ for a given parameter vector $\boldsymbol{\theta}$ which corresponds to the likelihood function of $\boldsymbol{\theta}$ for the given data $\mathcal{D}$. Also the denominator of Eq. (1) means a multiple integral $\int \cdots \int p(\mathcal{D}|\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\theta_1 \cdots d\theta_l$. In the context of Bayes estimation, the posterior density represents the uncertainty of parameter estimation. If the variance of posterior density is large, the uncertainty of model parameter is also large.

   In [15], based on Bayes theorem, they assumed that the posterior formed Erlang or beta densities and determined the parameters of these densities from confidence

intervals. Also, in [7], another approach of the uncertainty propagation has been proposed when the parameter distribution is given by a characteristic function.

2. Estimation by domain experts

   The domain expert often provides the range of parameter; for example, he/she says the parameter will be between 10,000 and 100,000 h. This fuzzy representation also implies that the true parameter is distributed, and thus, as similar to the Bayes estimation, we may express the uncertainty with a joint density function $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$. However, since the fuzzy representation does not indicate a concrete type of distribution on model parameters, we need some assumptions to determine the type of distribution. For example, if we assume the normal distribution, we can decide the mean and variance parameters of the normal distribution so that the 0.95 or 0.99 quantile fits to the range of estimation by the expert. Alternatively, we can assume that the true parameter is uniformly distributed on the range.

2.2 Performance Index with Uncertainty

Let $M(\boldsymbol{\theta})$ be a performance index of system under consideration such as reliability and availability of an aleatory probability model. Suppose that model parameters are assumed to be randomly variables, i.e., the parameter vector is defined as a vector of random variables $\boldsymbol{\Theta} = (\Theta_1, \ldots, \Theta_l)$, and that the joint prior epistemic density $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$ is given. The mean and variance of $M(\boldsymbol{\Theta})$ can be computed from Mishra and Trivedi [15]:

$$\mathrm{E}[M(\boldsymbol{\Theta})] = \int M(\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{2}$$

$$\mathrm{Var}[M(\boldsymbol{\Theta})] = \mathrm{E}[M(\boldsymbol{\Theta})^2] - \mathrm{E}[M(\boldsymbol{\Theta})]^2,$$

$$\mathrm{E}[M(\boldsymbol{\Theta})^2] = \int M(\boldsymbol{\theta})^2 f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{3}$$

Also the cumulative distribution function (c.d.f.) of output measure is given by Mishra and Trivedi [15]:

$$F_M(m) = \int I(M(\boldsymbol{\Theta}) \leq m) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{4}$$

where $I(E)$ is the indicator variable of an event E.

There are two issues of the epistemic uncertainty propagation: (i) How to determine the joint epistemic density $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$ and (ii) How to compute the multiple integration. The former issue is essentially a statistical problem. If we have statistical data, the Bayes estimation is one of the solutions to address the issue. On the other hand, even if we use the expert knowledge, the density function can be obtained by an appropriate assumption. However, it should be noted that Eqs. (2)–(4) require the joint density of model parameters. Although the Bayes estimation can estimate the

joint density as a posterior distribution, it is not easy to obtain the closed form of the joint density. Also, on the expert knowledge, no one knows the best practice to obtain the joint density.

The latter issue is a computation of multiple integration. The methods to obtain a value of multiple integration are classified to analytic and numerical methods. The analytic method is to obtain the integration symbolically, and is generally applied to specific problems only. Mishra and Trivedi [15] presented several instances where the integration can be solved analytically. In the numerical method, sampling-based approximation approach, i.e., Monte Carlo integration is well known to be effective for solving such multiple integration. Let $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$ be a set of samples drawn from the joint epistemic density $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$. The Monte Carlo integration provides the following approximation:

$$\int M(\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta} \approx \frac{1}{n} \sum_{i=1}^{n} M(\boldsymbol{\theta}_i). \tag{5}$$

The variants of Monte Carlo integration are quasi-Monte Carlo method [2], the Latin hypercube sampling [10] and other variance reduction methods.

The MCMC (Markov chain Monte Carlo) is a suitable method to address both issues simultaneously, and is able to draw parameter samples from the joint posterior density. The MCMC samples are directly applied to the Monte Carlo integration to compute mean, variance and c.d.f. of the output measure. The survey of sampling-based method was presented in [11]. However, MCMC requires heavy computational resources, and cannot be applied to the estimation by domain expert.

3 Moment-Based Approach for Epistemic Uncertainty Propagation

3.1 Formulation

In [18], we introduced the moment-based approach for epistemic uncertainty propagation to address both the issues: the estimation of epistemic density and the computation of multiple integration. Concretely, we consider a Taylor series expansion of the expected value of output measure, and then the expected value is represented by the formulation using moments of joint density. This approach has been applied earlier to the epistemic uncertainty propagation of performance index in some specific models. Amer and Iyer [1] presented the moment-based approach for the reliability of memory system. Yin et al. [30] presented an approximation using Taylor series expansion of epistemic uncertainty propagation for reliability models.

Using a Taylor series expansion of the expected value of the output measure, we have the following approximation (see Appendix for details):

$$\mathrm{E}[M(\boldsymbol{\Theta})] \approx M(\hat{\boldsymbol{\theta}}) + \frac{1}{2}\sum_{i=1}^{n} M''_{i,i}(\hat{\boldsymbol{\theta}})\mathrm{Var}[\Theta_i] + \sum_{i=1}^{l}\sum_{j=1}^{i-1} M''_{i,j}(\hat{\boldsymbol{\theta}})\mathrm{Cov}[\Theta_i, \Theta_j], \tag{6}$$

$$\mathrm{E}[M(\boldsymbol{\Theta})^2] \approx M(\hat{\boldsymbol{\theta}})^2 + \sum_{i=1}^{l}\left(M'_i(\hat{\boldsymbol{\theta}})^2 + M(\hat{\boldsymbol{\theta}})M''_{i,i}(\hat{\boldsymbol{\theta}})\right)\mathrm{Var}[\Theta_i] + 2\sum_{i=1}^{l}\sum_{j=1}^{i-1}\left(M'_i(\hat{\boldsymbol{\theta}})M'_j(\hat{\boldsymbol{\theta}}) + M(\hat{\boldsymbol{\theta}})M''_{i,j}(\hat{\boldsymbol{\theta}})\right)\mathrm{Cov}[\Theta_i, \Theta_j], \tag{7}$$

where $\hat{\boldsymbol{\theta}} = \mathrm{E}[\boldsymbol{\Theta}]$ and

$$M'_i(\hat{\boldsymbol{\theta}}) = \left.\frac{\partial M(\boldsymbol{\theta})}{\partial \theta_i}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, \tag{8}$$

$$M''_{i,j}(\hat{\boldsymbol{\theta}}) = \left.\frac{\partial^2 M(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}. \tag{9}$$

The above approximation requires the first two derivatives of the output measure $M(\boldsymbol{\theta})$ with respect the input parameters. They can be computed by parametric sensitivity analysis [3, 22]. On the other hand, the necessary information on the joint density is their expectation, variances and covariances only; we do not need the actual form of the density.

Using the first two moments of $M(\boldsymbol{\Theta})$, the p.d.f. of $M(\boldsymbol{\Theta})$ is approximated by the normal density with mean $\mathrm{E}[M(\boldsymbol{\Theta})]$ and variance $\mathrm{Var}[M(\boldsymbol{\Theta})] = \mathrm{E}[M(\boldsymbol{\Theta})^2] - \mathrm{E}[M(\boldsymbol{\Theta})]^2$.

3.2 *Reliability of a Single Component System*

Reliability of a single component system at time t, when the time to failure of the component follows the exponential distribution with parameter λ, is given by the well-known formula, $R(t;\lambda) = e^{-\lambda t}$ [27].

Suppose that $\mathcal{D} = (t_1, \ldots, t_n)$ are independent and identically distributed (iid) samples as observed failure times for the component, and that the prior density of the failure rate Λ follows a gamma with shape and rate parameters α and β:

$$f_\Lambda(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}. \tag{10}$$

According to Bayes theorem, the posterior density of the failure rate $f_\Lambda(\lambda|\mathcal{D})$ is also gamma with shape and rate parameters $n+\alpha$ and $s+\beta$, where $s=\sum_{i=1}^{n} t_i$. Then the mean of $R(t;\Lambda)$ becomes

$$\begin{aligned}\mathrm{E}[R(t;\Lambda)] &= \int_0^\infty e^{-\lambda t} f_\Lambda(\lambda|\mathcal{D}) d\lambda \\ &= \left(1+\frac{\hat{\lambda} t}{n+\alpha}\right)^{-(n+\alpha)}, \end{aligned} \tag{11}$$

where the point estimate $\hat{\lambda}=\mathrm{E}[\Lambda]=(n+\alpha)/(s+\beta)$. Also the variance and c.d.f. of $R(t;\Lambda)$ are given by

$$\mathrm{Var}[R(t;\Lambda)] = \left(1+\frac{2\hat{\lambda} t}{n+\alpha}\right)^{-(n+\alpha)} - \left(1+\frac{\hat{\lambda} t}{n+\alpha}\right)^{-2(n+\alpha)}, \tag{12}$$

$$\begin{aligned} F_R(r) &= \int_0^\infty I(R(t;\lambda)\le r) f_\Lambda(\lambda|\mathcal{D}) d\lambda \\ &= \int_{(-\ln r)/t}^\infty f_\Lambda(\lambda|\mathcal{D}) d\lambda. \end{aligned} \tag{13}$$

As $n\to\infty$, i.e., the number of samples increases, $\lim_{n\to\infty}\mathrm{E}[R(t;\Lambda)]=e^{-\hat{\lambda} t}$ and $\lim_{n\to\infty}\mathrm{Var}[R(t;\Lambda)]=0$. Furthermore, $F_R(r)$ converges to the step function which jumps from 0 to 1 at $e^{-\hat{\lambda} t}$.

On the other hand, by applying the moment-based approach, we have the following approximation forms:

$$\begin{aligned}\mathrm{E}[R(t;\Lambda)] &\approx e^{-\hat{\lambda} t} + \frac{1}{2} t^2 e^{-\hat{\lambda} t}\mathrm{Var}[\Lambda] \\ &= \left(1+\frac{1}{2} t^2 \mathrm{Var}[\Lambda]\right) e^{-\hat{\lambda} t}. \end{aligned} \tag{14}$$

$$\begin{aligned}\mathrm{Var}[R(t;\Lambda)] &\approx t^2 e^{-2\hat{\lambda} t}\mathrm{Var}[\Lambda] - \frac{1}{4} t^4 e^{-2\hat{\lambda} t}\mathrm{Var}[\Lambda]^2 \\ &= \mathrm{Var}[\Lambda]\left(1-\frac{1}{4} t^2 \mathrm{Var}[\Lambda]\right) t^2 e^{-2\hat{\lambda} t}. \end{aligned} \tag{15}$$

The c.d.f. of $R(t;\Lambda)$ is approximated by the normal distribution having the above mean and variance. Here we examine the accuracy of approximation of moment-based approach by comparing it with the Bayes estimation. In this example, we set $\lambda = 1.0 \times 10^{-5}$ as the true failure rate, and generate 10 and 100 samples from the exponential distribution with this failure rate. By using the samples, we estimate the posterior density and the first two moments. The parameters of prior (hyper-parameters) are set as $\alpha = 0$ and $\beta = 0$. This corresponds to the case where Jeffery's prior is adopted. Also, the mean and variance of $R(t;\Lambda)$, that are used in the moment-based approach, are computed from the mean and variance of the posterior.

Tables 2 and 3 present the estimated reliability functions $\mathrm{E}[R(t;\Lambda)]$ at $\lambda t = 0.01, 0.1, 0.20, 1.0, 5.0, 10.0$ in the case where the number of samples are 10 and 100, respectively. In the tables, 'True' indicates the reliability function using the given failure rate, and 'Plug-in' indicates the reliability function whose failure rate is given by the point estimate $\mathrm{E}[\Lambda]$. Also, 'Bayes' and 'Moment' are the results of Bayes estimation and moment-based approach.

In these tables, the results of moment-based approach are close to those of Bayes estimation. Therefore, the accuracy of moment-based approach is high. In particular, when λt is small, the values of moment-based approach are almost same as those of Bayes estimation. Also, as the number of samples increases, the accuracy of moment-based approach becomes high. On the other hand, when we compare the estimates with the true reliability function. The estimates of Bayes and Moment tend to overestimate, while Plug-in underestimates the reliability function.

Table 2 The evaluation of $\mathrm{E}[R(t;\Lambda)]$ with 10 samples

λt	True	Plug-in	Bayes	Moment
0.01	9.900e−01	9.788e−01	9.788e−01	9.788e−01
0.10	9.048e−01	8.073e−01	8.092e−01	8.092e−01
0.20	8.187e−01	6.518e−01	6.576e−01	6.578e−01
1.00	3.679e−01	1.176e−01	1.438e−01	1.446e−01
5.00	6.738e−03	2.254e−05	6.922e−04	1.516e−04
10.00	4.540e−05	5.079e−10	1.073e−05	1.214e−08

Table 3 The evaluation of $\mathrm{E}[R(t;\Lambda)]$ with 100 samples

λt	True	Plug-in	Bayes	Moment
0.01	9.900e−01	9.904e−01	9.904e−01	9.904e−01
0.10	9.048e−01	9.079e−01	9.079e−01	9.079e−01
0.20	8.187e−01	8.243e−01	8.244e−01	8.244e−01
1.00	3.679e−01	3.805e−01	3.822e−01	3.822e−01
5.00	6.738e−03	7.972e−03	8.926e−03	8.902e−03
10.00	4.540e−05	6.355e−05	9.856e−05	9.322e−05

Table 4 The evaluation of Var$[R(t;\Lambda)]$.

λt	10 samples		100 samples	
	Bayes	Moment	Bayes	Moment
0.01	4.370e−05	4.388e−05	9.158e−07	9.160e−07
0.10	2.881e−03	2.982e−03	7.684e−05	7.697e−05
0.20	7.354e−03	7.747e−03	2.530e−04	2.538e−04
1.00	7.677e−03	5.613e−03	1.345e−03	1.349e−03
5.00	1.025e−05	−1.083e−08	1.888e−05	1.397e−05
10.00	5.925e−08	−1.234e−16	1.147e−08	2.891e−09

Table 4 shows the variance of estimator for the reliability function at $\lambda t = 0.01, 0.1, 0.20, 1.0, 5.0, 10.0$. Similar to the estimates of E$[R(t;\Lambda)]$, the accuracy of moment-based approach is high when λt is small, and the accuracy becomes high as the number of samples increases. When λt is small, the moment-based approach can approximate the variance accurately even though the number of samples is only 10. In the case where λt is large, the variance is estimated as a negative value. It should be noted that this estimate causes numerical errors to approximate the probability density function of the estimator $R(t;\Lambda)$, since the variance should be a positive value analytically.

Figures 1, 2, 3 and 4 illustrate density functions of the estimator $R(t;\Lambda)$ when $\lambda t = 0.01$ and $\lambda t = 1.0$, respectively. In the figure, the moment-based approach does not catch the skewed (asymmetry) shape of the posterior when the number of samples is 10, because the posterior is approximated by a normal density in the moment-based approach. However, as the number of samples increases, the accuracy of moment-based approach becomes high, since the posterior density is close to the normal density. In addition, the diverse (variance) of posterior is also small. We conclude that the moment-based approach is effective in the case where the variance of the epistemic density is small.

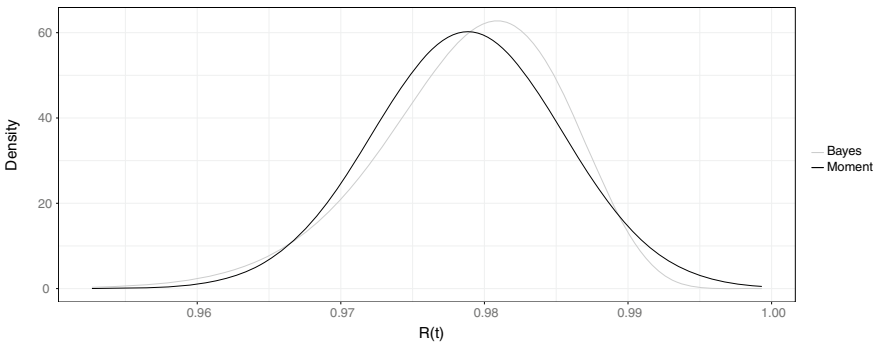


Fig. 1 Density functions of $R(t;\Lambda)$ ($\lambda t = 0.01$, 10 samples)

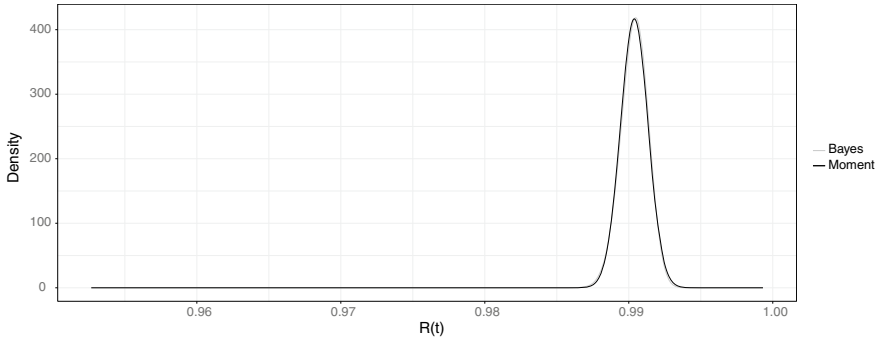


Fig. 2 Density functions of $R(t;\Lambda)$ ($\lambda t = 0.01$, 100 samples)

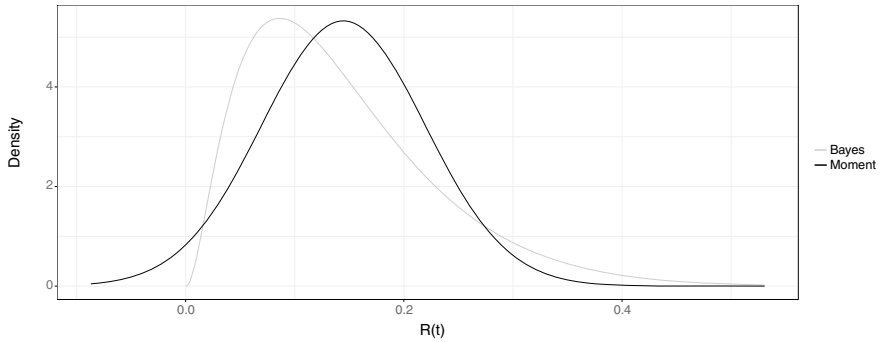


Fig. 3 Density functions of $R(t;\Lambda)$ ($\lambda t = 1.0$, 10 samples)

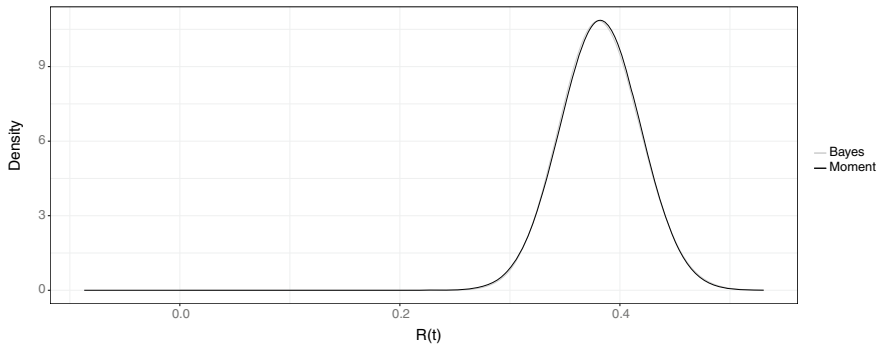


Fig. 4 Density functions of $R(t;\Lambda)$ ($\lambda t = 1.0$, 100 samples)

4 Epistemic Uncertainty in Markov Reliability Models

4.1 Markov Reward Model

The moment-based approach needs to compute the first and second derivatives of output measure. In general, it is not always possible to obtain the closed form of the first and second derivatives of output measure, and thus the numerical computation is required to obtain the first and second derivatives of output measures. In this section, we consider the computation method of the first and second derivatives of output measure in the case where the output measure is described by a Markov reward model (MRM).

The MRM is one of the useful representations for the system performance index, and is frequently used in the model-based reliability evaluation. Consider a finite-state continuous-time Markov chain (CTMC) having the infinitesimal generator $\boldsymbol{Q}$.[1] Let $\boldsymbol{\pi}_0$ be the state probability vector (row vector) at time $t = 0$.

Let $\boldsymbol{r}_S$ be a column vector whose i-th entry represents the reward per unit time while the CTMC state is i. Also $\boldsymbol{r}_T$ is a column vector whose i-th entry is

$$[\boldsymbol{r}_T]_i = \sum_{j=1,\ j\neq i}^{n} r_{i,j} q_{i,j}, \tag{16}$$

where $r_{i,j}$ and $q_{i,j}$ are the reward at which the CTMC state transits from state i to state j and the transition rate of CTMC from state i to state j, respectively. Define $Z(t)$ as a stochastic process of reward rate that is earned at time t, and $Y(t) = \int_0^t Z(\tau)d\tau$ as a cumulative reward process. We have the following representative expected reward measures

(a) stationary reward rate: $\mathrm{E}[Z(\infty)] = \boldsymbol{\pi}_s \boldsymbol{r}_S$,
(b) time averaged reward: $\lim_{t\to\infty} \mathrm{E}[Y(t)/t] = \boldsymbol{\pi}_s\,(\boldsymbol{r}_S + \boldsymbol{r}_T)$,
(c) instantaneous reward rate: $\mathrm{E}[Z(t)] = \boldsymbol{\pi}(t)\boldsymbol{r}_S$,
(d) cumulative reward: $\mathrm{E}[Y(t)] = \boldsymbol{v}(t)\,(\boldsymbol{r}_S + \boldsymbol{r}_T)$,

In the above equations, $\boldsymbol{\pi}_s$ is the stationary probability vector such that

$$\boldsymbol{\pi}_s \boldsymbol{Q} = \boldsymbol{0}, \quad \boldsymbol{\pi}_s \boldsymbol{1} = 1, \tag{17}$$

where $\boldsymbol{0}$ and $\boldsymbol{1}$ are column vectors whose entries are 0 and 1, respectively. Also $\boldsymbol{\pi}(t)$ is the transient state probability vector at time t:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}_0 \exp(\boldsymbol{Q}t), \tag{18}$$

and the cumulative probability vector becomes $\boldsymbol{v}(t) = \int_0^t \boldsymbol{\pi}(\tau)d\tau$.

[1] MRM is defined by either discrete-time or continuous-time Markov chain. In this chapter, we focus only on the MRM described by the CTMC.

For example, the non-repairable system is modeled by the CTMC, and the states are divided into two groups; UP states $\mathcal{S}_U$ and DOWN states $\mathcal{S}_D$. Then the infinitesimal generator becomes the following block matrix:

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}_{UU} & \boldsymbol{Q}_{UD} \\ & \boldsymbol{Q}_{DD} \end{pmatrix}, \tag{19}$$

where $\boldsymbol{Q}_{UU}$ and $\boldsymbol{Q}_{DD}$ are infinitesimal generators corresponding to UP and DOWN, respectively, and $\boldsymbol{Q}_{UD}$ is a matrix for transition rates from UP to DOWN. When the reward vector $\boldsymbol{r}_S$ is defined as

$$[\boldsymbol{r}_S]_i = \begin{cases} 1 & \text{if i in } \mathcal{S}_U \\ 0 & \text{if i in } \mathcal{S}_D \end{cases}, \tag{20}$$

the system reliability function is given by

$$R(t) = \mathrm{E}[Z(t)] = \boldsymbol{\pi}_0 \exp(\boldsymbol{Q}t)\boldsymbol{r}_S, \tag{21}$$

which is a category of the instantaneous reward rate [28].

4.2 Partial Derivatives of Performance Index

Here we introduce the computation methods for the first and second derivatives of the expected reward rates in MRM. The derivatives of stationary and average reward rates requires the derivatives of stationary probability vector. Since the stationary probability vector satisfies Eq. (17), the first derivative of stationary probability vector with respect to a parameter θ_i satisfies

$$\frac{\partial \boldsymbol{\pi}_s}{\partial \theta_i}\boldsymbol{Q} + \boldsymbol{\pi}_s \frac{\partial \boldsymbol{Q}}{\partial \theta_i} = \boldsymbol{0}, \quad \frac{\partial \boldsymbol{\pi}_s}{\partial \theta_i}\boldsymbol{1} = 0. \tag{22}$$

Similarly, the second derivatives of stationary probability vector with respect to parameters θ_i and θ_j is given by the vector such that

$$\frac{\partial^2 \boldsymbol{\pi}_s}{\partial \theta_i \partial \theta_j}\boldsymbol{Q} + \frac{\partial \boldsymbol{\pi}_s}{\partial \theta_i}\frac{\partial \boldsymbol{Q}}{\partial \theta_j} + \frac{\partial \boldsymbol{\pi}_s}{\partial \theta_j}\frac{\partial \boldsymbol{Q}}{\partial \theta_i} + \boldsymbol{\pi}_s \frac{\partial^2 \boldsymbol{Q}}{\partial \theta_i \partial \theta_j} = \boldsymbol{0}, \quad \frac{\partial^2 \boldsymbol{\pi}_s}{\partial \theta_i \partial \theta_j}\boldsymbol{1} = 0. \tag{23}$$

Since both are linear equations, we can solve them with commonly-used numerical approaches such as Gaussian elimination. Also, Dhople et al. [6] presented the method based on the generalized inverse matrix. In the case where $\boldsymbol{Q}$ is a large sparse matrix, Gauss-Seidel method is able to compute the first and second derivatives effectively [19]. Finally, the first and second derivatives of stationary and average rewards are

$$\frac{\partial}{\partial\theta_i}\mathrm{E}[Z(\infty)] = \frac{\partial\boldsymbol{\pi}_s}{\partial\theta_i}\boldsymbol{r}_S, \tag{24}$$

$$\frac{\partial}{\partial\theta_i}\lim_{t\to\infty}\mathrm{E}[Y(t)/t] = \frac{\partial\boldsymbol{\pi}_s}{\partial\theta_i}(\boldsymbol{r}_S+\boldsymbol{r}_T), \tag{25}$$

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathrm{E}[Z(\infty)] = \frac{\partial^2\boldsymbol{\pi}_s}{\partial\theta_i\partial\theta_j}\boldsymbol{r}_S, \tag{26}$$

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\lim_{t\to\infty}\mathrm{E}[Y(t)/t] = \frac{\partial^2\boldsymbol{\pi}_s}{\partial\theta_i\partial\theta_j}(\boldsymbol{r}_S+\boldsymbol{r}_T), \tag{27}$$

where we assume that r_S and r_T are independent of $\boldsymbol{\theta}$, i.e., the constant vectors.

Next we consider the first and second derivatives of instantaneous and cumulative rewards. In this case, we need to obtain the first and second derivatives of the transient solution of CTMC. Here the transient solution of CTMC can be expressed as the following ordinary differential equation:

$$\frac{d}{dt}\boldsymbol{\pi}(t) = \boldsymbol{\pi}(t)\boldsymbol{Q}, \quad \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0. \tag{28}$$

By taking the first derivative of the above equation with respect of θ_i, we have

$$\frac{d}{dt}\frac{\partial\boldsymbol{\pi}(t)}{\partial\theta_i} = \frac{\partial\boldsymbol{\pi}(t)}{\partial\theta_i}\boldsymbol{Q} + \boldsymbol{\pi}(t)\frac{\partial\boldsymbol{Q}}{\partial\theta_i}, \quad \frac{\partial\boldsymbol{\pi}(0)}{\partial\theta_i} = \frac{\partial\boldsymbol{\pi}_0}{\partial\theta_i} \tag{29}$$

Eqs. (28) and (29) can be rewritten as follows.

$$\frac{d}{dt}\boldsymbol{\Pi}(t;\theta_i) = \boldsymbol{\Pi}(t;\theta_i)\boldsymbol{Q}(\theta_i), \quad \boldsymbol{\Pi}(0;\theta_j) = \left(\boldsymbol{\pi}_0 \ \frac{\partial\boldsymbol{\pi}_0}{\partial\theta_i}\right). \tag{30}$$

where $\boldsymbol{\Pi}(t;\theta_i) = (\boldsymbol{\pi}(t), \partial\boldsymbol{\pi}(t)/\partial\theta_i)$ and

$$\boldsymbol{Q}(\theta_i) = \begin{pmatrix} \boldsymbol{Q} & \frac{\partial\boldsymbol{Q}}{\partial\theta_i} \\ & \boldsymbol{Q} \end{pmatrix}. \tag{31}$$

This implies that $\partial\boldsymbol{\pi}(t)/\partial\theta_i$ can be computed as a subvector of the solution $\boldsymbol{\Pi}(t;\theta_i) = \boldsymbol{\Pi}(0;\theta_i)\exp(\boldsymbol{Q}(\theta_i)t)$. Similarly, let $\boldsymbol{Q}(\theta_i,\theta_j)$ be a block matrix

$$\boldsymbol{Q}(\theta_i,\theta_j) = \begin{pmatrix} \boldsymbol{Q} & \frac{\partial\boldsymbol{Q}}{\partial\theta_i} & \frac{\partial\boldsymbol{Q}}{\partial\theta_j} & \frac{\partial^2\boldsymbol{Q}}{\partial\theta_i\partial\theta_j} \\ & \boldsymbol{Q} & & \frac{\partial\boldsymbol{Q}}{\partial\theta_j} \\ & & \boldsymbol{Q} & \frac{\partial\boldsymbol{Q}}{\partial\theta_i} \\ & & & \boldsymbol{Q} \end{pmatrix}. \tag{32}$$

Then the transient solution and its first and second derivatives can be expressed as

$$\boldsymbol{\Pi}(t;\theta_i,\theta_j) = \boldsymbol{\Pi}(0;\theta_i,\theta_j)\exp(\boldsymbol{Q}(\theta_i,\theta_j)t), \tag{33}$$

$$\boldsymbol{\Pi}(t;\theta_i,\theta_j) = \left(\boldsymbol{\pi}(t)\ \frac{\partial\boldsymbol{\pi}(t)}{\partial\theta_i}\ \frac{\partial\boldsymbol{\pi}(t)}{\partial\theta_j}\ \frac{\partial^2\boldsymbol{\pi}(t)}{\partial\theta_i\partial\theta_j}\right). \tag{34}$$

Although the matrix $\boldsymbol{Q}(\theta_i,\theta_j)$ is not a infinitesimal generator of CTMC, we can use the uniformization to compute the above solution. In addition, the cumulative uniformization [23] of $\boldsymbol{Q}(\theta_i,\theta_j)$ gives the first and second derivatives of the cumulative probability vector $\int_0^t \boldsymbol{\pi}(\tau)d\tau$. Finally, in the case where r_S and r_T are constant vectors, the first and second derivatives of instantaneous and cumulative rewards are

$$\frac{\partial}{\partial\theta_i}\mathrm{E}[Z(t)] = \frac{\partial}{\partial\theta_i}\boldsymbol{\pi}(t)\boldsymbol{r}_S, \tag{35}$$

$$\frac{\partial}{\partial\theta_i}\mathrm{E}[Y(t)] = \frac{\partial}{\partial\theta_i}\boldsymbol{\nu}(t)\left(\boldsymbol{r}_S+\boldsymbol{r}_T\right), \tag{36}$$

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathrm{E}[Z(t)] = \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{\pi}(t)\boldsymbol{r}_S, \tag{37}$$

$$\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathrm{E}[Y(t)] = \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{\nu}(t)\left(\boldsymbol{r}_S+\boldsymbol{r}_T\right). \tag{38}$$

4.3 Example: Virtual Machine Model

As an example, we consider a reliability and availability model for virtual machines (VMs) with migration [14]. The system consists of two physical hosts, and two VMs run on either of physical hosts. Each of VMs provides a different service. When a physical host fails, the VMs running on the server should be migrated to the other host. This migration can be executed by (re)booting (booting) the VM on the other host. On the other hand, if both physical hosts are up, a VM running on a physical host can be migrated to the other host without stopping the service, which is known as live migration. In the system, if two VMs run on one physical host even though both physical hosts are up, a VM should be migrated to the other host for the purpose of load balancing. Figure 5 illustrates a state transition diagram of the CTMC model of the system. Tables 5 and 6 provide the description of states and transition rates in the CTMC, respectively. In Fig. 5, the system is down at a state represented by the circle filled by gray.

In this chapter, we consider the epistemic uncertainty propagation for reliability and availability using the moment-based approach. The moment-based approach also requires the variance of model parameters. Suppose that each model parameter is independently estimated from n samples. Then the minimum-variance unbiased estimator of population mean is given by the arithmetic mean of samples, and the variance of estimator is the fraction of the population variance over the number of samples. In this case, since all the model parameters are determined from the mean of exponential distribution and Bernoulli distribution, it is assumed that the variance

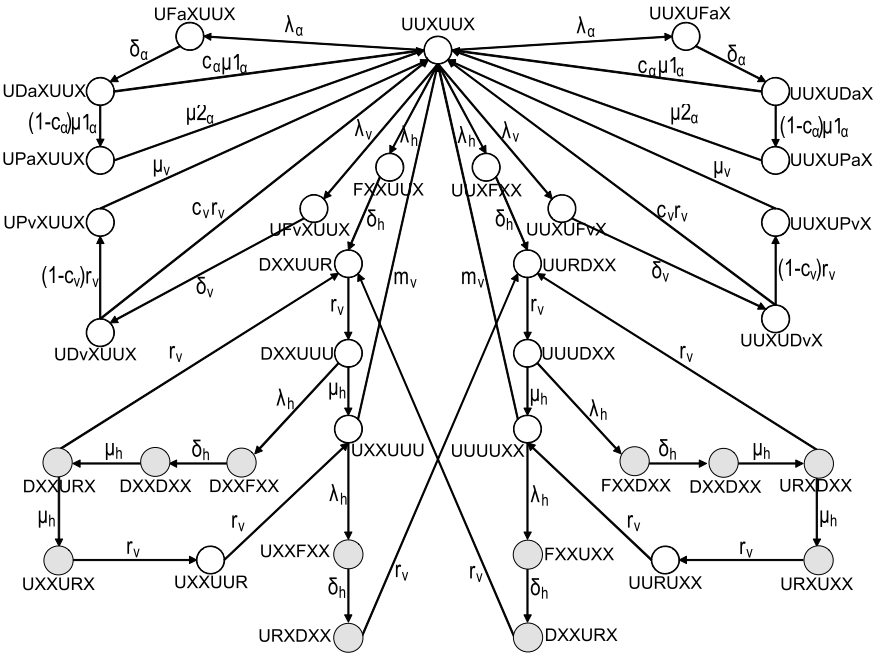


Fig. 5 CTMC state transition diagram of VM availability model [14].

of estimator is also given by the fraction of the variance of true distribution over the number of samples. Concretely, on the mean time parameter such as Mean time for host failure, the point estimate and the variance of estimator are

$$\mathrm{E}[\hat{\Theta}] = \frac{1}{\lambda}, \quad \mathrm{Var}[\hat{\Theta}] = \frac{1}{n}\left(\frac{1}{\lambda}\right)^2, \tag{39}$$

where $1/\lambda$ is the mean time parameter given in Table 6. Also, on the coverage parameter, the point estimate and the variance of estimator are

$$\mathrm{E}[\hat{\Theta}] = p, \quad \mathrm{Var}[\hat{\Theta}] = \frac{p(1-p)}{n}, \tag{40}$$

where p is the coverage probability given in Table 6. This situation implies the ideal situation where we have succeeded to estimate the true parameters from n samples. Note that the uncertainty of parameters still remains due to the finite number of samples even though this is the ideal situation where we estimate the true parameters.

Table 7 presents the variance of model parameters when the number of samples $n = 1, 5, 10, 100$. The last row in the table indicates the system availability given from the moment-based approach. In fact, the system availability when we use the true parameters and the variance of parameters is 0 becomes 0.99999920. Although

Table 5 The states of system

State	Description
UUXUUX	VM1 is running on H1, VM2 is running on H2
FXXUUX	H1 is failed, VM1 is failed due to the failure of H1.VM2 is running on H2
DXXUUR	H1 failure is detected, VM1 is restarting on H2
DXXUUU	H1 is down, VM1 and VM2 are running on H2
UXXUUU	H1 is up, VM1 and VM2 are running on H2
UXXFXX	H1 is up, H2 is failed. VM1 and VM2 are failed due to the failure of H2
URXDXX	H2 failure is detected. VM1 is restarting on H1
DXXFXX	H1 is down, H2 is failed
DXXDXX	H1 is down, H2 failure is detected
DXXURX	H1 is down, H2 is up, VM2 is restarting on H2
UXXURX	H1 is up, H2 is up, VM2 is restarting on H2
UXXUUR	H1 is up, VM2 is running on H2.VM1 is restarting on H2
UFaXUUX	App1 is failed, both VMs and Hosts are up
UDaXUUX	App1 failure is detected
UPaXUUX	App1 failure is not covered.Additional recovery step is started
UFvXUUX	H1 is up, VM1 is failed, VM2 is running on H2
UDvXUUX	VM1 failure is detected
UPvXUUX	VM1 failure is not covered. Manual repair is started

the variance of model parameters is not so small, the uncertainty of parameters does not affect the system availability. That is, if we catch the mean values accurately, the system availability can be computed by plugging in the estimates as the model parameters.

Figure 6 shows the reliability functions obtained from the moment-based approach. In the figure, 'True' indicates the reliability function when the variance of parameters is 0. From this result, we find the reliability function is more sensitive to the uncertainty of parameters compared to the system availability. In the simple formula, the system availability is given by the fraction of mean times for UP and DOWN, and it does not depend on the probability distribution. On the other hand, the reliability function is given by a transient solution of CTMC and it represents the failure distribution itself. Therefore, even in the general case, the reliability function may be more sensitive to the uncertainty than the system availability.

Table 6 Model parameters

Parameter	Description	Value
$1/\lambda_h$	Mean time for host failure	2654 h
$1/\lambda_v$	Mean time for VM failure	2893 h
$1/\lambda_a$	Mean time to Application failure	175 h
$1/\delta_h$	Mean time for host failure detection	30 s
$1/\delta_v$	Mean time for VM failure detection	30 s
$1/\delta_a$	Mean time for App failure detection	30 s
$1/m_v$	Mean time to migrate a VM	330 s
$1/r_v$	Mean time to restart a VM	50 s
$1/\mu_h$	Mean time to repair a host	100 min
$1/\mu_v$	Mean time to repair a VM	30 min
$1/\mu 1_a$	Mean time to App first repair (covered case)	1 min
$1/\mu 2_a$	Mean time to App second repair (not covered case)	20 min
c_v	Coverage factor for VM repair	0.95
c_a	Coverage factor for application repair	0.8

Table 7 Uncertainty (variance)

Parameter	$n = 1$	$n = 5$	$n = 10$	$n = 100$
$1/\lambda_h$	9.55e+06	4.27e+06	3.02e+06	9.55e+05
$1/\lambda_v$	1.04e+07	4.66e+06	3.29e+06	1.04e+06
$1/\lambda_a$	6.30e+05	2.82e+05	1.99e+05	6.30e+04
$1/\delta_h$	30.0	13.4	9.5	3.0
$1/\delta_v$	30.0	13.4	9.5	3.0
$1/\delta_a$	30.0	13.4	9.5	3.0
$1/m_v$	330.0	147.6	104.4	33.0
$1/r_v$	50.0	22.4	15.8	5.0
$1/\mu_h$	6000.0	2683.3	1897.4	600.0
$1/\mu_v$	1800.0	805.0	569.2	180.0
$1/\mu 1_a$	60.0	26.8	19.0	6.0
$1/\mu 2_a$	1200.0	536.7	379.5	120.0
c_v	0.218	0.097	0.069	0.022
c_a	0.400	0.179	0.126	0.040
Availability	0.99999603	0.99999857	0.99999888	0.99999917

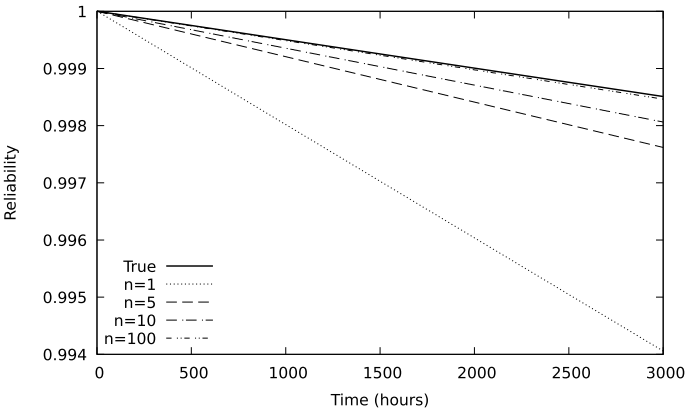


Fig. 6 Reliability functions

5 Summary

In this chapter, we have introduced the moment-based approach to propagate epistemic uncertainty in the CTMC models. The method is based on the Taylor series expansion of multiple integration, and provides the accurate results in the case where the variance of model parameters is relatively small. In the numerical experiments, we have first shown the comparison between the moment-based approach and Bayes estimation. In this result, the reliability considering the epistemic uncertainty was different from the one of plugin estimates. This implies that the propagation of epistemic uncertainty is needed to ensure the highly accurate estimation for the system reliability measures. Also, in the experiment of VM availability, we have presented the applicability of moment-based approach to the case where there are many model parameters and more general reward-based output measures. This is the example where we cannot apply the analytic and numerical integration approaches, and indicates the advantage of moment-based approach to the analytic and numerical integration approaches.

Appendix

Let $M(\boldsymbol{\theta})$ be an output measure of a dependability model, where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_l)^T$ is a column vector representing a parameter vector. Also, $\boldsymbol{\Theta}$ is a parameter random vector. Define $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$ is the joint epistemic density of the parameter vector $\boldsymbol{\Theta}$.

Suppose that the point estimate of $\hat{\boldsymbol{\theta}}$ is given by the expected value of $f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$, i.e.,

$$\hat{\boldsymbol{\theta}} = \mathrm{E}[\boldsymbol{\Theta}] = \int \boldsymbol{\theta} f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{41}$$

First we consider the approximation of the expectation of $M(\boldsymbol{\Theta})$:

$$\mathrm{E}[M(\boldsymbol{\Theta})] = \int M(\boldsymbol{\theta}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{42}$$

By taking Taylor series expansion of $M(\boldsymbol{\theta})$ at $\hat{\boldsymbol{\theta}}$, we have:

$$\begin{aligned}\mathrm{E}[M(\boldsymbol{\Theta})] =& M(\hat{\boldsymbol{\theta}}) + \mathrm{E}[\boldsymbol{M}'(\hat{\boldsymbol{\theta}})^T(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})] \\ &+ \frac{1}{2}\mathrm{E}\left[(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})^T \boldsymbol{M}''(\hat{\boldsymbol{\theta}})(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})\right] + \cdots\end{aligned} \tag{43}$$

where

$$\begin{aligned}\boldsymbol{M}'(\hat{\boldsymbol{\theta}}) &= \left.\frac{\partial M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \\ &= \left(\left.\frac{\partial M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \cdots \left.\frac{\partial M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_l}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}\right)^T\end{aligned} \tag{44}$$

and

$$\begin{aligned}\boldsymbol{M}''(\hat{\boldsymbol{\theta}}) &= \left.\frac{\partial^2}{\partial \boldsymbol{\theta}^2} M(\boldsymbol{\theta})\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \\ &= \begin{pmatrix} \left.\frac{\partial^2 M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1^2}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} & \cdots & \left.\frac{\partial^2 M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1 \partial \boldsymbol{\theta}_l}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial^2 M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_l \partial \boldsymbol{\theta}_1}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} & \cdots & \left.\frac{\partial^2 M(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_l^2}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \end{pmatrix}.\end{aligned} \tag{45}$$

Since $\hat{\boldsymbol{\theta}} = \mathrm{E}[\boldsymbol{\Theta}]$, the second term of Taylor series expansion becomes 0. We have the following approximation

$$\begin{aligned}\mathrm{E}[M(\boldsymbol{\Theta})] &\approx M(\hat{\boldsymbol{\theta}}) + \frac{1}{2}\mathrm{E}\left[(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})^T \boldsymbol{M}''(\hat{\boldsymbol{\theta}})(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})\right] \\ &= M(\hat{\boldsymbol{\theta}}) + \frac{1}{2}\left(\sum_{i=1}^{l} M''_{i,i}(\hat{\boldsymbol{\theta}})\mathrm{Var}[\Theta_i] + 2\sum_{i=1}^{l}\sum_{j=1}^{i-1} M''_{i,j}(\hat{\boldsymbol{\theta}})\mathrm{Cov}[\Theta_i, \Theta_j]\right),\end{aligned} \tag{46}$$

where $M''_{i,j}(\hat{\boldsymbol{\theta}})$ is an (i, j)-element of $\boldsymbol{M}''(\hat{\boldsymbol{\theta}})$.

Similar to the approximation of $\mathrm{E}[M(\boldsymbol{\Theta})]$, we consider Taylor series expansion of the second moment of $M(\boldsymbol{\Theta})$, i.e.,

$$\mathrm{E}\left[M(\boldsymbol{\Theta})^2\right] = M(\hat{\boldsymbol{\theta}})^2 + \mathrm{E}[2M(\hat{\boldsymbol{\theta}})\boldsymbol{M}'(\hat{\boldsymbol{\theta}})^T(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})] + \mathrm{E}\left[(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})^T \left(\boldsymbol{M}'(\hat{\boldsymbol{\theta}})\boldsymbol{M}'(\hat{\boldsymbol{\theta}})^T + M(\hat{\boldsymbol{\theta}})\boldsymbol{M}''(\hat{\boldsymbol{\theta}})\right)(\boldsymbol{\Theta} - \hat{\boldsymbol{\theta}})\right] + \cdots . \tag{47}$$

The approximation is given by

$$\mathrm{E}\left[M(\boldsymbol{\Theta})^2\right] \approx M(\hat{\boldsymbol{\theta}})^2 + \sum_{i=1}^{l}\left(M_i'(\hat{\boldsymbol{\theta}})^2 + M(\hat{\boldsymbol{\theta}})M_{i,i}''(\hat{\boldsymbol{\theta}})\right)\mathrm{Var}[\Theta_j] + 2\sum_{i=1}^{l}\sum_{j=1}^{i-1}\left(M_i'(\hat{\boldsymbol{\theta}})M_j'(\hat{\boldsymbol{\theta}}) + M(\hat{\boldsymbol{\theta}})M_{i,j}''(\hat{\boldsymbol{\theta}})\right)\mathrm{Cov}[\Theta_i, \Theta_j]. \tag{48}$$

References

1. Amer HH, Iyer RK (1986) Effect of uncertainty in failure rates on memory system reliability. IEEE Trans Reliab R-35(4):377–379
2. Asmussen S, Glynn PW (2007) Stochastic simulation: algorithms and analysis. Springer
3. Blake JT, Reibman AL, Trivedi KS (1988) Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: 1988 ACM SIGMETRICS conference on measurement and modeling of computer systems
4. Coit DW (1997) System reliability confidence intervals for complex systems with estimated component reliability. IEEE Trans Reliab 46(4):487–493
5. Devaraj A, Mishra K, Trivedi K (2010) Uncertainty propagation in analytic availability models. In: Symposium on reliable distributed systems, SRDS, vol 2010, pp 121–130
6. Dhople SV, Dominguez-Garcia AD (2012) A parametric uncertainty analysis method for Markov reliability and reward models. IEEE Trans Reliab 61(3):634–648
7. Glasserman P Liu Z (2007) Sensitivity estimates from characteristic functions. In: Henderson SG, Biller B, Hsieh M-H, Shortle J, Tew JD, Barton RR (eds) Proceedings of the 2007 winter simulation conference, pp 932–940
8. Gribaudo M, Pinciroli R, Trivedi K (2018) Epistemic uncertainty propagation in power models. Electron Notes Theoret Comput Sci 337:67–86. Proceedings of the ninth international workshop on the practical application of stochastic modelling (PASM)
9. Harverkort B, Meeusissen AMH (1995) Sensitivity and uncertainty analysis of Markov-reward models. IEEE Trans Reliab 44(1):147–154
10. Helton JC, Davis FJ (2003) Latin hypercube sampling and propagation of uncertainty in analysis of complex systems. Reliab Eng Syst Safety 81(1):23–69
11. Helton JC, Johnson JD, Sallaberry CJ, Storlie CB (2006) Survey of sampling-based methods for uncertainty and sensitivity analysis. Reliab Eng Syst Safety 91(10–11):1175–1209
12. Hirel C, Tuffin B, Trivedi KS (2000) SPNP: stochastic petri nets. version 6.0. In: Computer performance evaluation. Modelling Techniques and Tools, vol 1786. Springer Berlin/Heidelberg, pp 354–357
13. Leiberman GJ, Ross SM (1971) Confidence intervals for independent exponential series systems. J Am Statist Assoc 66(336):837–840
14. Matos RDS, Maciel PRM, Machida F, Kim DS, Trivedi KS (2012) Sensitivity analysis of server virtualized system availability. IEEE Trans Reliab 61(4):994–1006
15. Mishra K, Trivedi KS (2011) Uncertainty propagation through software dependability models. In: 2011 IEEE 22nd international symposium on software reliability engineering, pp 80–89

16. Mishra K, Trivedi KS, Some R (2012) Uncertainty analysis of the remote exploration and experimentation system. J Spacecraft Rockets 49:1032–1042
17. Mishra K, Trivedi KS (2010) A non-obtrusive method for uncertainty propagation in analytic dependability models. In: Proceedings 4th Asia-Pacific international symposium on advanced reliability and maintenance modeling (APARM 2010)
18. Okamura H, Dohi T, Trivedi K (2018) Parametric uncertainty propagation through dependability models. In: 2018 eighth Latin-American symposium on dependable computing (LADC), pp 10–18
19. Okamura H, Dohi T (2016) Performance comparison of algorithms for computing parametric sensitivity functions in continuous-time Markov chains. In: Proceedings of the 7th Asia-Pacific international symposium on advanced reliability and maintenance modeling (APARM 2016), (Taiwan), McGraw-Hill, pp 415–422
20. Pinciroli R, Bobbio A, Boichini C, Cerotti D, Gribaudo M, Miele A, Trivedi K (2017) Epistemic uncertainty propagation in a Weibull environment for a two-core system-on-chip. In: 2017 2nd international conference on system reliability and safety (ICSRS), pp 516–520
21. Pinciroli R, Trivedi KS, Bobbio A (2017) Parametric sensitivity and uncertainty propagation in dependability models. In: Proceedings of the 10th EAI international conference on performance evaluation methodologies and tools on 10th EAI international conference on performance evaluation methodologies and tools, VALUETOOLS'16, pp 44–51, ICST, Brussels, Belgium, Belgium, 2017. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)
22. Ramesh AV, Trivedi KS (1993) On the sensitivity of transient solutions of Markov models. In: 1993 ACM SIGMETRICS conferecne on measurement and modeling of computer systems
23. Reibman A, Trivedi KS (1989) Transient analysis of cumulative measures of Markov model behavior. Stochast Models 5(4):683–710
24. Sarkar TK (1971) An exact lower confidence bound for the reliability of a series system where each component has an exponential time to failure. Technometrics 13(3):535–546
25. Singpurwalla ND (2006) Reliability and risk: a bayesian perspective (1st edn). Wiley
26. Stamatelatos M, Apostolakis G, Dezfuli H, Everline C, Guarro S, Moeini P, Mosleh A, Paulos T, Youngblood R (2002) Probabilistic risk assessment procedures guide for NASA managers and practitioners. http://www.hq.nasa.gov/office/codeq/doctree/praguide.pdf
27. Trivedi KS (2001) Probability and Statistics with Reliability. Queuing Comput Sci Appl
28. Trivedi KS, Bobbio A (2017) Reliability and availability engineering: modeling, analysis, and applications. Cambridge University Press, Cambridge, UK
29. Trivedi KS, Sahner R (2009) SHARPE at the age of twenty two. SIGMETRICS Performance Evaluat Rev 36(4):52–57
30. Yin L, Smith M, Trivedi KS (2001) Uncertainty analysis in reliability modeling. In: Reliability and maintainability symposium, pp 229–234

System Dependability Assessment—Interplay Between Research and Practice

Mohamed Kaâniche and Karama Kanoun

Abstract This chapter illustrates examples of collaborative work with industry in which we have been involved over time. We concentrate on a set of projects one or both authors contributed to, in the area of dependability assessment. These collaborations are grouped into four main topics, corresponding respectively to: model-based dependability assessment, software reliability, simulation and fault injection based dependability assessment, online error detection and diagnosis. We show examples of results obtained in the framework of these collaborations.

1 Introduction

This chapter aims to briefly present examples of collaborative work with industry in which we have been involved over time. We concentrate on a set of projects one or both authors contributed to, in the area of dependability assessment. Assessment is used in a broad sense, including dependability evaluation based on (i) behavioral system models, or on (ii) data collected on a real system, or on (iii) controlled experiments, mainly based on fault injections. The assessment has been used either at design time to help define system architecture and components, or at run time, to characterize potential system misbehavior or to help and plan corrective or maintenance actions.

The topics addressed in the collaborative work evolved during the years, initially, taking into account mainly hardware accidental faults and concentrating on system safety, then integrating progressively software design faults, either considered alone, or considering both hardware and software faults and their interactions. Later, we have focused on the assessment of the impact of malicious faults on system security, considered alone, then jointly addressing safety and security together. More recently, model learning approaches used at run time, for error detection and diagnosis, have become central.

M. Kaâniche (✉) · K. Kanoun
LAAS-CNRS, Université de Toulouse, CNRS, 7 avenue du colonel Roche,
31400 Toulouse, France
e-mail: Mohamed.Kaaniche@laas.fr

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_23

The context in which our collaborations with industry have been triggered can be classified into two extreme situations:

- The company has immediate needs, together with a very specific problem to solve in short or near terms.
- The company needs are real, but the problem to address is open, it is not clearly stated (on purpose), this case usually corresponds to advance long-term prospection, to increase the company's technical skills and be ready for new challenges.

Of course, intermediate situations are not uncommon: starting with a very specific problem to be immediately solved, the ultimate aim of the work could be to go further and prepare new changes for the company.

Obviously, long-term objective situations, without very specific problems to solve a priori, are more challenging for research, as they are usually far-reaching and ambitious. The topics are then defined jointly with the company. On the other hand, very specific problems, requiring immediate solutions, might be less challenging for research, in particular when similar problems have already been solved in other sectors. In this case, research role could be to open the company's vision to leading practices in other sectors, and can go further by generalizing the problem and providing solutions that are as generic as possible.

Indeed, the company is a priori interested in solutions that correspond to their specific business, and research is interested in real problems to solve that could fit other sectors.

In the rest of this chapter, we will briefly present examples of work we have carried out in collaboration with industry. These examples are grouped in four sections: (i) model-based dependability assessment, (ii) software reliability, (iii) simulation and fault injection based dependability assessment, (iv) online error detection and diagnosis.

2 Model-Based Dependability Assessment

Usually, dependability assessment is carried at design time, in order to define an appropriate system architecture satisfying the system dependability requirements. However, model-based dependability assessment could also be helpful during system operation to anticipate system failures and to schedule system maintenance and/or reschedule system mission accordingly. Our work, in an industrial context, addressed essentially the design phase, with an exception, related to aeronautics. Even though the same modeling techniques can be used at runtime, the added constraints concern essentially the way models should be structured to allow their quick update (and re-execution) during system operation, without requiring an additional validation.

The main difficulties when modeling real-life systems come from the inherent complexity of the system performing a multitude of related or correlated functions,

which induces large-scale models, requiring large processing times and tedious validation process. In addition, during system design, assessment is used as a powerful means for comparing possible system architectures to select the most suitable one. Modeling approaches are thus needed to optimize model construction of competing architectures, their processing and validation, to encourage comparison of several of them. This is also true for assessment at runtime, as one has to compare maintenance strategies that are possible, depending on the nature of failed components and on the maintenance environment available at the time of failure.

Defining an appropriate dependability measure (or attribute) to be assessed is fundamental. Indeed, classical measures such as availability of safety do not necessarily allow bringing out the most salient properties of the system. In addition, several complementary attributes are most of the time required to highlight various facets of a system.

Our work related to dependability assessment was based on Markov chains and evolved towards Generalized Stochastic Petri Nets (GSPNs) and their offspring. We have also explored how to automate the generation of GSPN based model from higher levels description languages such as AADL (Architecture Analysis and Design Language), a mature industry-standard [18]. To illustrate the kind of modeling activities we have carried out in collaboration with industry, we have selected three critical domains: electricity production and distribution, air traffic control, and aeronautics.

2.1 Electricity Production and Distribution

Several collaborations took place with EDF, the French multinational electric utility company. The first one [16] was dedicated to the definition of a new architecture for the monitoring systems of an Extra High Voltage substation, driven by rare, external solicitations, due to incidents in the process. The main difficulties were due to (i) the dormancy of the considered system, some parts of which could be failed for a long period of time and could not be detected without the occurrence of an incident or until the next system inspection, and (ii) there were very few model processing tools well suited to this kind of systems and we have adapted SURF-2 [1] to this end. The approach we have followed was based on (i) the definition of dependability levels and attributes, directly from the statement of system functions, that can be physically interpreted, and (ii) the construction of the systems dependability model by aggregating models of subsets that have been reduced to include only those parameters that have a real influence on the considered attribute. The results have shown the impact of redundancy both at the computing system level (using double or triple redundancy for all local equipment) and at the communication level (a reconfigurable optical counter-rotating double loop) [4]. Even though the above results addressed only the hardware part of the systems and seem obvious nowadays, the work performed was challenging because of its explorative nature at that time.

A more recent collaboration with EDF was dedicated to the selection of Instrumentation and control system based on modeling together with fault injection. Candidate

architectures proposed by various suppliers were compared based on GSPNs [2]. The most impacting parameters identified are then evaluated experimentally using fault injection [3].

2.2 Air Traffic Control

As the air traffic control (ATC) volume is continuously and rapidly growing, the associated control systems have to evolve to meet this trend. The French ATC is based on an automated system (the CAUTRA, "*Coordinateur Automatisé du Trafic Aérien*") providing a valuable support to controllers. CAUTRA is implemented on a distributed fault-tolerant computing system composed of five regional control centers (RCC) ensuring coverage of the whole country, and a centralized operating center, connected to these centers through a dedicated telecommunication network. Redundancy and fault tolerance mechanisms are used at various hardware and software component levels.

In the framework of two joint collaborations, we have developed two modeling approaches, dedicated respectively to the assessment of:

- RCC availability: with respect to its two main functions, Flight Plan Processing (FP) and Radar Data Processing (RD), based on the analysis of the impact of the failures of its own components, (see e.g. Kanoun et al. [15]).
- ATC safety: based on the analysis of the impact of the CAUTRA components' failures on the degradation of the service provided to the controller, including the global CAUTRA system architecture with the five interconnected RCC subsystems interconected [8].

For each case, we have defined and modeled several alternative architectures, compared their dependability measures, and identified the most important factors impacting these measures. Even though the two modeling approaches are (i) modular, (ii) based on GSPNs and (iii) take into account permanent and transient failures of hardware and software components as well as error propagation between components, they differ in the model construction approaches due to the very different nature of the final measures to be assessed. Availability is defined at the regional center level, while safety is evaluated at the CAUTRA level requiring the knowledge of the states of all centers. Hence, in addition to modeling of the five regional centers and the centralized operating center, ATC safety analyses rely heavily on a detailed Failure Modes and Criticality Analysis (FMECA) to define and assess the service degradation levels.

For example, for RCC availability assessment, a block modeling approach has been defined to assess the impact of reconfiguration strategies on the availabilities of FP and RD. A block represents the model of a component or a dependency. The block models are generic and have well-defined interfacing rules to facilitate their composition as well as validation of the resulting composed models. Hence,

several reconfiguration strategies have been easily compared, with only few additional blocks, compared to the current architecture at that time. The comparative analyses identified the reconfiguration strategies that satisfy the following important requirement: RD unavailability should be less than 5 min per year.

For ATC safety, the models of the regional and the centralized centers are built independently in successive steps according to an incremental approach, following specific construction guidelines, to assess the centers dependability, together with an appropriate specification language associated with transformation rules allowing an automatic generation of optimized GSPNs. The resulting models are very complex. For example the GSPN of the Radar Data Processing and the Flight Plan Data Processing Systems has about 100 places and 500 transitions and corresponds to a reduced Markov chain of about 25,000 states. At the global level, the partial measures assessed for the six centers are combined to assess their impact on ATC safety, more precisely, on the levels of degradation of the service provided to the controller. The results identified the most impacting features that need to be monitored during the design phase.

2.3 Aeronautics

As stated earlier, model-based dependability assessment can be helpful during system operation too, to anticipate system failures (or mission interruption). This is typically the case in aeronautics where airline companies and operators need to reschedule a mission if some components fail. Our collaboration with Airbus was in this context.

The main challenge comes from the fact that the model has to be tuned dynamically, in operation, to take into account the current state of the system, the maintenance environments and potential new information by operators. Indeed, these operators usually do not have any knowledge related to dependability modeling techniques. Hence the model should be prepared and validated in advance, offline, in a way that makes it easily and very quickly configurable in operation.

To this end, we have developed a modeling approach, based on a metamodel used to (i) structure the information needed to assess operational reliability, and to (ii) build a stochastic model to be tuned dynamically to take into account the system operational state, the mission profile and the maintenance facilities [21]. This model allows to (i) assess, on-the-fly, the ability to succeed in continuing on the remaining part of the mission, in case of an unscheduled event occurrence, and to (ii) support maintenance planning. A case study, based on an aircraft subsystem, is considered for illustration, using the Stochastic Activity Networks formalism. It shows how to re-schedule a mission, based on the failed component and its impact on the remaining part of the mission, as well as the maintenance possibilities at the various stops of the aircraft [22].

3 Software Reliability

Even though the first software reliability growth models have been published in early 70s, their practical use in industrial environments was very seldom at the time we had our collaborations with industry. Our first investigations showed that preliminary rigorous analyses of failure data are required before model applications. For example, reliability growth/decrease tests (i) tell about the impact of fault removal on software reliability evolution and (ii) guide selection of models to be applied according to their trend. In particular, reliability growth model assuming pure reliability growth give non-meaningful results if applied to data displaying reliability decrease over some periods of time. Data partitioning improves model results.

We briefly report on two collaborations concerning Electronic Switching Systems (ESSs), developed by two different companies Alcatel and TELEBRAS. Even though both companies were interested in assessing software behavior in operation, the measures of interest were different, due essentially to the different maturity levels of the software, when the study took place.

For Alcatel, the dataset analyzed was collected during 3 years, on a mature system, in operation on more than 1000 sites. One of the expected results of the collaboration was an estimate of the software failure rate. Alcatel aim was to build a Markov chain, taking into account hardware and software failures, to assess the switching system unavailability, to check its compliance with the international telecommunications requirements (that was less than 3 min/year).

Detailed analyses of the data either at the level of the ESS, at the components levels, as well as taking into account the severity of failures (impact of failures on service loss) is performed in [14]. They show for example that: (i) the defense component (in charge of hardware fault tolerance) has the highest failure rate, the three other software components dedicated to functional operation have equivalent, lower, failure rates, (ii) only a very low number of faults led to service unavailability, the others had minor impact.

TELEBRAS started the development of a new series of ESSs increasing progressively the ESS capacity. We concentrated first on the software of the early one before considering together three successive generations. This very first analysis [13] allowed to gain insight into the development process and provided, among other things, an estimate of the number of failures that will occur in the field (equivalently, the number of required corrections), for planning the maintenance effort after system delivery. A comparative analysis of the reliability, in terms of failure rates, of the three generations gives insight into the evolution of the reliability of a family of products [11]. Examples of comparative analyses, with respect to the evolution of the nature of faults activated and their impact through the successive generations, can be found in [9].

4 Dependability Assessment Based on Simulation and Fault injection

Simulation offers complementary means to analytical modeling for analyzing and assessing dependability. Specifically, it offers the possibility to take into account a much wider spectrum of assumptions and to describe the system behavior at a relatively lower level of detail, in order to analyze the effects of faults as close as possible to the components where they occur, and to study their impacts at the system level. Also, it can be used to estimate the parameters involved in analytical dependability models. The main challenge is related to the need to master the simulation time that increases dramatically when the model is simulated at a low level of detail. One possible solution is to develop a hierarchical simulation approach to analyze the behavior of the target system in the presence of faults by considering different levels of abstraction. We have explored this approach in the context of a collaborative research project involving the University of Illinois at Urbana-Champaign, and the StorageTek Company in Colorado, USA [10]. This approach was developed to support the dependability analysis and evaluation of a highly available commercial cache-based RAID storage system. The architecture is complex and includes several layers of overlapping error detection and recovery mechanisms. Three abstraction levels have been considered to model the cache architecture, cache operations, and error detection and recovery mechanism. The impact of faults and errors occurring in the cache and in the disks was analyzed at each level of the hierarchy. The models have been developed using the DEPEND simulation-based environment developed at UIUC, which provides facilities to inject faults into a functional behavior model, to simulate error detection and recovery mechanisms, and to evaluate quantitative measures. Several fault models were defined for each submodel to simulate cache component failures, disk failures, transmission errors, and data errors in the cache memory and in the disks. Some of the parameters characterizing fault injection in a given submodel correspond to probabilities evaluated from the simulation of the lowerlevel submodel. Based on the proposed methodology, we evaluated and analyzed (i) the system behavior under a real workload and high error rate (focusing on error bursts), (ii) the coverage of the error detection mechanisms implemented in the system and the error latency distributions, and (iii) the accumulation of errors in the cache and in the disks. It is important to emphasize that an analytical modeling of the system is not appropriate in this context due to the complexity of the architecture, the overlapping of error detection and recovery mechanisms, and the necessity of capturing the latent errors in the cache and the disks.

Another major challenge in industry concerns the efficient integration of dependability assessment techniques into their existing system engineering process and tools. We had the opportunity to explore this challenge with Technicatome, a French industrial leader in nuclear engineering and with Valeo, a global world-wide automotive supplier.

With Technicatome, the objective was to rely on a commercial systems engineering tool (RDD-100) in order to facilitate the integration of operational safety

analyses in industrial processes at early design stages. This work has resulted in the extension of the functionalities offered by this tool by defining mechanisms for injecting faults into RDD-100 models and analyzing their effects from the point of view of operational safety. The proposed mechanisms are based on two complementary techniques. The first one consists in inserting saboteurs, to disturb the inputs or outputs of the elementary components of the nominal model as well as their behavior. The second one consists in directly mutating the code of the elementary components of the nominal model. A critical analysis of the advantages and limitations of each of these techniques, in terms of difficulty of implementation and the possibilities offered for fault injection and observation of their effects, led us to propose a solution combining these two techniques [12].

Collaboration with Valeo took place in the context of the publication of the first standard specifically dedicated to automotive safety systems, ISO 26262. This standard requires introducing fault injection from the very early phases of the development process. Indeed, even though experimental validation of embedded systems, including fault injection, was of common practice in industry, its adoption in the early design phase, as advocated by the ISO26262 standard, was not common and unclear. In this context, we developed a global approach integrating fault injection in the whole development process in a continuous way, from system requirements to the verification and validation phase. We have shown the strong link between classical safety analyses, commonly used in industry for critical systems design and fault injection principles at design phase. In particular, we have shown the similarities between two well-known domains that are separated in practice, namely (i) Failure Modes, Effects, and Criticality analyses Analysis (FMECA) and (ii) fault injection. More precisely, we have shown how FMECA spreadsheets (i) can be used to guide fault injection on one hand, and to synthesize the results of fault injection in the other hand, and (ii) to link the successive development levels via their failure modes, their causes and their effects, to capture the failure propagation paths between the levels. These chains help making fault injection campaigns effective.

We have shown the benefits of the proposed approach [17], which is compliant with the ISO 26262 standard, on a case study from the automotive domain. The ultimate aim was to guide fault injection experiments, based on early system safety analyses to optimize the whole development process by defining an optimal set of experiments. From a practical point of view, a fault injection tool was developed by Valeo to implement fault injection at various levels.

5 Online Error Detection and Diagnosis

Traditional approaches to dependability assessment are based on the development of models during the design phase in order to assist in architectural choices. However, the need is more and more for automated solutions allowing to monitor and assess the dependability of the system at run-time, i.e. while the system is in operation, using in particular machine learning algorithms. Indeed, the massive collection of

data together with the significant progress and successes achieved with machine learning algorithms have motivated the exploration of these algorithms to support anomaly detection and diagnosis in several application areas. It should be noted that such models have been studied since the 1980s, but have received increased attention in the recent years due to the growing interest in artificial intelligence techniques, particularly in industry.

We have explored the use of such techniques to support online error detection and diagnosis in cloud infrastructures and future telecommunication architectures using network functions and software virtualization technologies (SDN and NFV). This study was carried out in collaboration with Orange Labs. In particular, we have defined a generic strategy enabling the detection of two types of anomalies in cloud services (errors and service level agreement violations) while providing two diagnosis levels to the cloud provider (i.e., identifying the anomalous virtual machine and the type of error causing the anomaly) [19]. The strategy is based on system monitoring data collected online either from the monitored cloud service, or from the underlying hypervisor(s) hosting the service. Different types of machine learning algorithms (supervised, unsupervised, and hybrid) were used to classify anomalous behaviors of the service. Moreover a fault injection tool was developed to collect training data including anomalous samples to train the detection and diagnosis models and to validate our detection strategy. The evaluation was applied to two case studies: a database management system (MongoDB) and an IP multimedia system developed as a virtual network function.

In particular, we have compared the efficiency of several classification algorithms and concluded that the Random Forests algorithm provides in our context the best tradeoff in terms of detection efficiency and training and detection time. The experimental results include a comparative analysis of the detection performance obtained with Operating Systems related monitoring data and hypervisor monitoring data.

We have explored similar approaches in the security area to support the detection of potential intrusions targeting critical embedded applications. In particular, in the context of a joint work with Thales avionics, we designed and implemented a host-based intrusion detection system (HIDS) adapted to the specific constraints and stringent requirements of real-time critical applications embedded in Integrated Modular Avionics (IMA) architectures [6]. The proposed HIDS implements an anomaly-based approach based on the monitoring of ARINC 653 API calls. The model of the legitimate behavior of the application is built based on data collected during the aircraft integration phase. Besides detecting anomalous behaviors of the target application, a signature-based system providing a first diagnosis after the detection of an anomalous behavior was also implemented. This approach has been validated on a real avionic computer and yielded good results in terms of classification accuracy and resource consumption [7]. To support the validation of the HIDS, we developed a tool enabling the automatic injection of attacks and the generation of application code mutations that mimic the behavior of malevolent pieces of code introduced inside the target application [5].

Besides avionics, we also had the opportunity to design and implement, in collaboration with Renault, an intrusion detection system for CAN (Controller Area

Network) based embedded automotive networks, that takes into account specific constraints of the automotive domain (simplicity of implementation without modifying the ECU (Electronic Control Unit) architecture, low cost, low detection latency). The proposed approach consists in automatically generating attack signatures from automata based models, derived from the ECU specification, describing the behavior of the ECUs on board interacting via messages on the CAN bus [20]. This approach was validated on an early prototype using simulated attacks performed on logs of an actual CAN network.

6 Lessons Learned and Concluding Remarks

Joint collaborations with industry have always been an important source of inspiration for new research challenges and solutions, as well as an opportunity to validate our results on real-life use cases and applications. Such collaborations have allowed us to address a wide range of topics with several industry partners from various application domains (aeronautics, automotive, telecommunication, energy, etc.). Development of scalable and generic solutions that can accommodate the increasing complexity of computing systems and be easily integrated within industrial system engineering processes are concerns shared by several industry partners. The examples presented in this chapter clearly show the evolution of the topics of interest to industry over the years, at least as we have experienced it from our side. This evolution is in-line with the evolution of the technological trends through the years.

Over the years, we learned to reach quickly a consensus between industry needs and research objectives while preserving intellectual and industrial properties. In particular, we have accepted not to publish all results. In an academic world where publication is becoming more and more a driving process, this may be not acceptable. However, we have always been able to agree on (i) parts of results we were allowed to publish and/or include in the PhD dissertation without harming the industrial properties and (ii) results to be delivered to the industrialist only. With the benefits of hindsight, this process was always challenging and extremely rewarding. Another lesson learned concerns the clear distinction, upfront, between the user and the system provider perspectives. This led us to define dependability attributes related to the two points of view, and try to optimize both of them or at least reach an acceptable compromise, which was not obvious all the time, but truly challenging too.

Currently, significant efforts in many industries are being devoted to exploring the opportunities opened by recent advances in artificial intelligent techniques together with massive data collection thanks to the widespread deployment of IoT technologies. One of the main questions is how to leverage these techniques in various areas (monitoring, anomaly and intrusion detection and diagnosis, testing, etc.) to improve the dependability, resilience and quality of service of networks and computer systems performance while reducing the costs. A major challenge in this context is related to the lack of trust and confidence in such techniques and the need for rigorous and

formalized approaches to provide justified assurance and ensure a better explainability and acceptability of AI algorithms in a context where malicious threats are also increasing. This problem has many dimensions and requires an interdisciplinary approach combining expertise from various scientific fields including mathematics, optimization theory, computer science, and also social and neuro-sciences. The study of use cases in different application domains and the cross-fertilization of solutions from different industrial sectors will be a key to achieve significant advances in this field.

Acknowledgements The chapter topic was inspired by friendly discussions with Ravi Iyer, over the years, especially during our sabbatical visits to the University of Illinois at Urbana Champaign or at LAAS-CNRS in Toulouse. We all know that Ravi enjoys the practical aspects of research, in addition to deep conceptual work. We had the opportunity to share with him our lessons learned and some ideas inspired from the industry projects that we have conducted as well as from our experimental-oriented research on system dependability and software reliability, and also more recently on the assessment and mitigation of security threats. Thank you Ravi!

We also thank all our numerous colleagues from LAAS-CNRS and from industry with whom we have carried out the collaborative work outlined in this chapter.

References

1. Béounes C, Aguera M, Aalat J, Bachmann S, Bourdeau C, Doucet JE, Kanoun K, Laprie JC, Metge S, Moeira De Souza J, Powell D, Spiesser P (1993) SURF-2: a program for dependability evaluation of complex hardware and software systems. In: 23rd IEEE international symposium on fault-tolerant computing (FTCS'23), Toulouse (France), pp 668–673
2. Betous-Almeida C, Kanoun K (2004) Dependability modelling of instrumentation and control systems: a compaaprison of competing architectures. Saf Sci 42(5):457–480
3. Betous-Almeida C, Arazo A, Crouzet Y, Kanoun K (2000) Dependability of computer control systems in power plants. Analytical and experimental evaluation. In: 19th international conference on computer safety, reliability and security (SAFECOMP'2000). Rotterdam, 24–27 October 2000
4. Blanquart JP, Boussin JL, Kanoun K, Laprie JC (1982) REBECCA: a dependable communication sub-system for the control system of extra-high-voltage substations. In: Fifth European conference on electrotechnics. Lyngby (Danemark), pp 825–829
5. Damien A, Feyt N, Nicomette V, Alata E, Kaâniche M (2019a) attack injection into avionic systems through application code mutation. In: 38th digital avionics systems conference (DASC-2019). San Diego, CA (US), 10 p
6. Damien A, Marcourt M, Nicomette V, Alata E, Kaâniche M (2019b) Implementation of a host-based intrusion detection systems for avionic applications. In: 24th IEEE pacific rim international symposium on dependable computing (PRDC-2019). Kyoto, IEEE CS, IEEE CS, 10 p
7. Damien A, Gimenez P-F, Feyt N, Nicomette V, Kaâniche M, Alata E (2020) On-board diagnosis: a first step from detection to prevention of intrusions on avionics applications. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE-2020). Coimbra, Portugal, pp 358–368
8. Fota N, Kaâniche M, Kanoun K (1999) Dependability evaluation of an air traffic control computing system. Perform Eval 35(3–4):253–273

9. Kaâniche M, Kanoun K (1998) Software reliability analysis of three successive generations of a telecommunications system. In IEEE workshop on application-specific software engineering and technology (ASSET'98). Richardson (USA), pp 122–127
10. Kaâniche M, Romano L, Kalbarczyk Z, Iyer R, Karcich R (1998) A hierarchical approach for dependability analysis of a commercial cache-based RAID storage architecture. In: 28th IEEE international symposium on fault-tolerant computing (FTCS-28). Munich (Germany), pp 6–15
11. Kaâniche M, Kanoun K, Cukier M, Bastos Martini M (1994) Software reliability analysis of three successive generations of a switching system. In: First european conference on dependable computing (EDCC-1). Berlin, Germany, pp 473–490
12. Kaâniche M, Le Guédart Y, Arlat J, Boyer T (2004) An investigation on mutation strategies for fault injection into RDD-100 models. Saf Sci 42(5):385–403
13. Kanoun K, Bastos Martini M, Moreira de Souza J (1991) A method for software reliability analysis and prediction—application to The TROPICO-R switching system. In: IEEE transactions on software engineering, vol 17, pp 334–344
14. Kanoun K, Sabourin T (1987) Software dependability of a telephone switching system. In: 17th IEEE international symposium on fault tolerant computing (FTCS-17). Pittsburgh (USA), pp 236–24
15. Kanoun K, Borrel M, Morteveille T, Peytavin A (1999) Availability of CAUTRA, a subset of the French air traffic control system. IEEE Trans Comput 48(5):528–535
16. Laprie JC, Kanoun K (1980) Dependability modeling of safety systems. In: 10th IEEE international symposium on fault-tolerant computing, pp 245–250
17. Pintard L, Fabre JC, Kanoun K, Leeman M, Roy M (2014) From safety analysis to experimental validation of automotive systems. In: IEEE Pacific rim dependable computing conference (PRDC 2014). Singapore
18. Rugina AE, Kanoun K, Kaâniche M (2011) Software dependability modeling using AADL (architecture analysis and design language). Int J Perform Eng 7(4):313–325
19. Sauvanaud C, Kaâniche M, Kanoun K, Lazri K, Da Silva Silvestre G (2018) Anomaly detection and diagnosis for cloud services: practical experiments and lessons learned. J Syst Softw Special issue Softw Reliab 139:84–106
20. Studnia I, Alata E, Nicomette V, Kaâniche M, Laarouchi Y (2018) A language-based intrusion detection approach for automotive embedded networks. Int J Embed Syst 10(1)
21. Tiassou K, Kanoun K, Kaâniche M, Seguin C, Papadopoulos C (2012) Impact of operational reliability re-assessment during aircraft missions. In: 31st IEEE international symposium on reliable distributed systems (SRDS 2012). Irvine, CA, USA, pp 219–224
22. Tiassou K, Kanoun K, Kaâniche M, Seguin C, Papadopoulos C (2013) Aircraft operational reliability—a model-based approach and a case study. RESS (Reliab Eng Syst Saf) 120:163–176

Assessing Dependability of Autonomous Vehicles

Saurabh Jha

Abstract Autonomous vehicles (AVs) such as self-driving cars and unmanned aerial vehicles are complex systems that use artificial intelligence (AI) and machine learning (ML) to make real-time navigational decisions. Ensuring the dependability of AVs in terms of robustness, correctness, reliability, and safety is critical for their mass deployment and public adoption. However, it is challenging to assess and ensure the dependability of these systems due to their complexity both in terms of software and hardware and in terms of the inherent stochasticity and uncertainty in the sensor data and ML/AI algorithms. In this chapter, we design and develop novel assessment techniques to rigorously validate the AV system, including its runtime operational characteristics. The developed assessment techniques address the challenges mentioned above and significantly outperform the current state-of-the-art assessment techniques. We demonstrate our developed techniques and scientific contributions using self-driving cars as a motivating example.

Keywords Autonomous vehicles · Safety · Assessment

1 Introduction

Autonomous vehicles (AVs) such as self-driving cars and unmanned aerial vehicles are complex systems that use artificial intelligence (AI) and machine learning (ML) to integrate mechanical, electronic, and computing technologies to make real-time navigational decisions. AI enables AVs to make their way through complex environments while maintaining a *safety envelope* [1, 2] that is continuously measured and quantified by onboard sensors (e.g., camera, LiDAR, RADAR) [3–5].

AVs suffer both from traditional dependability challenges in hardware and software and from ML/AI-related challenges. Traditional dependability challenges in hardware and software include single-event upsets, bugs, and performance anomalies, whereas ML/AI-related challenges include data and model-related failures. (see

S. Jha (✉)
IBM T. J. Watson Research, Yorktown Heights, USA
e-mail: Saurabh.Jha@ibm.com

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_24

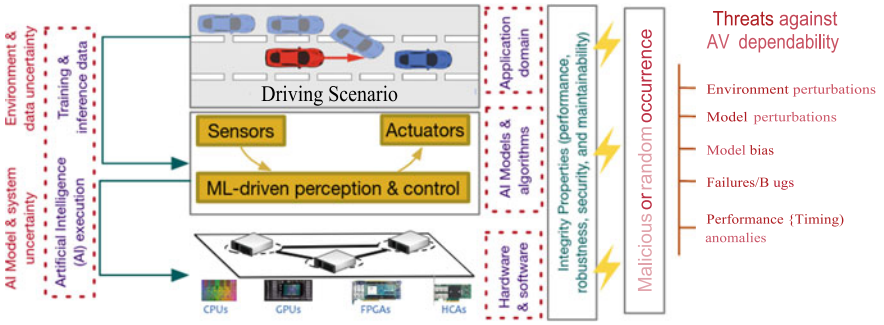


Fig. 1 Threats and challenges in achieving dependability in mission-critical systems

Fig. 1). All of these factors create significant uncertainty at runtime in the system, which may lead to erroneous decisions. For AVs, such errors must be detected and mitigated dynamically at runtime with limited resource (because a commercial AV has to be a cost-sensitive product for wide adoptability) and time budgets (because a delay of few milliseconds can lead to safety hazards).

Industry-grade AVs are equipped with detection and mitigation techniques to handle those challenges; however, a significant number of failures silently escape detection. Such silent failures, if not dealt with, lead to erratic driving behavior and safety hazards; thereby, reducing the trust we place on them, as exemplified from several headline-making AV crashes [6, 7]. Moreover, an adversary can masquerade an attack as a silent failure by intelligently perturbing the environment or models to evade detection and successfully cause a safety hazard [8]. Hence, there is a compelling need for a comprehensive assessment of AV technology to identify and handle those silent failures.

Contributions. In this chapter, we propose novel assessment techniques to rigorously validate the system, including its runtime operational characteristics. We demonstrate these techniques and scientific contributions using self-driving cars as a motivating example. A self-driving car uses an autonomous driving system (ADS) technology capable of supporting and assisting a human driver in the tasks of (i) monitoring the surrounding environment (e.g., other vehicles/pedestrians, traffic signals, and road markings) [9] and (ii) planning and control of the vehicle by actuating the vehicle using throttle, steer, and brake commands. We target self-driving cars because they represent one of the most complex systems built by humans and because they integrate mechanical, electrical, electronic, and machine-learning components. However, the work described in this chapter is broadly applicable to other ML-driven control systems.

Our contributions in the field of assessment include the following:

(a) *Empirical assessment using production systems and field-failure datasets*: Empirical measurement using field failure datasets obtained from a production systems allows designers to (i) characterize unanticipated operational behavior and identify the underlying cause, (ii) statistically estimate the percentage contribution

of different failure modes to safety hazards, and (iii) track the improvement in safety over time. In this line of work, our contributions include (i) overlaying domain-driven models with empirical datasets to deal with the inherent noise and partial observability present in the real world and (ii) using real-world data collected over 26 months from September 2014 to November 2016, obtained from California Department of Motor Vehicle (CA-DMV), consisting of data from 12 AV manufacturers for 144 vehicles that drove a cumulative 1,116,605 autonomous miles.

(b) *Validation using fault injection and fuzzing*: Perturbation techniques such as fault injection and fuzzing allow designers to perturb the system and the environment, respectively. Such techniques assess the system's susceptibility to various failure modes and identify failure propagation chains, which can be used to develop defenses. These validation techniques can be applied in both the real and simulated worlds. However, validation in a simulated world is safer and faster than validation in the real world, as it poses no risk to humans or property. In this line of work, our contributions include accelerating validation techniques to (i) significantly reduce the time needed to identify safety-hazard-causing faults and fuzzes and (ii) increase the fault/fuzz coverage. The acceleration is achieved by pruning the fault/fuzz space using domain knowledge and causal and counterfactual reasoning techniques.

(c) *Masquerading attacks as failures*: Creating a security hazard that can be masqueraded as a random silent failure and result in a serious safety compromise (for example, an accident) is attractive from an adversary's perspective. Demonstrating these attacks is particularly important for highly exposed systems like self-driving cars because it allows designers to understand the steps required to forge such an attack and to develop deterrents to make these attacks significantly harder. In this line of work, our contributions include showcasing the steps of an attack that can masquerade as a silent failure, including answering the questions of *what*, *when* and *how* to attack.

Chapter organization. §2 describes the broader research impact. §3 describes empirical techniques and results. §4 describes fault injection and fuzzing-based validation techniques. It also describes an attack technique that masquerades attacks as naturally and randomly occurring faults or perturbations. Finally, §5 concludes the chapter by summarizing the work and providing ideas for future work.

2 Broader Research Impact

Society as a whole is going to witness exponential growth and adoption of AI/ML-driven cyber-physical systems in critical application domains such as healthcare, transportation, agriculture, and manufacturing. The availability and deployment of dependable AI-driven cyber-physical systems is valuable, as they (i) increase efficiency of tasks carried out by humans (e.g., search and rescue missions, pack-

age delivery, and healthcare) and (ii) enable execution of tasks that are nearly impossible or dangerous for humans (e.g., mineral mining and deep-sea exploration). The widespread adoption of such systems in a human-centric environment necessitates understanding AI-engineered systems and their capabilities in the presence of a wide range of uncertainties, from specification to real-time operations. These next-generation, AI-driven systems demand an ever-increasing level of system dependability (i.e., performance, robustness, security, maintainability, and ease of use) not available today. The classical approach to dependability (availability, fault tolerance, integrity, security, etc.) is based upon component reliability views and fault/error/attack management at the architecture level. While necessary, the classical approaches are not sufficient, and new methods must be developed to account for autonomy and safety requirements.

Our work is a step in that direction. We develop novel, causality-driven techniques that meet those demands and provide the theory and foundation for designing dependable (trustworthy) ML/AI-driven cyber-physical systems. Methods proposed in this work will allow designers to assess and ensure the dependability of such systems. We showcased these techniques on self-driving cars, which are complex, mission-critical ML/AI-driven systems. We believe that the techniques proposed in this work will pave the way to ensuring the dependability of other autonomous systems, such as unmanned aerial vehicles, agricultural robots, and kitchen bots, among others.

3 Empirical Assessment Using Production Systems and Field-Failure Datasets

An empirical assessment of field datasets from real-world production systems enables (i) discovery of failure modes and operational characteristics encountered in the field; (ii) quantification of failure statistics, the relative contribution of each failure mode, and failure propagation paths; and (iii) mining and specification of assertions and integrity properties that can further guide offline assessment and online detection.

However, using field-failure datasets is challenging, as the internal details of the system are not available or interpretable (e.g., DNNs). Moreover, these datasets are inherently noisy and incomplete, forcing the user to make assumptions that may not be true. Our work addresses these challenges by designing data analysis techniques that overlay field-failure datasets with an abstract representation of the control system, enabling us to query factual and counterfactual questions (use causal reasoning) to reason about the observed safety hazards.

3.1 Concept and Approach

We demonstrated our model-driven, field-failure analysis technique by characterizing the dependability of AVs to showcase the technique's capability to evaluate the cause, dynamics, and impact of failures across a wide range of AV manufacturers. We use publicly available field data from tests on California public roads, including urban streets, freeways, and highways [11].

Any feedback-based control system, such as an autonomous vehicle, must internally imitate a causal framework for achieving its goals. For example, a self-driving car must perceive objects (via sensors and recognition systems), determine their trajectories (via trajectory estimation), plan its own trajectory, and execute control commands to follow the planned trajectory (via planning and control models). This must be true irrespective of the nature of the system (i.e., whether it is driven using expert-rules, deep learning techniques, or a combination of both). An accident or disengagement[1] must be a result of an inconsistency in reasoning, which in turn must be due to an ML inference failure, hardware failure, or a bug in software or hardware. We leverage this insight to develop an abstract model of the ML-driven cyber-physical systems and overlay the field dataset to pinpoint failure causes. In particular, we model a cyber-physical system using a control flow graph and ask "what-if" questions to the model (factual and counterfactual reasoning). In our work, we use System Theoretic Process Analysis (STPA [12]) for modeling and the California Department of Motor Vehicles (CA DMV) dataset to characterize the dependability of self-driving cars.

System Theoretic Process Analysis. To identify multidimensional causes of AV disengagements/accidents, we built a hierarchical control structure for AVs, rooted in causal reasoning, by using systems-theoretic hazard modeling and analysis abstraction (STPA). Figure 2 shows the abstract AV hierarchical control structure and highlights the three control loops (`CL-1`, `CL-2`, and `CL-3`, indicated with different types of dashed lines). STPA employs concepts from systems and control theories to model hierarchical control structures in which the components at each level of the hierarchy impose safety constraints on the activity of the levels below them and communicate their conditions and behavior to the levels above them. Accidents and disengagements are complex dynamic processes resulting from inadequate perception control and decision-making at different layers of the system control structure. Accidents and disengagements seen in the data were overlaid on this structure. Analysis of dependencies along those control loops allows for the identification of inadequate controls and of the potential causes of those unsafe control actions through the examination of the operation of components and their interactions in each loop of the control structure. Any flaws or inadequacies in the algorithm, the process model, or the feedback used by a controller are considered potential causal factors leading to unsafe control actions and resultant disengagements/accidents.

[1] A transfer of control from the autonomous system to the human driver in the case of a failure is called a *disengagement*. Disengagements can be initiated either manually by the driver or autonomously by the car.

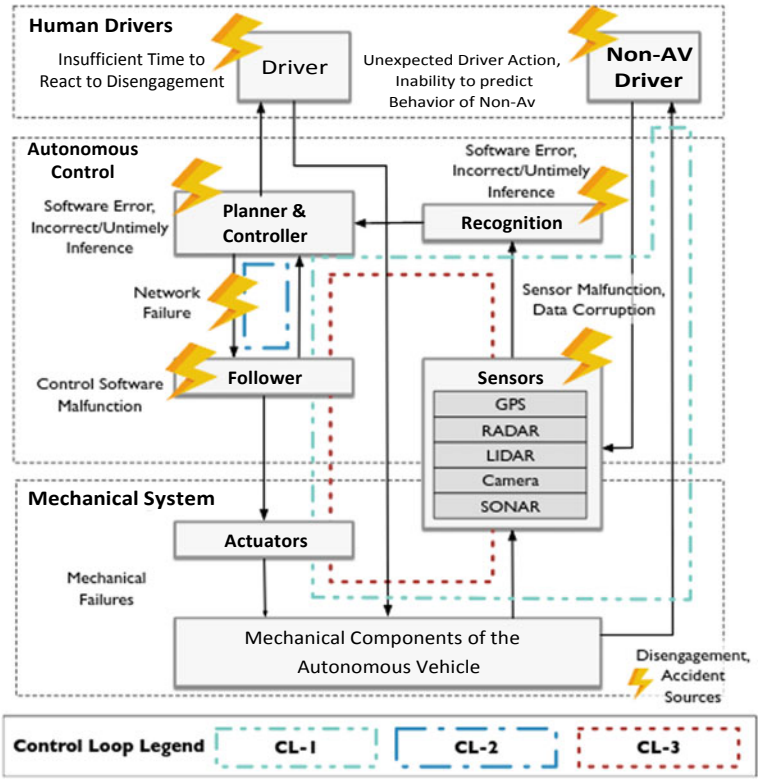


Fig. 2 Autonomous vehicle hierarchical control structure drawn based on [10]. Examples of control loops are highlighted as `CL-1`, `CL-2`, and `CL-3`

Dataset. The California Department of Motor Vehicles (CA DMV) mandates that all manufacturers testing AVs on public roads file annual reports detailing both disengagements and accidents (actual collisions with other vehicles, pedestrians, or property) [13]. We digitize and normalize the schema for all the datasets. Moreover, we use NLP techniques to label the failure cause (tag) of each accident/disengagement. Example data after post-processing of manufacture-provided dataset is shown in Table 1. We analyze field data collected over a 26-month period from September 2014 to November 2016 (part of the DMV's 2016 and 2017 data releases), containing data from 12 AV manufacturers for 144 vehicles that drove a cumulative 1,116,605 autonomous miles. Across all manufacturers, we observe a total of 5328 disengagements, 42 of which led to accidents.

Table 1 Sample of disengagement reports from the CA DMV dataset

Manufacturer	Raw disengagement report (log)	Category	Tags
Nissan	1/4/16	1:25 PM	**Software module froze**. As a result driver safely disengaged and resumed manual control.	City and highway	Sunny/Dry	System	Software
Nissan	5/25/16	11:20 AM	Leaf #1 (Alfa)	The AV **didn't see** the lead vehicle, driver safely disengaged and resumed manual control.	ML/Design	Recognition system
Waymo	May-16	Highway	Safe operation	Disengage for a **recklessly behaving** road user	ML/Design	Environment
Volkswagen	11/12/14	18:24:03	Takeover-Request	**watchdog error**	System	Computer system

We use the "|" to denote field separators
Note that log formats vary across manufacturers and time
Bold-face text represents phrases analyzed by the NLP engine to categorize log lines

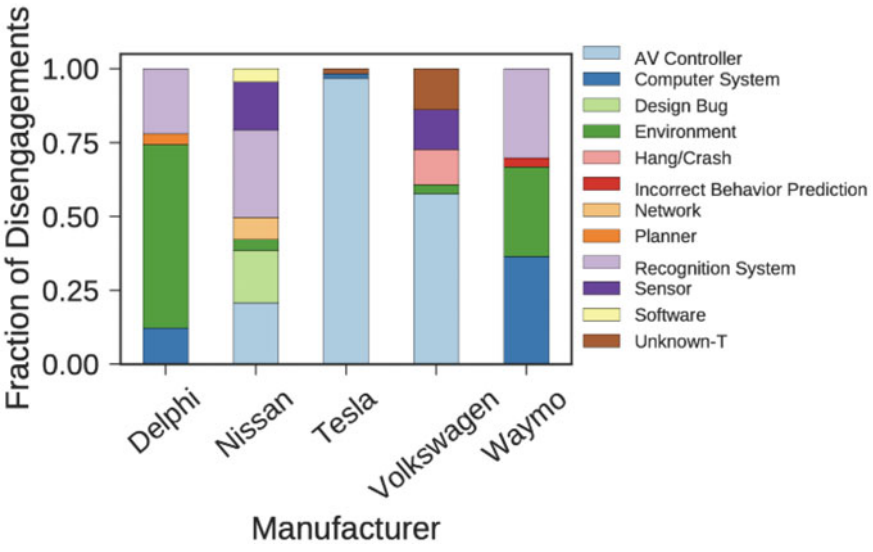


Fig. 3 Categorization (in terms of fault tags) of faults that led to disengagements across manufacturers

3.2 Results

Our study shows the following:

(i) 64% of disengagements across all manufacturers were the result of problems in, or untimely decisions made by, the machine learning. Figure 3 shows the failure causes across all manufacturers. Table 2 describes the failure causes.

Table 2 Definition of fault tags and categories that are assigned to disengagements

Tag (failure causes)	Category	Definition
Environment	ML/Design	Sudden change in external factors (e.g., construction zones, emergency vehicles, accidents)
Computer system	System	Computer-system-related problem (e.g., processor overload)
Recognition system	ML/Design	Failure to recognize outside environment correctly
Planner	ML/Design	Planner failed to anticipate the other driver's behavior
Sensor	System	Sensor failed to localize in time
Network	System	Data rate too high to be handled by the network
Design bug	ML/Design	AV was not designed to handle an unforeseen situation
Software	System	Software-related problems such as hang or crash
AV Controller	System ML/Design	"System" when AV controller does not respond to commands
		"ML/Design" when AV controller makes wrong decisions/predictions
Hang/crash	System	Watchdog timer error

(ii) Drivers of AVs need to be as alert as drivers of non-AV vehicles. Further, the small size of the overall action window (detection time + reaction time) would make reaction-time-based accidents a frequent failure mode with the widespread deployment of AVs.

(iii) For the same number of miles driven, for the manufacturers that reported accidents, human-driven non-AVs were $15 - 4000\times$ less likely than AV's to have an accident. In terms of reliability per mission, AVs are $4.22\times$ worse than airplanes.

3.3 Novelty

The majority of the prior research into AV systems focuses on the functionality of the systems. Numerous demonstrations of end-to-end computing systems for autonomous vehicles have recently been done (e.g., [14–21]). While safety is emphasized in a number of publications, including [22, 23], they tend to ignore the end-to-end AV safety and dependability challenges.

Our goal is to assess the end-to-end safety and dependability challenges in AVs using field-failure data. We present an analysis of the entire control system of the AV, of which DNNs form the subset of the overall system.

4 Validation Using Fault Injection and Fuzzing

A key issue for autonomous systems is rigorously demonstrating and validating their safety. The evolutionary, context-sensitive behavior of autonomous systems can cause unexpected emergent behavior or unforeseen interactions that were not necessarily envisioned at the architectural stage of the system design. For these reasons, it is necessary to develop more refined, at-scale implementations of the autonomic functions and architectures that can challenge assumptions and modes of operations.

The causes of "unexpected" behaviors include unforeseen interactions between autonomy and vehicle, various notions of failure and hazard scenarios, faults (design and physical), and security threats. However, it is difficult to assess such systems for several reasons, including (i) Although collecting real-world data in the field is valuable (as discussed in §3), it is not viable to test mission-critical systems because (a) it is too slow to test systems in the field, and (b) it can be dangerous for humans and property to do so and therefore may not be ethical. Moreover, collecting this data can be slow and expensive, as it must be preprocessed and labeled correctly before use. (ii) Enumerating over fault/attack space and inputs, which is combinatorially large, is infeasible.

4.1 Concept and Approach

We address those challenges by modeling the problem of identifying safety-critical perturbations[2] (inputs, faults, attacks) as a machine learning problem under causal framework (do-calculus [24]). The proposed framework, called the Bayesian Fault Injector (BFI) [25], relies on factual and counterfactual reasoning about the system state under the perturbation. Unlike STPA, which requires manual reasoning, our proposed framework is automatic. The goal is to generate perturbations that will most likely lead to a safety hazards if activated in the system. We use ML because it helps prune the combinatorially large space of inputs quickly by eliminating an entire subspace of faults/inputs where the likelihood of a resultant safety hazard is low (due to inherent masking offered by the system or environment).

The approach overview, including training and inference steps of BFI, is shown in Fig. 4. BFI models the system state as a joint distribution over the inputs and individual autonomous driving software modules. BFI, once trained, allows us to infer the state of the system under perturbation. In our toy example in Fig. 4, the joint distribution of the system is $P(U, X, Y, Z)$. Perturbation impacts the software state ($J_p = P(U, X, Y, Z|perturb(X = x))$, which eventually impacts the driving trajectory leading to a safety hazard (e.g., collision). We use do-calculus [26], a kinematics model, and ean mergency stop-based safety model to estimate the likelihood of a collision.

[2] We use the terms "perturbation" and "fault" interchangeably.

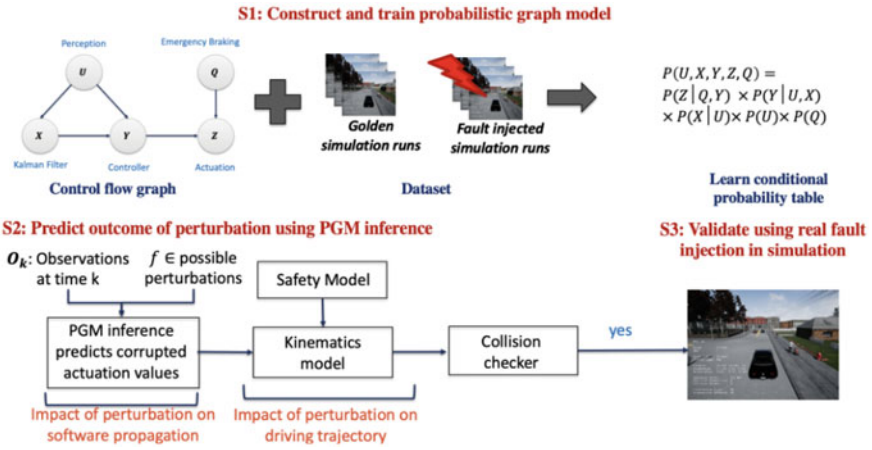


Fig. 4 Bayesian Fault Injection framework on a toy AV system consisting of perception, Kalman Filter, controller, actuation and emergency braking

Do-calculus allows factual and counterfactual reasoning about the system-state under the perturbation. Using do-calculus, the system state under perturbation can be rewritten as $J_p = P(U, X, Y, Z|do(X = x)$. The do-calculus requires a specification of (i) the control-flow graph, which captures the dependency model among random variables, and (ii) the conditional distribution among the random variables. The dependency model can be specified using the prior knowledge of system design (which is assumed to be available to the tester). The conditional distribution among the nodes of CFG is modeled using Temporal Bayesian Networks (TBNs), a probabilistic graph model (PGM)-based ML approach. Using TBNs to represent the joint distribution reduces the number of parameters required to estimate the joint distribution significantly. Finally, we specify functional relationships capturing the dependency between the system state and safety (e.g., collision avoidance). Thus, it explicitly models the propagation and masking of faults/attacks and their impact on safety. TBNs are trained using system traces gathered with and without fault injection.[3] TBNs significantly reduce the need to gather training data (system traces), as it explicitly models the relation using statistical parameters, thereby reducing the computational overhead and training time. The overall training procedure of BFI is marked as S1 in Fig. 4.

Once the TBN model is trained, the proposed framework is used in offline mode (i.e., using the collected simulation traces) to infer/predict the impact of perturbation on the AV software outputs (i.e., actuation values). In this work, we evaluated the susceptibility of AVs to one perturbation at a time. However, more than one perturbation is allowed in BFI. Such offline inference is significantly faster than perturbing

[3] In this work, we use simulation traces consisting of the recorded inputs and outputs of the system software modules at each timestep obtained while simulating a driving scenario in a Physics-based simulation engine.

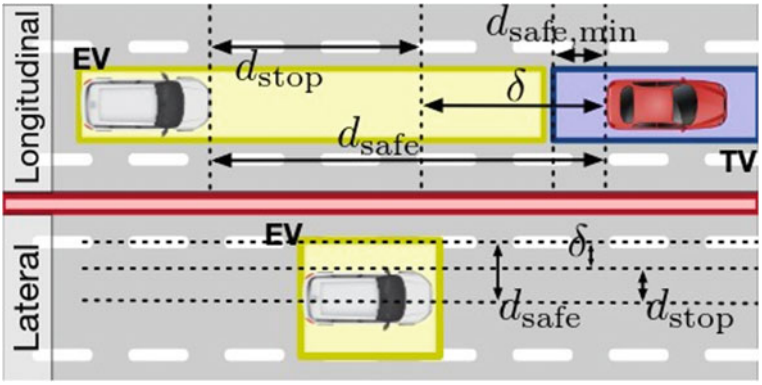


Fig. 5 Definition of d_{stop}, d_{safe}, and δ for lateral and longitudinal movement of the car. Non-AV vehicles are labeled as *target vehicles* (TV)

the system using fault injection techniques to observe the outputs or safety hazards. This is because inference does not require fault injection or execution of the software; instead, it uses the collected traces. The corrupted outputs can change the trajectory of the autonomous vehicle, which in turn can lead to a collision. However, most vehicles are equipped with a safety monitoring system to detect imminent collisions and engage emergency braking. But if the stopping distance is more than or equal to the distance to the closest in-path object on the road, the vehicle will nonetheless collide with the object. We estimate the stopping distance using a kinematics model, assuming that the vehicle has a safety monitoring system capable of engaging emergency brakes (using a safety model explained later in this section). The collision checker uses the stopping distance to predict a collision. The overall inference procedure of BFI is marked as S2 in Fig. 4. Finally, if BFI predicts that a perturbation is likely going to cause a safety hazard, we do an actual fault injection on the software while executing the driving scenario in a Physics-based simulator to validate the output of BFI itself (as shown in S3 in Fig. 4).

Modeling AV safety. BFI requires a precise definition of AV safety. We define the instantaneous safety criteria of an AV in terms of the longitudinal (i.e., direction of motion of the vehicle) and lateral (i.e., perpendicular to the direction of motion of the vehicle) distance travelled by the AV (see Fig. 5). Those criteria form a "primal" definition of safety based on collision avoidance, which can be extended with other notions of safety, e.g., obeying traffic rules. The extended notions of safety are not considered in this paper, as they can be nuanced based on the laws of the geographic regions in which they are applied.

Definition 1 The *stopping distance* d_{stop} is defined as the maximum distance the vehicle will travel before coming to a complete stop while the maximum comfortable deceleration a_{max} is being applied.

Definition 2 The *safety envelope* d_{safe} [1, 2] of an AV is defined as the maximum distance an AV can travel without colliding with any static or dynamic object.

A safety envelope is used to ensure that the vehicle trajectory is collision-free. Production ADSs use techniques such as those in [27, 28] to estimate vehicle and object trajectories, thereby computing d_{safe} whenever an actuation command is sent to the mechanical components of the vehicle. These ADSs generally set a minimum value of d_{safe} (i.e., $d_{\text{safe,min}}$) to ensure that a human passenger is never uncomfortable about approaching obstacles.

Definition 3 The *safety potential* δ is defined as $\delta = d_{\text{safe}} - d_{\text{stop}}$. An AV is defined to be in a *safe state* when $\delta > 0$ in both lateral and longitudinal directions.[4]

BFI finds perturbations that lead to safety hazards under this definition (i.e., $\delta <= 0$).

Identifying safety-critical driving scenarios via fuzzing. To assess the AV's ability to operate under novel driving scenarios, we developed AVFuzzer [29], which uses the above-mentioned safety model to generate test cases that determine the safety violations of an AV in the presence of an evolving traffic environment. Here, we use genetic algorithms (GA) to perturb valid driving scenarios. Unfortunately, existing GA techniques converge slowly. Hence, the goal is to accelerate GA in the context of AV fuzzing. Starting from a valid driving scenario, AVFuzzer perturbs the driving maneuvers of traffic participants (e.g., other vehicles in the environment) to create situations in which an AV runs into safety hazards. To optimally determine the perturbations that are to be introduced, we use the above-mentioned safety model to guide the search. Here, the goal of GA is to decrease the safety potential of the vehicle. AVFuzzer consists of three parts: (i) a GA-based search engine that learns and assesses the safety of the AV, generating seed scenarios that are likely to have high potential of safety hazards, (ii) a local fuzzer, which exploits the seeds and dynamically evolves them into safety-hazard scenarios when possible, and (iii) an on-demand restart mechanism that repeats the optimization with significantly different starting points in unexplored search space in order to determine diverse safety hazardous situations.

In this work, we used a black-box testing technique and hence assumed no prior knowledge of the internals of the AV system. However, if the domain knowledge on the system's internals is available, BFI can be used in conjunction with AVFuzzer to accelerate fuzzing significantly. Testing techniques that assume internal knowledge of the system fall under the umbrella of white-box testing. This work is described in detail in [29].

Masquerading attacks as random faults. The above-mentioned techniques are also used by the attackers to identify runtime vulnerabilities in the system. However, a challenge for an attacker is to hide the footprint and evade detection. One approach to evade detection is to masquerade the attack as a naturally occurring random fault in the system. However, it is challenging to masquerade attacks as faults in AVs

[4] We use the shorthand $\delta > 0$ to mean both lateral and longitudinal δs.

because of the inbuilt compensation in the system and environment. For example, state tracking algorithms such as Kalman and particle filters tolerate random noise. Similarly, an attack launched on an AV on an empty road will not result in a collision. Moreover, the intrusion detection system can detect the attack if the attacker maintains its presence for too long in the system.

To address these challenges and to identify vulnerabilities that still exist in the system, we created *RoboTack* [30], an intelligent malware. The goal of the work is to identify the steps of the attack and to prevent the attack by building security measures that thwart the execution of those steps. A key feature of RoboTack is its ability to disguise attacks as accidental/random faults to evade detection yet cause serious safety/reliability incidents (e.g., an accident of an autonomous vehicle). RoboTack does so by answering the questions of *what*, *how*, and *when* to attack the system under test by using a runtime decision framework whose goal is to decrease the safety potential within some threshold duration. The decision framework uses telemetry data to identify the most vulnerable system state (answering *when*) and the corresponding faults (answering *what*) that will minimally perturb the system (i.e., without being detected) while still leading to safety/reliability incidents (answering *how*). This approach focuses on evaluating the end-to-end dependability of AVs instead of focusing *only* on ML/AI models (e.g., stop-sign attacks).

4.2 Results

We have used *BFI* on Apollo [4, 31], an industry-grade autonomous vehicle technology stack from Baidu. Our initial experiences in applying BFI on AVs had the following results:

(i) BFI is highly effective in finding fault-tolerance-related bugs (561 unique perturbations/faults) that can lead to fatal collisions. Moreover, it provides a speedup of 1600× over traditional methods.
(ii) RoboTack demonstrated the steps of an attack that can be used by an adversary to leverage known faults and failure modes. A RoboTack-generated attack is highly fatal (15–25× more likely to be fatal than state-of-the-art adversarial attacks [32]), and it evades known detection techniques.
(iii) AVFuzzer found 13 unique adversarial traffic patterns in which Apollo runs into hazardous situations that lead to crashes. In comparison, exiting techniques such as random fuzzing and adaptive stress testing can find only 1 and 5 unsafe cases during the same period of search time.

4.3 Novelty

Assessment of the safety and resilience of AVs requires robust testing techniques that are scalable and directly applicable in real-world driving scenarios. It is not scalable or practical to base a safety argument solely on statistical measures, such as a billion miles on roads, or on simulations done on platforms such as CARLA [33] or Open Pilot [5, 34, 35]. Testing the robustness of an ADS has proven to be challenging and mostly ad hoc or experience-based [36]. In particular, to test the functionality and design of the hardware and software components of an ADS, current methods rely on injection of invalid or perturbed inputs [37–39] or faults and errors [39–41] into an ADS in simulation or into ADS components, and accrual of millions of miles on roads [23].

However, these methods are not scalable because (i) they lack simulated or real datasets that would represent all kinds of driving scenarios [34], (ii) it would take billions of miles of driving to add functionality or do a bug fix in order to drive statistical measures [42], (iii) they are restricted to DNNs [35, 40, 43–45] and sensors [38, 39], even when DNNs form only a small part of the whole ecosystem, and (iv) once the easy bugs have been fixed, finding rare hazardous events would be exponentially more expensive, as faults might manifest only under specific conditions (e.g., a certain software state). Our approach addresses this gap by modeling the problem of identifying safety-critical perturbations (inputs, faults, attacks) as a machine learning problem under a causal framework (do-calculus [24]).

5 Conclusion and Future Work

In this chapter, we discussed and demonstrated dependability and safety assessment techniques for AVs. We described several assessment techniques, such as empirical analysis of field datasets, fault (or perturbation) injection, input fuzzing, and adversarial techniques. We highlighted the benefits and challenges of using these techniques in the context of AVs. Finally, we addressed these challenges in ways that significantly outperform the current state-of-the-art AV assessment techniques.

Extending the techniques for online dynamic risk assessment. The proposed techniques currently are suitable for design-time assessment, so they are not suitable for runtime assessment. It is plausible that the proposed methods may miss important rare bugs or faults at design time, leading to safety hazards at runtime. Therefore, there is a need for extending the proposed techniques for online assessment. However, achieving this would require significantly scaling the techniques to handle a large number of actors in milliseconds, as is needed to meet the resource and deadline constraints on runtime decision making. Our early work [46] proposes techniques only for safety risk evaluation. In this work, we define a novel, safety-importance metric that characterizes the influence of an actor (and a driving scene) on the driving decisions of the Ego actor. This metric allows us to identify all the important actors

on the road and to use that information to quickly evaluate the risk at runtime. We plan to integrate this safety-importance metric with the techniques proposed in this chapter to meet the runtime deadline and resource constraints.

Relevance to other ML-driven systems. ML and AI methods are increasingly being used in mission-critical, cyber-physical systems to automate various tasks to achieve high autonomy and performance. Popular examples of mission-critical, cyber-physical systems include autonomous vehicles (e.g., self-driving cars), medical devices (e.g., ML-assisted surgical bots), and compute infrastructures (e.g., AIOps for Cloud/HPC). To realize the promise of such automation, next-generation mission-critical cyber-physical systems, consisting of both ML and non-ML algorithms and technologies, must provide an ever-increasing level of runtime system dependability (i.e., robustness, reliability, safety, and security) that is not available today. The proposed techniques in this chapter can be adapted and applied in such settings. However, significant research is required to integrate these techniques into other domains.

Exploring dependability challenges in collaborative settings. Our work so far only tackles the dependability assessment of autonomous systems working in isolation. However, future autonomous systems are expected to rely heavily on the Internet of Things (IoTs) and to communicate with one another using 5G and other communication technologies. One potential future direction of this work is to extend these techniques to such collaborative settings, where agents (i.e., individual autonomous systems) can help one another to avoid safety issues.

References

1. Erlien SM (2015) Shared vehicle control using safe driving envelopes for obstacle avoidance and stability. PhD thesis, Stanford University
2. Jongsang S, Boemjun K, Kyongsu Y (2016) Design and evaluation of a driving mode decision algorithm for automated driving vehicle on a motorway. IFAC-PapersOnLine 49(11):115–120
3. Nvidia. Nvidia Drive. https://developer.nvidia.com/driveworks
4. Apollo Open Platform. http://apollo.auto. Accessed: 2018-09-02
5. Openpilot git repo
6. Alvarez S (2018) Research group demos why Tesla Autopilot could crash into a stationary vehicle. https://www.teslarati.com/tesla-research-group-autopilot-crash-demo/
7. Economist T (2018) Why Uber's self-driving car killed a pedestrian. The Economist May 29, 2018 https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian
8. Chung K, Kalbarczyk ZT, Iyer RK (2019) Availability attacks on computing systems through alteration of environmental control: Smart malware approach. In: Proceedings of the 10th ACM/IEEE international conference on cyber-physical systems, ICCPS '19, pp 1–12, New York, NY, USA, 2019. Association for Computing Machinery
9. SAE International (2016) Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles
10. Mujica F (2014) Scalable electronics driving autonomous vehicle technologies. Texas Instruments White Paper

11. Banerjee SS, Jha S, Cyriac J, Kalbarczyk ZT, Iyer RK (2018) Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 586–597
12. Leveson N (2011) Engineering a safer world: systems thinking applied to safety. MIT Press
13. California Department of Motor Vehicles. Testing of autonomous vehicles. https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/testing. Accessed: 27 Nov 2017
14. Umit O, Christoph S, Keith R (2007) Systems for safety and autonomous behavior in cars: The DARPA grand challenge experience. Proc IEEE 95(2):397–412
15. Martin B, Karl I, Sanjiv S (eds) (2009) The DARPA Urban Challenge. Springer, Berlin Heidelberg
16. Urmson C, Anhalt J, Bagnell D, Baker C, Bittner R, Clark MN, Dolan J, Duggins D, Galatali T, Geyer C, Gittleman M, Harbaugh S, Hebert M, Howard TM, Kolski S, Kelly A, Likhachev M, McNaughton M, Miller N, Peterson K, Pilnick B, Rajkumar R, Rybski P, Salesky B, Seo Y-W, Singh S, Snider J, Stentz A, Whittaker WR, Wolkowicki Z, Ziglar J, Bae H, Brown T, Demitrish D, Litkouhi B, Nickolaou J, Sadekar V, Zhang W, Struble J, Taylor M, Darms M, Ferguson D (2008) Autonomous driving in urban environments: boss and the urban challenge. J Field Rob 25(8):425–466
17. Stanek G, Langer D, Müller-Bessler B, Huhnke B (2010) Junior 3: a test platform for advanced driver assistance systems. In: Intelligent vehicles symposium (IV), 2010 IEEE, pp 143–149
18. Levinson J, Askeland J, Becker J, Dolson J, Held D, Kammel S, Kolter JZ, Langer D, Pink O, Pratt V, Sokolsky M, Stanek G, Stavens D, Teichman A, Werling M, Thrun S (2011) Towards fully autonomous driving: systems and algorithms. In: 2011 IEEE Intelligent vehicles symposium (IV), pp 163–168
19. Chatham A (2013) Google's self-driving cars: the technology, capabilities, and challenges. In: 2013 embedded linux conference, pp 20–24
20. Chris U (2012) Realizing self-driving vehicles. In: IEEE intelligent vehicles symposium (IV). Alcalá des Henares, Spanien, p 2012
21. Paden B, Čáp M, Yong SZ, Yershov D, Frazzoli E (2016) A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Trans Intell Veh 1(1):33–55
22. Fan C, Qi B, Mitra S, Viswanathan M (2017) DryVR: data-driven verification and compositional reasoning for automotive systems. In: Computer aided verification. Springer International Publishing, pp 441–461
23. Waymo (2017) On the road to fully self-driving. Waymo safety report https://assets.documentcloud.org/documents/4107762/Waymo-Safety-Report-2017.pdf. Accessed: 27 Nov 2017
24. Pearl J (2018) Theoretical impediments to machine learning with seven sparks from the causal revolution
25. Jha S, Banerjee S, Tsai T, Hari SKS, Sullivan MB, Kalbarczyk ZT, Keckler SW, Iyer RK (2019) ML-based fault injection for autonomous vehicles: a case for bayesian fault injection. In: 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 112–124
26. Pearl J (2014) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann
27. Erlien SM, Fujita S, Gerdes JC (2013) Safe driving envelopes for shared control of ground vehicles. IFAC Proc 46(21):831–836
28. Anderson SJ, Karumanchi SB, Iagnemma K (2012) Constraint-based planning and control for safe, semi-autonomous operation of vehicles. In: 2012 IEEE intelligent vehicles symposium, pp 383–388
29. Li G, Li Y, Jha S, Tsai T, Sullivan M, Hari SKS, Kalbarczyk Z, Iyer R (2020) Av-fuzzer: finding safety violations in autonomous driving systems. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE), pp 25–36
30. Jha S, Cui S, Banerjee S, Cyriac J, Tsai T, Kalbarczyk Z, Iyer RK (2020) Ml-driven malware that targets AV safety. In: 2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 113–124

31. Apollo. https://github.com/ApolloAuto/apollo
32. Jia Y, Lu Y, Shen J, Chen QA, Chen H, Zhong Z, Wei T (2020) Fooling detection alone is not enough: adversarial attack against multiple object tracking. In: International conference on learning representations
33. Dosovitskiy A, Ros G, Codevilla F, Lopez A, Koltun V (2017) CARLA: an open urban driving simulator. In: Levine S, Vanhoucke V, Goldberg K (eds) Proceedings of the 1st annual conference on robot learning, volume 78 of Proceedings of machine learning research, pp 1–16. PMLR, 13–15 Nov 2017
34. Anderson James M, Nidhi K, Stanley Karlyn D, Paul S, Constantine S, Oluwatola Tobi A (2016) Autonomous vehicle technology: a guide for policymakers. RAND Corporation, Santa Monica, CA
35. Kalra N, Paddock SM (2016) Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp Res Part A: Pol Pract 94:182–193
36. Laura Blumenthal Marjory, S, Anderson James M, Nidhi K, (2018) Measuring automated vehicle safety: forging a framework. RAND Corporation, Santa Monica, CA
37. Pei K, Cao Y, Yang J, Jana S (2017) DeepXplore: automated whitebox testing of deep learning systems. In: Proceeding of the 26th symposium on operating systems principles, pp 1–18
38. Rubaiyat AHM, Qin Y, Alemzadeh H (2018) Experimental resilience assessment of an open-source driving agent. In: 2018 IEEE 23rd pacific rim international symposium on dependable computing (PRDC). IEEE, pp 54–63
39. Jha S, Banerjee SS, Cyriac J, Kalbarczyk ZT, Iyer RK (2018) AVFI: fault injection for autonomous vehicles. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks workshops (DSN-W), pp 55–56
40. Li G, Hari SKS, Sullivan M, Tsai T, Pattabiraman K, Emer J, Keckler SW (2017) Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, SC '17, New York, NY, USA, 2017. Association for Computing Machinery
41. Jha S, Tsai T, Hari S, Sullivan M, Kalbarczyk Z, Keckler SW, Iyer RK (2018) Kayotee: a fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors. In: Third IEEE international workshop on automotive reliability and Test. IEEE
42. Philip K, Michael W (2018) Toward a framework for highly automated vehicle safety validation. Technical report, SAE Technical Paper
43. Lu J, Sibai H, Fabry E, Forsyth D (2017) No need to worry about adversarial examples in object detection in autonomous vehicles. arXiv preprint arXiv:1707.03501
44. Pei K, Cao Y, Yang J, Jana S (2017) Towards practical verification of machine learning: the case of computer vision systems. arXiv preprint arXiv:1712.01785
45. Lakkaraju H, Kamar E, Caruana R, Horvitz E (2017) Identifying unknown unknowns in the open world: representations and policies for guided exploration. In: AAAI, vol 1, pp 2
46. Saurabh J, Yan M, Zbigniew K, Iyer RK (2021) Assessing risk in dynamic environments for safe driving. Watch out for the risky actors

Personal Reflections

Foreword: Computing and Genomics at Illinois

Gene E. Robinson

Professor Ravishankar Iyer's long and impactful career has led to major advances in computer engineering, especially the design and validation of dependable computing systems. This volume, on the occasion of his 70th birthday, honors the many ways that Ravi has contributed to his field, with leadership, vision, and accomplishment.

Over the past few years, Ravi broadened his scope and has put tremendous energy and focus into improving the computational infrastructure for the life and medical sciences. I was with him in this effort since its early days, and have had the privilege of a rich and impactful collaboration with Ravi in computational genomics over the past ten years. I will briefly tell this story here, and leave it to Ravi's many students, postdocs, and collaborators to describe and honor his many outstanding contributions in systems design.

One of my very first meetings after being appointed director of the Carl R. Woese Institute for Genomic Biology (IGB) in 2011 was with Ravi when he was serving as interim Vice-Chancellor for Research. On our campus, institute directors report to the VCR, and that's what I was there to do: provide an overview of what was going on in the IGB. We had much to discuss—back then the IGB was just a young and up and coming multidisciplinary institute.

But our conversations also took a surprising turn. In addition to asking probing questions about the operation of the IGB, Ravi started inquiring about the status of computational genomics at the IGB, on campus, and in the field more generally. His questions were deep and insightful, and I was delighted. Why? Because advances in genomic biology can proceed only when in tandem with advances in the computational infrastructure that supports genomics, and here was a world- renowned figure in computer system design turning his attention to this! Our regular meetings soon developed a special rhythm: we would move briskly through an account of what was

G. E. Robinson (✉)
Carl R. Woese Institute for Genomic Biology, University of Illinois at Urbana-Champaign, Champaign, IL, USA
e-mail: generobi@illinois.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_25

going on at the IGB in order to save time for spirited discussions on the future of genomics.

Ravi and I developed a keen joint interest in exploring how we might strengthen computational genomics on our campus. We also realized that we had the seeds of a powerful partnership. He had formerly directed the Coordinated Science Laboratory (CSL), a powerhouse research laboratory in the Grainger College of Engineering famous for research on diverse applications of computing and information technology. We started to imagine the possibilities of a campus computational genomics initiative, jointly led by CSL and IGB, which we called "CompGen." Exploratory meetings were held to introduce computer scientists and computer engineers to genomic biologists, and vice versa, and there were encouraging signs of interest and engagement.

Ravi finished his VCR term in 2012, and he was then ready to focus more on developing the CompGen Initiative. An opportunity soon presented itself when campus announced the annual Request for Proposals from the National Science Foundation (NSF) for the Major Research Instrumentation (MRI) Program later that year. We quickly recruited outstanding colleagues—Victor Jongeneel, Steven Lumetta, and Saurabh Sinha—and Ravi inspired us with an exciting vision for a new computing system involving hardware and software to improve the speed of handling and analysis of genomic data. The grant was awarded, and the CompGen Initiative was off and running!

CompGen quickly took shape. A strong partnership was developed between CSL and IGB to run the initiative on behalf of the campus. Thanks to generous support from several parts of campus, numerous graduate fellowships were awarded for projects co-supervised by biologists and faculty from the Department of Electrical and Computer Engineering and the Department of Computer Science. Our goal was to build a robust research community in which the computer scientists and engineers understand the nature of the diverse biology problems that require innovative computational genomics tools, and the genomic biologists understand the growth points in computer science and engineering that could lead to powerful new tools.

The CompGen Initiative achieved its overarching goal and so we ended it in 2021. In its nine-year existence, the CompGen Initiative incubated and launched collaborations that garnered over 15 million dollars in federal and industry funding and contributed substantially to the Illinois research enterprise. CompGen Fellows seeded many new research projects that resulted in publications, patents, and grants. For example, several CompGen faculty and trainees including Ravi, me and Ravi's student Zachary Stephens collaborated on a paper entitled, "*Big Data: Astronomical or Genomical?,"* which showed that the projected future needs in computational genomics exceed even those of astronomy, YouTube, and Twitter. Published in 2015 in *PLoS Biology*, the paper already has accumulated more than one thousand citations. In addition to the NSF MRI grant mentioned above, the CompGen Initiative spawned two campus-wide center grants: a National Institutes of Health Big Data to Knowledge Center of Excellence, and the NSF Center for Computational Biology and Genomic Medicine (CCBGM), led by Ravi and Liewei Wang from the Mayo Clinic, established in 2017.

CCBGM includes a large contingent of engineering, biological, and medical faculty from Illinois, Mayo, and diverse industry partners from the pharmaceutical, technology, and agriculture sectors. Projects provide faculty with the opportunity to improve the computational genomic infrastructure needed to address the pressing needs of the center's industry and medical partners.

The CompGen Initiative has fostered the development of graduate students who have gone on to develop faculty careers at the intersection of healthcare and artificial intelligence. This includes one of the organizers of this volume, Professor Arjun P. Athreya, Ravi's first CompGen student, now an Assistant Professor of Pharmacology in the College of Medicine at the Mayo Clinic. Arjun's doctoral research through close collaborations with Drs. Richard Weinshilboum, William Bobo and Liewei Wang at Mayo Clinic resulted in ALMOND, a machine learning framework to predict individual differences in drug response, which is now being implemented for use at point of care across the Mayo Clinic. And Ravi's collaboration with Mayo's Dr. Gregory Worrell on "21st Century Cybernetics and Disorders of Brain and Mind," which focuses on accelerating the application of machine learning and computing to devices designed to treat epilepsy, also has provided fertile opportunities for other graduate students in the Iyer group. You can read about both of these exciting projects in this volume.

New discoveries in genomics are profoundly changing our view of fundamental biology, human health, and agriculture. Ten years ago, Ravi correctly saw that we were at an important crossroads; while analytical technologies had been evolving at unprecedented speed, biologists were facing major, computational, algorithmic and statistical challenges in the analysis of the massive amounts of genomic data being produced. New computing technologies for genomics were (and are still) needed to better support large-scale analyses of complex biological and biomedical systems, and to transform the data into reliable information for the extraction of new knowledge, or "actionable intelligence," as Ravi is fond of saying. Ravi saw that new developments in computational hardware and software offered new opportunities. I have been honored to partner with Ravi to develop the CompGen Initiative and I am pleased to be able to share this brief account with you as just a small part of his exceptional legacy of accomplishment and impact. Professor Iyer's vision, leadership, and technical contributions have helped provide new multidisciplinary opportunities for students and faculty at Illinois, thus helping to ensure our leadership in both life sciences and computing technologies.

Gene E. Robinson, Urbana

July 30, 2021

An Academic Life Begins and Continues at University of Illinois at Urbana-Champaign

Janak H. Patel

Prof. Ravishankar K. Iyer (Ravi Iyer) came to Illinois in 1983 for a faculty interview with a strong recommendation from Late Professor Edward J. McCluskey. He was a Post-Doc at Stanford with Prof. McCluskey's research group. There were many faculty interviewers in the Computer area in Illinois. The principal interviewers among them were Professors Edward S. Davidson, Jacob A. Abraham and me Janak H. Patel. Ravi came with lots of measured data collected on IBM main frame computers at SLAC (Stanford Linear Accelerator Center) owned by the Department of Energy (DOE) of U.S. Government and run by the Stanford University. His interview talk was on his hypothesis that Main Frame Computers produce errors or complete failures when the computing load is very high. His statistics showed that there was a strong correlation between error rate and the compute load. For traditional Architecture and Fault-Tolerant faculty, this was a novelty in two important ways—(1) Actual Measurement of failure incidences in real computers with real compute loads. And (2) First time claim that failures are dependent on compute load. The other faculty were also impressed. We collectively decided that his strength in measurements will complement most of us theoreticians. An offer for a faculty position was made to him which he graciously accepted.

He joined Illinois in Fall of 1983. He came with his wife Pamela Iyer. I suggested them an apartment complex for their first year stay since I had known many new faculty members before had also stayed in the same complex before buying a house. They found an apartment there and stayed one year before buying a house. We became good friends. He was very social and got along well with all his colleagues.

In later years, he convinced National Aeronautics and Space Administration (NASA) about doing research on Fault Tolerance of Computers. NASA had interest

J. H. Patel (✉)
Donald Biggar Willett Professor Emeritus, Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL, USA
e-mail: janakmail@gmail.com

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_26

in sending computers in space which can withstand radiation induced errors in on-board computers. Ravi Iyer persuaded a Computer Science colleague, Professor Jane Liu to join him in a Center Proposal. Prof. Liu's expertise was in real-time computing that was of interest to NASA for real time control of space crafts and remote satellites. Together the two of them attracted other faculty members and made a successful Center proposal to NASA. He served as Co-Director of the Center named as "Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS)." The Center at Illinois produced many Ph.D.s and research papers under the direction of Ravi Iyer. The Center operated from 1985 to 1998. During this period, he also convinced then well known fault-tolerant computer company, Tandem Inc, to donate one of their computers for measurement of failure data. Tandem was known as the dominant manufacturer of fault-tolerant computer systems for ATM networks, banks, stock exchanges, telephone switching centers, and other similar commercial transaction processing applications requiring maximum uptime and zero data loss. He with his research group injected errors in the bus system to the detriment of its fault-tolerant capabilities. This was a revelation to the company.

Ravi Iyer was very active in international community of Dependable Computing. As a result, he was awarded the 1989 IEEE Fault-Tolerant Computing Symposium in Chicago. As General Chair he attracted a large group of experts in Program Committee and conference organization. The conference had one of the largest attendances at Chicago. Not content with just one NASA center, he successfully obtained a four-year research project from DARPA. He was the Director of that Multi-University DARPA Project on Design for Dependability, from 1994 to 1998.

University of Illinois, awarded him a Chair Professorship in 1998, named "George and Ann Fisher Distinguished Professor" which he continues to hold to this day. In the interim years he held several administrative positions. He was Director of Coordinated Science Laboratory from 2000 to 2008. Later he served at campus level senior position of Vice Chancellor of Research from 2008 to 2011. In later years, Ravi Iyer expanded his research to Bio-Medical and Post Genome research. He became involved with Mayo Clinic and currently holds an Adjunct Professorship with them, where he directs the "Illinois/Mayo Center for Computational Biotechnology and Genomic Medicine."

Ravi Iyer continues to apply his knowledge of Dependable Computing to may devices, notably among them are Surgical Robots and Autonomous Vehicle Control. In both cases he showed the undocumented perils of errors in computing. In these and Mayo clinic research, Ravi has come full circle to his roots in Statistical Inferences that he had employed in early 1980. In modern parlance, his methods are known as Machine Learning.

Throughout his carrier he has received numerous awards. Notably among these are: Fellow of the IEEE, ACM, and AAAS. In addition, he received the IEEE Emanuel R. Piore Award, Humbolt Foundation Award, among many other awards. Ravi continues to do research and advise many graduate students.

Learning from Prof. Iyer

One Conversation at a Time

Wen-Mei Hwu

In February 1987, I met Prof. Ravi Iyer for the first time during my interview for an Assistant Professor position at the Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign (ECE Illinois). At the time, I was completing my Ph.D. dissertation describing the design of an out-of-order execution microarchitecture, which would later be adopted into the Intel Pentium Pro microprocessor, the world's first mass-market microprocessor that performs out-of-order execution. The main contribution of my dissertation work was an efficient mechanism for recovering of the in-order execution state so that the exceptions and faults can be handled precisely in an out-of-order execution microarchitecture. My work targeted performance improvement by issuing multiple instructions in parallel and overlapping the execution of instructions with long latencies. I had little understanding of the work being done in the fault-tolerant computing community.

During my individual meeting with Ravi, he was extremely collegial. Rather than trying to show me how little I knew about fault-tolerance computing, he kindly discussed opportunities for me to collaborate with faculty and students in the NASA ICLASS that he co-directed with Prof. Jane Liu of the Computer Science Department. In particular, he expressed genuine interest in having a computer architect to complement the then existing strength in fault tolerance and testing in ECE Illinois. This genuine collegiality and strategic thinking helped attract me to Illinois.

After I joined ECE Illinois, Ravi immediately invited me to join the NASA ICLASS center and started to support my first Ph.D. student Pohua Chang. We used the funding to work on a new compiler technique called superblock for extracting instruction-level parallelism and fully utilizing superscalar and VLIW processors. When I was preparing for my first presentation to the NASA visitors during an

W.-M. Hwu (✉)
Senior Distinguished Research Scientist, Senior Director of Research, NVIDIA; Professor and AMD-Sanders Chair Emeritus, ECE, University of Illinois at Urbana-Champaign, Champaign, USA
e-mail: w-hwu@illinois.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_27

annual review, I felt awkward that my research was off the focus of the center—to investigate novel fault-tolerant computing techniques for future space missions. Obviously, there was not strong connection between the work done by Pohua and the focus of the center. I discussed my anxiety with Ravi. His advice was to make an honest presentation on the vision—enabling extremely low-power computing for embedded applications, including space mission applications. During the review, Ravi also presented the work as part of the center vision. It was well-received by the NASA visitors.

One important insight that I gained from the review was the level of challenge in certifying a computing device for space missions. It is extremely costly to certify that a computing device is suitable for the harsh radioactive and temperature environment of a spaceship. As a result, the computing devices deployed in space missions are typically a decade or more behind the cutting edge of the commercial market. As a result, NASA is always interested in compiler techniques that can simplify future hardware and/or enhance the performance of existing hardware.

The work was supported by the ICLASS center in the following four years. The publications from the work accumulated more than 2000 citations and received multiple recognitions such as the International Symposium on Computer Architecture (ISCA) Most Influential Paper Award. The impact of the generous support from Ravi on my career cannot be overstated. The work also impacted the design of and compilers for the VLIW DSP processors that have been adopted into billions of mobile devices.

A more subtle impact from Ravi is how he helped cultivate a whole generation of ECE faculty members who later created mega centers at the University of Illinois. Through our daily interaction with Ravi, faculty members like Bill Sanders, David Nicol and I learned how to think big while taking care of the details of a large center. For example, I was a terrible negotiator when I started my career at Illinois. After 34 years of learning from Ravi, I finally feel that I can do reasonable negotiations with both the funding agencies and the University administration when it comes to large research contracts.

I have only had a small number of opportunities co-authoring papers with Ravi. Through my limited number of co-authoring activities with him, I have always been impressed with Ravi's ability to pull back from the details and bring up the concerns from 20,000 miles away. He has a unique capability to help the students to see the concerns from readers who may not be fully invested in the subject area. This has impacted my interactions with my own students in a very subtle but important way.

Another important lesson that I learned from Ravi is to when and how to use math in my own research. Very few people know that I have a minor in statistics as part of my Ph.D. program at the University of California, Berkeley. However, until I started to interact with Ravi, I did not find a good way to use my math skills in my research. By observing how Ravi's use of math in his research, I learned how to use math productively. This has become very important when I worked on accelerating machine learning applications using GPUs. I am grateful to Ravi who demonstrated the practical use of math skills so naturally to all of us.

When I started to work on GPU Computing technology in 2006, most people thought that I was going stray down a rabbit hole. When I started writing the "Programming Massively Parallel Processors—A Hands-on Approach" textbook with David Kirk in 2008, most people thought that I was wasting my time. However, Ravi provided me with strong encouragement as a colleague. To date, the book is in its 4th edition with more than fifty thousand copies sold worldwide. The book has also been an academic success and accumulated more than 3600 citations. He also offered his enthusiastic support when I created the ECE498 and ECE598 courses that were later adopted by more than 100 universities.

It would be my failure if I did not mention the sense of humor that many of us managed to rub off from Ravi. It is through so many casual social conversations that I learned from Ravi not to take myself too seriously. His unique way of injecting laughter into even the setbacks has been inspiring to many of us. Afterall, as we try to reach big goals, we are bound to have more setbacks. As Ravi would say, there are many more failed shots than goals in any good soccer game. Being able to find humor in these failed shots is so important for anyone who ultimately make the goals.

I would like to finish by crediting Ravi for his keen sense of the potential candidates. Over the past two decades, Ravi and I agreed on supporting several faculty candidates whom our department unfortunately decided to pass on. All of them later became extremely successful. Ravi taught me how to ask the type of questions during interview seminars and individual meetings to get to know the true potential, not just the work that has already been done, of the candidate. For example, a surprising number of candidates do not provide high-level intuition and simple examples for people to truly understand the key ideas and potential generalizations of their work. Using a simple example to ask about the intuition of the work can help both the presenter and the audience to fully grasp the important contributions of the work. It was through numerous candidate seminars and private conversations that Ravi managed to teach me how to see the potential of a person that may not be even clear to the person himself/herself. This skill has fundamentally impacted how I mentor my students.

In so many ways, I have been effectively a postdoc learning from Ravi for the past 34 years. Each conversation with Ravi elevates my understanding of research and humanity. At 70, Ravi is going stronger than ever with impressive initiatives such as NSF CCBGM and NSF PPoSS. I look forward to the many more conversations with Ravi in the years to come!