UMEÅ UNIVERSITY

Machine Learning-based Diagnostics and Observability in Mobile Networks

Tobias Sundqvist

Doctoral Thesis, March 2023
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

sundqtob@cs.umu.se

ISBN 978-91-8070-053-5 (print)
978-91-8070-054-2 (digital)
ISSN 0348-0542
UMINF 23.02

Printed by Cityprint i Norr AB, Umeå, 2023

"As the anomalies, I hope this thesis significantly deviates from expectations!"

Abstract

To meet the high-performance and reliability demands of 5G, the Radio Access Network (RAN) is moving to a cloud-native architecture. The new microservice architecture promises increased operational efficiency and a shorter time-to-market, but it also comes with a price. The new distributed and virtualized architecture is far more complex than ever before, and with the increasing number of features it brings, troubleshooting becomes more difficult. So far, RAN troubleshooters have relied on their expertise to analyze systems manually, but the ever-growing data and increased complexity make it challenging to grasp system behavior.

This thesis contributes threefold, where the proposed machine learning and statistical methods help RAN troubleshooters find deviations in system logs, identify the root cause of these deviations, and improve the system's observability. These methods learn the application's behavior from the system logs events and can identify behavior deviations from many different aspects. The thesis also demonstrates how observability can be improved by using a new software instrumentation guideline. The guideline enables the tracking of systemized procedures and enhances system understanding. The purpose of the guideline is to make RAN developers aware that machine learning can utilize debug information and help their troubleshooting process. To familiarize the reader with the research area, the challenges, and methods that can be used to detect anomalies, perform root cause analysis and observe RAN system behavior. The proposed research methods are integrated and tested in an advanced 5G test bed to evaluate the methods' accuracy, speed, system impact, and implementation cost.

The results demonstrate the advantage of using machine learning and statistical methods when troubleshooting the behavior of RAN. Machine learning methods, similar to those presented in this thesis, may help those who troubleshoot RAN and accelerate the development of 5G. The thesis ends with presenting potential research areas where this research could be further developed and applied, both in RAN and other systems.

Sammanfattning

För att möta de höga kraven på prestanda och tillförlitlighet i det nya mobila 5G nätet sker nu en övergång till en molnbaserad arkitektur i radioaccessnätverket (RAN). Den nya mikrotjänstarkitekturen är tänkt att öka skalbarheten, prestandan och korta ner ledtiderna för produktleveranserna. Den distribuerade och virtuella arkitekturen är däremot mer komplicerad än tidigare och medför att det blir svårare att felsöka. Hittills har de som felsökt RAN förlitat sig på sin expertis för att manuellt analysera systemet. Men den ständigt växande datamängden och den ökade komplexiteten gör det svårt att förstå systemets beteende.

Denna avhandling bidrar med kunskap inom tre närliggande områden, där de föreslagna maskininlärnings- och statistiska metoderna hjälper de som felsöker RAN att hitta avvikelser i systemloggar, hjälper till att identifiera grundorsaken till dessa avvikelser och förbättrar systemets observerbarhet. Dessa metoder lär sig RANs beteende utifrån händelser i systemloggar och kan identifiera ett antal beteendeavvikelser. Avhandlingen visar också på hur observerbarheten kan förbättras genom att använda en ny riktlinje för mjukvaruinstrumentering. Riktlinjen gör det möjligt att följa hur RANs applikationer påverkar varandra vilket i sin tur förbättrar systemförståelsen. Syftet med riktlinjerna är att göra dem som arbetar med RAN medvetna om hur maskininlärning kan hjälpa till i deras felsökningsprocess. För att bekanta läsaren med forskningsområdet diskuteras först utmaningarna och metoderna som kan användas för att upptäcka avvikelser i RAN data, orsaken till avvikelserna samt hur observerbarheten av systemet kan förbättras. För att utvärdera de föreslagna metodernas noggrannhet, hastighet, systempåverkan och implementeringskostnad, integrerar och testas metoderna i en avancerad 5G-testbädd.

Resultatet visar på de stora fördelarna med att använda maskininlärning och statistiska metoder vid felsökning av beteendet hos RAN. Maskininlärningsmetoder, liknande de som presenteras i denna avhandling, kan komma att hjälpa dem som felsöker RAN och påskynda utvecklingen av 5G. Avhandlingen avslutas med en presentation av potentiella forskningsområden där forskningen i denna avhandling skulle kunna vidareutvecklas och tillämpas, både i RAN men även i andra system.

Preface

This thesis begins with a short description of the 5G Radio Access Network (RAN) and discusses the challenges associated with identifying anomalies and root causes for RAN troubleshooters. A summary of this thesis contribution is then presented in the five included papers:

Paper I **T. Sundqvist**, M. Bhuyan, J. Forsman, and E. Elmroth. Boosted Ensemble Learning for Anomaly Detection in 5G RAN. *IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI 2020)*, pp. 15-30, 2020.

Paper II **T. Sundqvist**, M. Bhuyan, and E. Elmroth. Uncovering Latency Anomalies in 5G RAN - A Combination Learner Approach. *14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, IEEE, pp. 621-629, 2022.

Paper III **T. Sundqvist**, M. Bhuyan, and E. Elmroth. Unsupervised Root-Cause Identification of Software Bugs in 5G RAN. *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC 2022)*, IEEE, pp. 624-630, 2022.

Paper IV **T. Sundqvist**, M. Bhuyan, and E. Elmroth. Robust Procedural Learning for Anomaly Detection and Observability in 5G RAN. *Submitted for publication*, 2023.

Paper V **T. Sundqvist**, M. Bhuyan, and E. Elmroth. Bottleneck Identification and Failure Prevention with Procedural Learning in 5G RAN. *23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Accepted for publication, IEEE/ACM, 2023.

In addition to the papers included in this thesis, the following article has been produced during the author's Ph.D. studies:

- **T. Sundqvist, C. Blöcker, T. Kampik**, M. Bhuyan, and E. Elmroth. Detecting Anomalies in SS7 Network Traffic: Towards a Holistic Approach. *Not published.*

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

Acknowledgements

My journey in the telecom business started in 2000 when I finished my Master of Science Studies at Umeå university. I then believed I was ready to solve all of the world's problems. But 20 years later, I realized I needed to return to the university again. The truth is still out there...

I am very grateful to Tietoevry for allowing me to grow in many areas, recently as an Industrial Ph.D. student in the Wallenberg AI, Autonomous Systems and Software Program (WASP). With the tremendous support of **Rolf Konradsson** and **Clas Högvall**, I have been able to focus on my research and open up a new machine learning area at Tietoevry. Special thanks to **Johan Forsman** for supporting and believing in me during this journey; without your out-of-the-box ideas, I would not have been able to be successful in this special area of research. I am also thankful to have such great colleagues such as **Håkan**, **Sofia**, **Olov**, **Robert**, and **Simon**, whom I have bothered many times during my brainstorming sessions. I am most grateful to my team members in SCUBA: **Christoffer**, **Ludvig**, **Iris**, **Salih**, **Aida**, and **Sahar**, who have supported my work and helped my find my way back from the long and lonely research process.

I am immensely grateful to my supervisor **Erik Elmroth**, who allowed me to join the Distributed Systems group and supported me through this long Ph.D. journey. I have grown as a researcher thanks to your experiences, guidance, and network. I now understand the importance of being at the front line in research to solve the many problems in industry applications. But I would not have completed this journey without the tremendous support from my co-supervisor **Monowar Bhuyan**, who has helped me day and night from the start of my Ph.D. journey. I am immensely grateful that I have had a chance to learn more about you and your family and help in other areas than research. Thanks also to **Henrik Björklund** for being my go-to person when I needed another angle on my research and for helping me understand the academic world. I am also grateful for all of the events, workshops, lunches, and other interesting meetings I shared with the members of the ADS Lab. I wish you all the best of luck in your future research. Additional thanks to the IT-support unit, especially **Tomas** and **Mattias**, who have helped me on many occasions. I would also like to sincerely thank **Anne-Lie** and **Helena**, who provided great administrative support during my Ph.D. studies.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. What was even more valuable than the funding was the invaluable network of people who are part of WASP. Thanks to all the WASP seniors for the great courses, conferences, and international study trips you have arranged. Also, thanks to all WASP batch 2 students for all the fun moments we shared, especially **Timotheus** and **Christoffer**, who made the WASP journey joyful and memorable. I would also like to thank **Martin**, who sacrificed himself in front of the self-driving car we developed during the WASP summer course. This helped me realize that no matter how difficult a task may be, it can always be solved piece by piece.

Lastly, I would like to thank my wife **Johanna**, who has been my guiding star throughout my life. Thanks to your cheerful advice and support, I survived the loneliest parts of this journey, especially during the pandemic. You help me see through the matrix of zeros and ones that sometimes prevents me from seeing the whole picture, and you simply make me a better person. You have also saved me countless times whenever I try to do too much for others instead of prioritizing my own needs and my family. I also adore my kids: **Alvin**, **Meja**, and **Julie**, who inspire me to play every day and bring so many moments of joy to my life. Also, thanks to my wonderful **family** and **friends**; you fill my life with more than research and work. I love you all!

Contents

Chapter 1

Introduction

During the past decade, the ever-growing mobile network has made us dependent on constantly being connected to the Internet and each other. Today, there are more wireless devices than people on earth [ana22], and a majority of the equipment is dependent on the Radio Access Network (RAN). To meet the never-ending need for faster and more reliable communication, RAN is now facing a major transformation. This chapter provides a brief overview of 5G RAN, discusses the challenges in troubleshooting RAN, and explains how the research can help detect problems in RAN.

1.1 Research Motivation

The base stations that serve as the main communication points in RAN have become very complex with the development of 3G/4G/5G and contain several million lines of code [Eri20]. Despite the rapid growth of RAN, where parts are distributed and virtualized, troubleshooters still manually analyze application logs to identify faults. As thousands of microservices work together to fulfill the many requirements, it is essential to strengthen the troubleshooting process in three domains:

Anomaly detection. The process of detecting deviation from normal behavior. The increasing number of features changes the behavior in daily deliveries, and it is difficult for troubleshooters to keep up with all updates. The complex microservice interactions and the ever-growing data make distinguishing between normal and abnormal behavior difficult. Static tests can catch the most obvious faults, but the behavior fluctuations in time, performance, and interactions are almost impossible to identify manually in large systems such as RAN [He+21].

Root cause analysis. The intricate process of finding the root cause of a problem. The root cause analysis is similar to detective work: the anomalies and behavior of RAN are the clues needed to find a solution. However,

not all of us can be Sherlock Holmes[1], not all troubleshooters can remember every single event and understand the complex interaction between microservices. Instead, several troubleshooters work together to narrow the problem area, and the success in finding the root cause relies solely on the expertise of the troubleshooters involved [Li+22a].

Observability. The ability to understand the system's behavior from the retrieved data. The key to detecting anomalies and their root cause is to provide good observability of the system. Unfortunately, storing everything that occurs in RAN is impossible, and developers must choose how and what to store. The debug information provided by individual developers is a diverse mix of information that can aggravate the understanding of RAN. The new 5G RAN microservice architecture increases complexity, and similar to other industries, there is a lack of useful tools to aid end users in understanding the system's behavior [Zho+21; Nie+19].

RAN vendors often point to Artificial intelligence (AI) as a solution that will enhance 5G RAN in areas such as energy saving, load balancing, and mobility optimization [Jit+21; 3gp22; Co19]. Less is mentioned about the challenges that RAN troubleshooters face when complexity increases. As time-to-market is crucial for RAN vendors, troubleshooters also need new tools to identify faults more quickly and accurately.

1.2 Radio Access Network

To fully understand the complex development pipeline that developers and troubleshooters of RAN deal with, this chapter provides a high-level presentation of the mobile communication network.

Traditionally, a mobile telecommunications network consists of four major domains:

- The **device**, often called User Equipment (UE), can be a smartphone, laptop, car, industry robot, etc.
- The **Radio Access Network (RAN)**, uses radio frequencies to provide wireless connectivity among the UEs.
- The **core network (CN)** is the infrastructure that provides connectivity to the Internet and an any-to-any connection to all UEs connected to the network.
- The **transport network (TN)**, provides connectivity between the RAN and the CN.

Generally speaking, RAN uses a radio unit to generate and receive signals to all wireless devices. The radio unit is often close to the antenna to support low

[1] https://en.wikipedia.org/wiki/Sherlock_Holmes

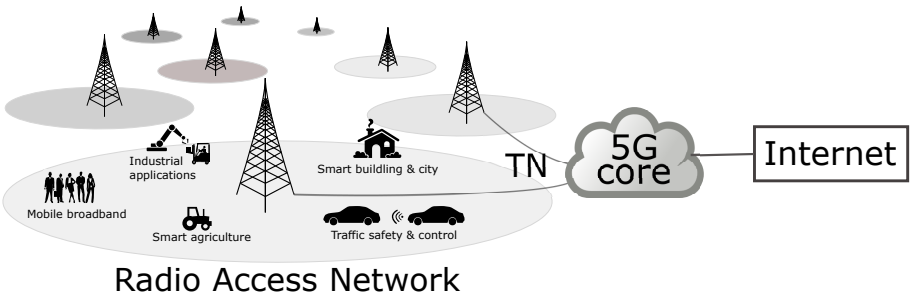

Figure 1.1: A mobile telecommunication network.

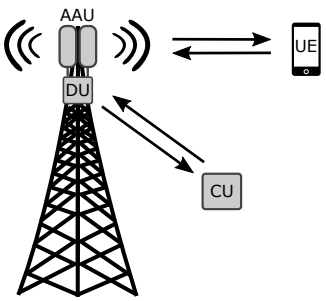

Figure 1.2: Example of communication between the antenna, user equipment (UE), distributed units (DU), and centralized units (CU).

latency, and a baseband unit converts the signals to and from the core network. A physical support system also contains an electrical power system, a backup battery, transmission equipment, and a cooling system. In 5G mobile networks, the radio unit is integrated into the antenna and is called the active antenna unit (AAU). To support low latency and high throughput, the baseband is split into distributed units (DU) that are close to the antenna and centralized units (CU) that are further away from the antenna (see Fig 1.2). Each physical unit contains complex software, and thousands of microservices work together to meet the many requirements.

RAN developers work in teams, each responsible for a small part of the functionality. The teams ensure their functionality works as expected in several test steps, where small parts are tested separately and are successively built together. A continuous integration loop handles the building, integration, and testing, which ensures the functionality passes all predefined tests (see Fig 1.3). Many parts of the software interact to fulfill the services in RAN, and several troubleshooters may be needed to identify the root cause(s) of problems.

Ideally, each software update should be tested by each step several times to make sure that the update is stable. Unfortunately, each step can take

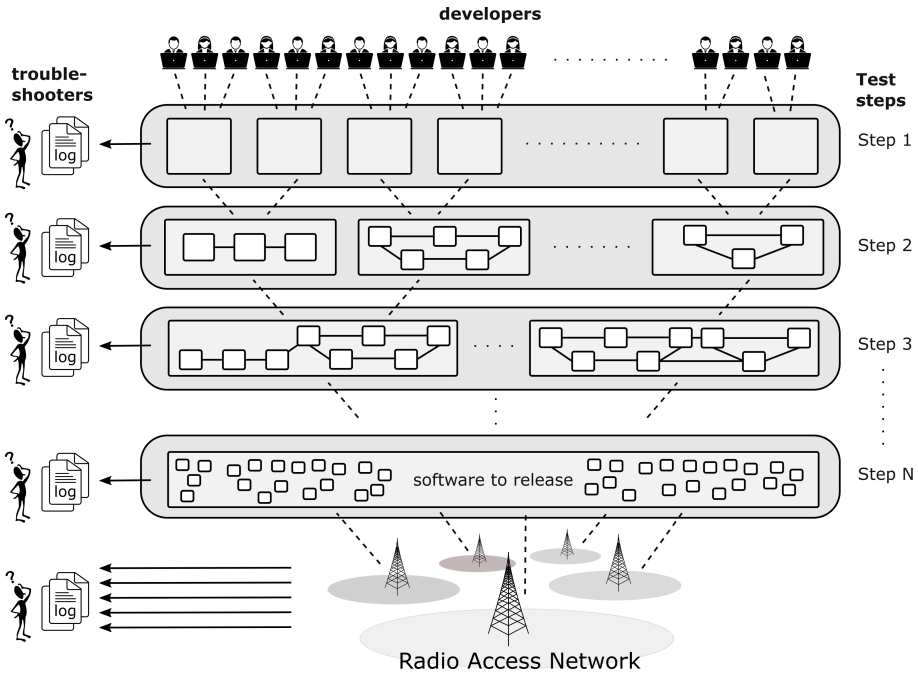

Figure 1.3: Thousands of engineers are part of the development of a RAN. Software is tested in steps where an increasing number of microservices are tested together.

hours, and several updates are made daily. Therefore, several updates are tested together to shorten the time-to-market for new functionality and to facilitate software bug correction. The drawback is that it can be challenging to determine which update caused a test to fail. Further complicating matters, some faults occur occasionally, and tests may give a passing result in the first software update but a failing result in a subsequent, non-relevant update. Furthermore, foreseeing and testing all possible configurations and scenarios in live networks is impossible. This means that problems will also appear in live networks, and many troubleshooters need to be involved in identifying the underlying problem. The events in the logs reflect the functional behavior of each microservice and can, for example, be events for sending or receiving signals, state updates, parameter print out, alarms, error messages, etc. Skilled troubleshooters memorize the essential events in the logs and recognize the many scenarios that can arise. Unfortunately, extra functionality is added for each software release, making it even harder to distinguish between normal and abnormal behavior.

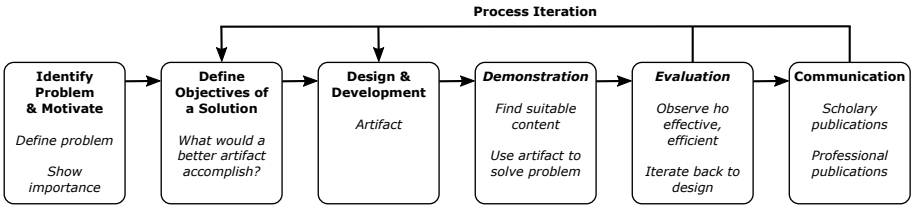

Figure 1.4: DSR methodology used in this thesis (adapted from [Pef+07]).

1.3 Research Objectives

This thesis aims to develop new machine learning (ML) and statistical methods to help troubleshoot RAN in the domains mentioned in Section 1.1: anomaly detection, root cause analysis, and observability.

The fault identification process has always relied on the expertise of the troubleshooters that analyze the behavior of RAN through metrics and system logs. As RAN has grown over many years of development, troubleshooting has become much more difficult, and the high-level research objectives are, therefore:

RO1 : To develop, implement, and evaluate machine learning methods for the detection of anomalies in system logs.

RO2 : To develop, implement, and evaluate machine learning methods to identify root causes of such anomalies from data in system logs.

RO3 : To develop software instrumentation guidelines that suit the ML needs and help improve the observability of RAN.

1.4 Methodology

The methodology in this thesis follows a classical design research model (DSR); see Fig 1.4. The problems troubleshooters encounter in RAN are first identified together with a definition of the addressed research problem. In this step, a review of the relevant literature is also conducted to gain a better understanding of the problem and existing solutions. In the second step, the objectives of the solution are defined in a purely quantitative step (e.g., the solution is compared to existing solutions). The third design and development process involves the creation of novel methods or new techniques to address the research objectives. Throughout this research, an advanced 5G testbed is utilized to demonstrate and evaluate the performance of the proposed contribution and compare it to existing state-of-the-art solutions. Each paper contains detailed information on the testbed and the research contribution.

1.5 Research Contributions

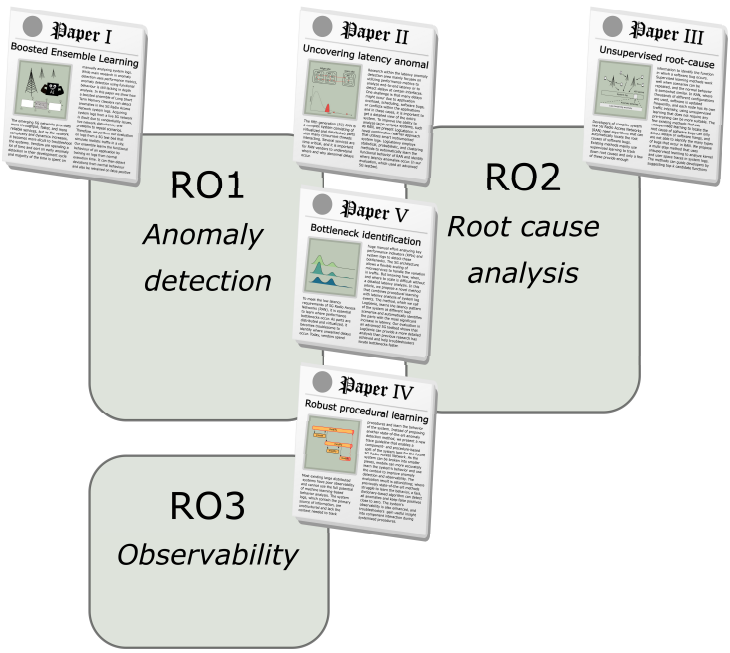

Figure 1.5: Main contribution of the thesis.

This research contributes to new machine learning and statistical methods that detect anomalies and their root causes from RAN system logs. It also outlines a path to increase observability in RAN and improve ML methods by enhancing software instrumentation. An illustration of how the five papers in this thesis contribute to the **ROs** is provided in Fig 1.5.

Paper I addresses **RO1** by introducing a novel ensemble of Long Short-Term Memory Models (LSTM) that learns the sequence order of events in the system log. The method can predict the behavior and identify when the sequences of events are abnormal. Paper II continues to address **RO1** and **RO2**, by combining three methods to detect small latency deviations between events in system logs. This results in a supervised learner that can provide a detailed analysis of where latency anomalies occur in large complex systems. Paper III utilizes the kernel and user space events in system logs to address **RO2**. By learning the pattern between the system log events added by designers and the function call chains, a top candidate list of possible areas that contain SW bugs can be provided. To further address **RO1**, Paper IV presents a new software instrumentation guideline that enables new ML methods to detect anomalies with very high accuracy in an advanced 5G testbed. **RO2** and **RO3** are also addressed by demonstrating how developers can better understand

the system and more quickly identify anomalies when using these guidelines. Finally, **RO1** and **RO2** are addressed in Paper V, by combining the latency detection technique from Paper II with the procedural learning proposed in Paper IV. This allows us to identify abnormal latency delays and bottlenecks as load increases in a complex system. The thesis further demonstrates how failures can be prevented by dynamically scaling the system at the identified bottlenecks.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 and Chapter 3 summarize the anomaly detection and root cause analysis area that are of interest for RAN. Chapter 4 explains the importance of understanding the system's behavior and how it can be improved in RAN. Finally, Chapter 5 summarizes the contributions of each research paper, and Chapter 6 outlines potential pathways for future research based on the results of this thesis.

Chapter 2

Anomalies in RAN

An anomaly is something that deviates from what is standard, normal, or expected. In this thesis, the functional behavior of RAN is the expected behavior as tested by developers. This chapter explains the challenges in troubleshooting RAN and what data and approaches are suitable for detecting RAN anomalies.

2.1 Troubleshooting Challenges

RAN developers face a number of challenges when troubleshooting the systems. Below are some of the challenges addressed throughout this thesis:

Difficulty catching anomalies using strict threshold values. All implemented functionality in RAN meets some kind of requirement. From a systematization point of view, there could be time constraints, support of protocols, new hardware support, etc. From a developer's perspective, there could be the support of a new signal, an update of attributes in the software, etc. The requirements are then the expected behavior; when requirements are not fulfilled, they can be treated as an anomaly. However, identifying anomalies is not as black and white as fulfilling the requirements. Even if a requirement states that a phone call should be able to be established within 1 second, there could be an abnormal behavior if the process takes 0.99 seconds. What if all other phone calls are setup within 0.1 seconds? The behavior could then be classified as abnormal and it may be interesting to know why there was a long delay.

Complex dependencies. Most of the RAN functionality depends on other functionality, and one anomaly may cause problems in many areas. A lack of understanding of the system may also cause new problems when software faults are resolved, or new functionality is added. Each software update may also affect the resource utilization or the interaction between the microservices, which may lead to performance issues in the long run.

Too many configurations. A large problem in RAN is that many kinds of hardware equipment need to be supported, and each piece of equipment requires a special configuration to work as intended. As there are hundreds of parameters to configure, testing all combinations of parameters and different hardware sets is impossible. In the end, there is only time to test a few requirements, and several software updates are tested simultaneously.

Occasional faults. Some faults occur only occasionally and can be challenging to repeat. Together with the many configurations that are not tested, this causes a large slip-through of anomalies in the live RAN network. The later a fault is found, the more costly it is for RAN vendors, who often receive negative feedback from customers and need to spend a lot of time on troubleshooting, software correction, and testing. Therefore, a vendor's success is partly based on how fast and early anomalies can be detected in the development cycle [Dam07].

A large amount of data. Every second, hundred of events occur in the system logs, and understanding the difference between normal and abnormal behavior can be very challenging. Small variations in normal behavior may also occur, which can cause confusion among troubleshooters. The data analyzed often represents the behavior when a fault occurs, and it is up to the troubleshooter to determine the normal behavior.

2.2 Data Monitored in RAN

Anomalies can be present at many different levels in RAN. A person who makes a phone call or watches a video may experience disturbances. The operator can, in such cases, receive metrics indicating a high amount of dropped calls or poor utilization from the faulty node. From a troubleshooting perspective, alarms or erroneous events can be written in the application's system log. As the research in this thesis aims to help the RAN troubleshooter, the possible input data to analyze is first explored:

- **Key Performance Indicators (KPI):** These metrics describe the radio network performance during a certain time window, typically 15 minutes. Abnormal KPI values could, for example, be an indicator that calls are dropped, poor subscriber quality, or other performance-related issues. RAN operators and vendors monitor KPI values to analyze the performance of the RAN.

- **System-specific parameters:** Most functionality in RAN causes the state of an object to switch between states or utilize a particular resource. When requested, these parameters can be dumped into a log file and examined by the troubleshooters.

- **Alarms:** Alerts that a certain part of the system is not working as intended. In many cases, alarms can be the first sign of an anomaly. The alarms are stored in a separate alarm log, revealing the time each alarm occurred.
- **System logs:** Important application events are stored in a log to understand the system's behavior. Developers design software instrumentation to store signals, state updates, alarms triggered, resources requested, parameters, etc. In large systems like RAN, it is possible to activate different levels of instrumentation to retrieve information from specific microservices or functionality.

Distributed tracing is also a category of input where the time to complete services and microservice interactions is monitored. This concept is new and requires extensive integration through the whole system to work properly. To my knowledge, there is not yet any support for distributed tracing in RAN. The distributed tracing concept is therefore only briefly covered in the remainder of this thesis. Some anomalies can also be related to memory usage, CPU usage, throughput, signal buffers, etc. Although these parameters are monitored in RAN, they are rarely used in troubleshooting.

KPIs provide network performance statistics for each node in RAN and have been utilized to detect anomalies and optimize systems [Sun+19; ZPH19; Hu+21; MHH18]. Still, it would be more beneficial for vendors to detect anomalies earlier in the development chain. A more helpful source of information is the system log, which is present during the whole development cycle and contains information about the application's behavior. The scope of this thesis is limited to utilizing the features extracted from system logs and analyzing them from a functional point of view.

2.3 Anomalies in System Logs

The behavior of applications is reflected as events in the system log. When functionality fails, it can appear as an event deviation. The illustration in Fig 2.1 provides an example of how these deviations can be expressed in the system log.

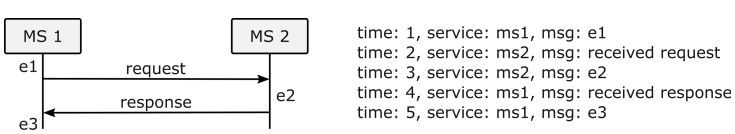

Figure 2.1: Simplified example of signaling between two microservices and the system log.

The events *e1*-*e3* represent the storage of information as two microservice interact. The sequence starts with an event (*e1*) for the first microservice

(*MS1*); this typically triggers a request to another microservice (*MS2*). This request could be postponed for some reason or not sent at all, and another event could then appear in the log. As the request is received by *MS2*, an event is written in the log. If too many requests are handled by *MS2*, there could be an abnormal delay before the event is written in the log. The receiver can also be incapable of handling the signal depending on its state, signaling version, or full signaling queues. If the request is handled by *MS2* it can trigger a status of resource update through the event *e2*. However, a resource shortage may lead to an exception instead of the event *e2* and a reject may then be sent back to *MS1* instead of a confirmation signal. Finally, *MS1* stores an event (*e3*) in the log with the outcome of the response.

To summarize the above example, anomalies can be detected as missing events, extra events, or delayed events. If the events normally follow a certain pattern, a change in the sequence order may also be treated as an anomaly. In the example, a delay was mentioned between two events, but it can also be worthwhile to examine the time to complete the whole request from *e1* to *e5*. If the delay occurs occasionally, it may also be interesting to determine the frequency of the delays.

2.3.1 Feature Extraction from Logs

The system logs are mainly written to be understood by developers and troubleshooters. In order to use the log data in ML methods, one must be able to decide what features may be essential to extract. This typically requires feedback from system experts familiar with the system log data. The features can then be extracted using regular expressions that select the important parts and filter out the noise. Unfortunately, it is more difficult to do this in large systems like a RAN, where thousands of developers choose to instrument the code based on their needs. Each part can require a unique regular expression, and as the software is updated, there can be a need to adapt the expressions to extract the correct information. Methods like Drain [He+17] can analyze the logs to select the correct regular expression and adapt when the software is updated. A strict software instrumentation guide is followed throughout this thesis papers to unify all events and minimize parsing errors. This is discussed in more detail in Chapter 4.

In the end, features such as timestamps, messages, object identifiers, or signals sent can be extracted from the system logs. These features are then converted to numbers from which the ML methods learn the behavior.

2.4 Anomaly Detection Approaches

Both statistics and ML provide a variety of methods that can be used to detect system log anomalies. This thesis mainly focuses on ML methods but statistical methods are in some cases used to further enhance the results. The following

sections provide an overview of approaches that can be used individually or in combination to detect anomalies in system logs.

2.4.1 Statistics

Statistics can approximate the world mathematically and help explain the relationship between variables. By comparing statistical features between samples of data, it is possible to detect anomalies using two different approaches:

- The first approach counts the occurrences of specific events in the log. It could, for example, be the number of times there is an error, parameter update, specific signal, total amount of events, or the features relevant to the anomaly detection approach. The occurrence of some events may also be correlated, and deviations in both metrics and correlations may help identify the abnormal behavior.
- The second approach utilizes the timestamp in system logs to measure the time spent between certain events. It could, for example, be between two sequential events, the time to complete a service, how often the events occur, etc.

The statistical findings in both approaches can also be utilized to evaluate whether the system requirements are fulfilled. The requirements could specify that a service should be completed within a specific time or that an application should be able to handle a certain number of requests per second.

2.4.2 Machine Learning

ML also uses statistics, but instead of using statistical variables to count the probability or compare data, it predicts the outcome through a model based on input data. The ML model can, for example, predict time series data, classify data, or reduce the number of variables in a data set. In anomaly detection, ML is often categorized by how the model learns from the data. There are then three main approaches:

- **Supervised learning** - The data used as input and output is then "labeled". This means that the input data has predefined tags with the correct output. The ML models can then train and update themselves when the output is incorrect and, in the end, produce the correct output.
- **Unsupervised learning** - Instead of defining the correct output given certain input data, unsupervised learning finds hidden patterns and insights in the analyzed data. The chosen method defines whether the data should be grouped or separated to detect outliers.
- **Semi-supervised learning** - A small part of the data is labeled, and a large part is unlabeled. A supervised learning model then trains on

the labeled data and predicts the outcome for the unlabeled data. By combining this prediction with unsupervised learning, it is possible to improve the supervised learning model.

Hybrid learning is sometimes mentioned as a fourth category. This concept is similar to semi-supervised learning, where any of the three categories above are combined into a hybrid learner. Many different methods can detect anomalies in logs, and most of them use either a supervised or unsupervised approach [Lan+22; YKD20; ZJM22; He+21]. This thesis explores both approaches, and they are covered in more detail in Section 2.4.3 and Section 2.4.4.

2.4.3 Supervised Anomaly Detection

Supervised models require data to be labeled. From an anomaly detection point of view, it is essential to know if the events in the log correspond to abnormal or normal behavior. Labeling each event in large logs is time-consuming and requires system experts to go through the log. A common approach is instead to automatically label the events as abnormal if they contain a system-specific error message [OS07]. System experts can then provide a list of known error messages, or researchers can make a best guess based on the severity of the message. This technique may miss erroneous events that do not contain the special error message or falsely label normal events as abnormal. There can also be delays, a burst of events, or sequential problems that do not appear as special error events but should be considered as anomalies. Cinque et al. report, for example, that only 40% of the software faults in real-time systems leave a visible error in the logs [CCP12]. An alternative to considering each event is to mark a complete sequence of events as abnormal [Zhu+18]. In such cases, the anomaly is pinned to a specific period of time, even though it is unknown exactly when and how the anomaly is expressed in the log. A final alternative is to train the ML model on data from either successful or unsuccessful scenarios. The models then learn either normal or abnormal behavior and can detect when the behavior deviates from the training data.

Paper I and **Paper IV** present supervised models that train on data from successful scenarios and then identify the abnormal events in unseen data. The chosen approach is suitable for continuous integration loops where functionality is tested repeatably. Logs from successful test cases are then treated as normal behavior, and a model can be trained to find deviations in unsuccessful tests.

The method used when detecting the anomalies is dependent on the data labeling. If each event is labeled, the trained model can categorize the events as abnormal or normal [WXG18]. If instead, the sequence of events corresponds to a particular fault, the pattern of events can help in categorizing the faults [VSP17]. When labeling is based on successful scenarios, the method can learn the sequential order to detect deviations from normal [Du+17; GYW21]. Like natural language processing, the grouping or order of events can signify an anomaly. Many of the state-of-the-art supervised methods [Hua+20; Zha+19;

Li+22b; Yu+21], use a sequence of events as input to calculate the probability that an event should occur. If the prediction differs from the log event, it is considered an anomaly. This is the same approach used in **Paper I**.

The combination learner also utilizes the order of events in **Paper II**, where abnormal delays are detected. A part of the learner then groups sequences of events based on successful scenarios and detects deviations in test data.

Instead of utilizing the order of events, it can be easier to group events that occur close to each other under normal conditions. By automatically grouping events to procedures and microservices, **Paper IV** demonstrates how anomalies can be detected faster and more accurately than they can by using state-of-the-art methods.

2.4.4 Unsupervised Anomaly Detection

The logs collected from real-time systems are, by their nature, unlabeled. The log events can be labeled by either manually analyzing each of them individually, guessing whether the log contains a fault, or using automatic labeling based on known error messages. The log events may, however, change for each software update and require more labeling and training for each version. An alternative approach is to not label the data and use unsupervised learning.

Unsupervised anomaly detection in system logs aims to detect unknown patterns and differences between individual events and log samples. There are two main approaches for unsupervised anomaly detection in logs: 1) quantitative detection models and 2) sequential detection models. The former learns the relation between events and how frequently they occur in log samples. Rare frequency patterns can then indicate an anomaly in the log. The anomalies can, for example, be sudden bursts of log events due to an attack on the system or events that only occur during failure scenarios. Models such as PCA [Xu+09], Log clustering [Lin+16], Isolation Forest [LTZ08], and Invariant Mining [Lou+10] have been frequently used to detect anomalies in this approach.

The second type of approach assumes that the order of events follows a pattern. The sequence of events can then be treated similarly to words in a sentence. Some sequences may occur less frequently than others, and by grouping sequences or parts of the sequences, it is possible to detect patterns that deviate.

Unsupervised learning can be helpful in many aspects of anomaly detection for RAN. It can help detect differences between software releases that traditional tests have not found or identify the behavior of occasional faults. The latter scenario is explored in **Paper III**, where occasional software bugs are detected using kernel and user space calls in combination with the events added by the developers.

Chapter 3

Root Cause Analysis

Anomalies are a sign of abnormal behavior. But why did they occur? What is the root cause of the anomalies? How can the root cause be identified? To answer these questions, this chapter first presents the challenges in identifying the root cause in RAN (see Section 3.1). Suitable root cause detection strategies in RAN are then discussed with my contributions in Section 3.2.

3.1 Challenges

Root cause analysis is the process that troubleshooters use to identify why an anomaly occurred. The anomaly type decides what input data (see Section 2.2) is suitable to analyze. Detecting several anomalies can narrow down the location and time when the root cause occurred. If, for example, the KPI measurements report an abnormal amount of dropped calls for the last 15 minutes, the scope can be further limited if anomalies are also detected in system logs during this period of time. The system's behavior and the anomalies are the clues the troubleshooter needs to identify the root cause. The more identified anomalies, the better the chance of pinpointing the root cause. To date, little research has been done on root cause analytics in RAN [Can+21], and RAN troubleshooters need better methods to detect the many potential root causes. Surveys of other industries also show a need for more intelligent trace analysis and visualization to improve observability and root cause analysis [Zho+21; Nie+19]. However, it is also essential to understand the challenges in RAN root cause analysis to develop new methods:

Similar symptoms. In a RAN, there can be many reasons an anomaly occurs; malfunctioning hardware may lead to performance issues, which are visible in the KPIs or system log; alarms may be raised; temperature problems; sensor failures; or an unstable connection between hardware. From a software perspective, there could be a bug in every single line of code that causes a failure in the virtualization, measurements, log collection, memory

handling, or other functionality. Anomalies also occur when humans configure a RAN, choose software versions, specify parameters, install the hardware, etc. One challenge presented by the variety of root causes is that some root causes result in similar anomalous behavior. Understanding the underlying behavior can then be the key to distinguishing between anomalies.

Many Data Sources. A 5G RAN is a large distributed system with thousands of interacting microservices. A fault may appear as an anomaly in many of these microservices, and the combined anomalies can help narrow down the root cause. Knowing where to search for anomalies and which data is important to analyze is a challenge. Understanding the complex interactions in a RAN can then help narrow down the scope and correlate the anomalies back to the source of the fault.

Lack of data. Even if there are plenty of data to analyze in a RAN it is not possible to track every action taken by the system. When a fault occurs, there is often too little information to identify the root cause. There is then a need to repeat the scenario to acquire more data. Unfortunately, some faults are occasional, and operators can be unwilling to repeat faults in their live system. The root cause can then be hidden until the fault is reported again.

Dependency on human experts. Developers are only experts on small parts of the system. Finding the root cause of the fault may require many expert teams to analyze several parts of a RAN. The anomalies found during the analysis are often explained in trouble reports that describe both the problem and the final solution that solves the problem. In the end, only those troubleshooters involved in searching for the fault or correcting it will remember and benefit from the lessons learned.

3.2 Root cause detection approaches

Finding the root cause of an anomaly is difficult for an ML model. If the root cause is a malfunctioning circuit or a software bug, the complete hardware specification or system design may be needed. Instead, the goal of most root cause detection approaches is to narrow down the time and place of the problem. This allows troubleshooters that know more about the hardware and software design to identify the root cause faster.

Depending on the available data, there are different root cause approaches to choose between. Most methods use only one source of information as input, but several methods can work together to narrow down the root cause [SB22; Sol+17]. Fig 2.1 provides a good overview of the techniques used in root cause analysis for multi-service applications. I have highlighted my contribution to the field in the extended taxonomy from [SB22] and provided a high-level

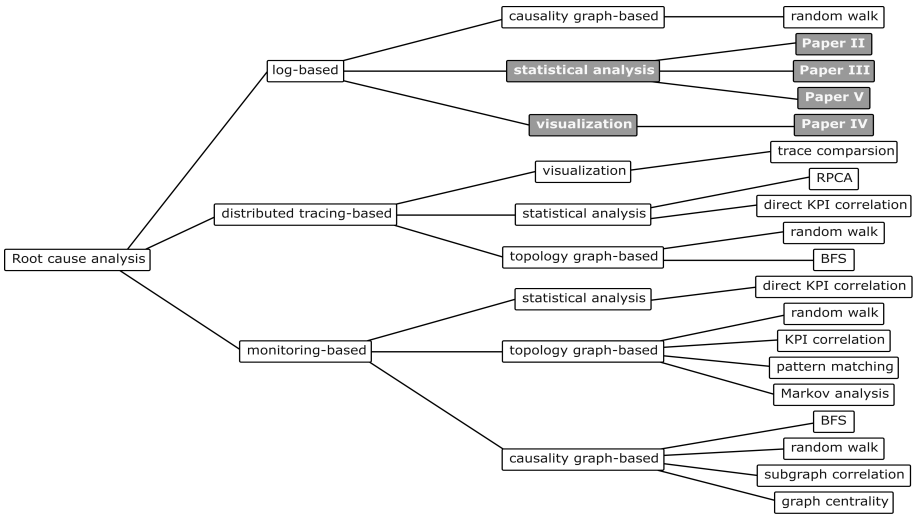

Figure 3.1: A taxonomy for classifying techniques for root cause analysis in multi-service applications. Extended version from [SB22].

description of the areas in the following three sub-sections. The methods used in each category are described in more detail in [SB22], and my contribution is summarized in Chapter 5.

3.2.1 Log-Based Root Cause Analysis Techniques

Logs contain a chain of events that describe the application's behavior. A fault may lead to several erroneous messages in the log or a sequence of events that has not been seen before. The combination of anomalies seen in the log and the chain of events may then aid in finding the root cause of the problem. This section summarizes approaches commonly used for log data before presenting my contribution to this field.

Causality graph-based analysis

The main purpose of system logs is to provide information to developers and troubleshooters. The logs can contain object information, state variables, alarms, etc., but most importantly, signal information describing the service interactions. This signaling information can be utilized to create "causality graphs" between services. Each node is then given an anomaly score that depends on the relations and where the anomalies occur. Traversing the graph from the anomaly to the highest anomaly score helps troubleshooters zero in on the root cause.

The logs can also contain information that allows troubleshooters to track objects or procedures in the system. Causality graphs can also be built on such

information, but there seems to be no research on this matter. The closest approach uses distributed tracing to create topology graphs (this is explored in more detail in Section 3.2.2).

Alarms are a kind of event that alert that a certain functionality is not working. The alarms can be visible in the system log, but they are commonly stored in a separate alarm log in a RAN. As a fault occurs in a RAN, several alarms may be triggered, and the combined alarm information can help troubleshooters identify the root cause. Canastro et al. demonstrate how a knowledge base containing alarms and their anomalies can be utilized with an alarm correlation graph to detect the root cause of alarms [Can+21]. The alarm correlations are built using a dynamic time window that groups alarms as they occur. The alarm level and type of anomaly are then used to build a graph of the alarms. As a new group of alarms appears, parts of the graph are compared with the knowledge base to find the root cause of similar graphs.

Statistical analysis

Suppose the logs contain information that enables the tracking of sequences. It is then possible to analyze the time to complete sequences and perform statistical analysis of delays between event pairs. In real-time systems like a RAN, timing is essential, and several KPIs and internal metrics measure the time to complete important services. However, these measurements seldom provide valuable information about why and where the delays occur. The statistical information from the system logs can, in such cases, help to narrow down the delays to specific areas of the software code. Paper II demonstrates how the time between sequential events can be utilized to detect timing-related anomalies in RAN. It also combines statistical, probabilistic, and clustering methods to identify between which events the delays occur.

RAN is built to handle many kinds of service, but the complex architecture sometimes does not scale well with an increased load. Paper V introduces a procedure ID to the system logs that help identify which parts of the services cannot handle an increased load. The procedure ID enables the tracking of sequences of events and measures the time to complete each part in all microservices. The interaction through the entire system can then be analyzed by combining the signaling and procedure information from system logs. It is then possible to statistically compare the time spent by all parts and narrow down those that suffer the most from an increasing load.

Developers seldom add events in the system log that provides information about the system's design. To find software bug-related root causes, troubleshooters often need to analyze the system log and the software. Paper III presents a novel use of kernel and function calls combined with existing system log events. The pattern they create is then used to find statistical deviations in time spent for each pattern and to identify patterns that seldom occur. The troubleshooters can then find the root cause of the software bugs by analyzing the largest deviations.

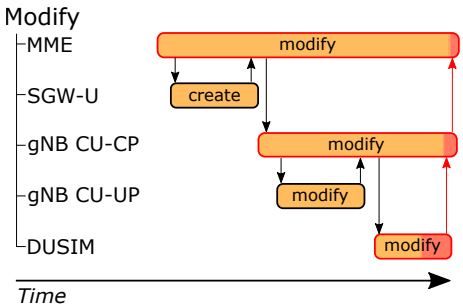

Figure 3.2: Visualization of an anomaly in a systemized procedure.

Visualization

The signaling information mentioned in Section 3.2.1 has been used to create sequence diagrams for end users. By visualizing the behavior, troubleshooters can better understand the service interaction. The sequence diagrams can then be compared as anomalies occur, and troubleshooters can investigate the flow of signals to back-track the root cause. Paper IV introduces a procedure concept that allows end users to track systemized procedures in the logs. A novel way of visualizing the anomalies is then enabled by combing the procedure information with anomalies detected by the ML methods (see Fig 3.2).

Paper IV further demonstrates how the procedure information and visualization can help troubleshooters identify anomalies faster and better understand the ongoing procedures.

3.2.2 Distributed Tracing-based Root Cause Analysis Techniques

Distributed tracing has been implemented in many industrial microservice applications [JLL22], but few have applied ML or statistics to aid in the root cause analysis [Li+22a]. Active projects, such as Zipkin [Zip20] or Jaeger [Jae20], provide a framework to implement and monitor distributed tracing. There are also tracing specifications such as OpenTracing [Ope20b] and OpenTelemetry [Ope20a], that allow developers to instrument their solution.

Even if there is limited use of distributed tracing in RAN, it is interesting to learn how tracing information can help in root cause analysis through visualization, statistics, and graph-based approaches. Distributed tracing can also help improve observability, which is further explored in Section 4.2.3.

Visualization-based analysis

The traces provide a visual overview of the microservice interaction and the time spent on each request. A trace comparison methodology can help end

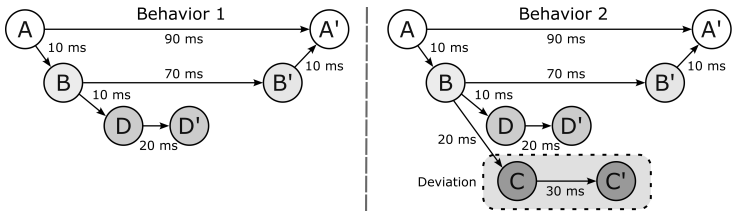

Figure 3.3: An example of how deviations can be visualized in a life cycle diagram for two traces.

users understand the differences between normal and abnormal behaviors (see Fig 3.3).

The deviations can then be highlighted in, for example, a life cycle diagram to help end users quickly locate the problem area.

Statistical analysis

The response time for each service can be collected through the tracing information. Anomalies can be detected if the response time is above a certain threshold, which can either be set manually or statistically. All parts of the interactions contributing to the long response time can then be grouped and analyzed. The response time for each interaction in the group is then utilized to determine the most significant deviation and possible root cause.

Topology graph-based analysis

The service interaction can also be utilized to generate a topology diagram. By comparing the response time (RT), error count (EC), and queries per second (QPS) for abnormal and normal scenarios, the diagram can help end users traverse a given flow of interactions. By assigning an error value for each node in the topology tree, it is possible to identify the most probable cause in the service chain.

3.2.3 Monitoring-Based Root Cause Analysis Techniques

This section explores how KPI values can be utilized to narrow down the root cause of an anomaly. In a RAN, there are hundreds of useful KPIs that are collected periodically. Some of them are correlated to each other, and as a fault occurs, several KPIs may be affected simultaneously. The size of the deviation can also help localize the root cause. The main research demonstrates how failing microservices can be identified, but the KPIs in a RAN can also provide additional information on the part of the service that fails.

Statistical analysis

Monitoring KPIs can detect deviations through statistical or ML-based methods. This can be of special interest when anomalies also occur in system logs, traces, or alarms. Comparing KPI values from periods with and without anomalies can help in identifying correlations between KPIs and anomalies. The correlations found can then aid in narrowing down the root cause and can also be useful when comparing the anomalies with future faults.

Another approach is to proactively cause or simulate problems instead of waiting for them to occur. Learning how the KPIs are affected due to many known reasons facilitates the creation of a knowledge base of the KPI-anomaly dependencies. As deviations are detected in KPIs, the knowledge base can help identify components likely to cause problems.

Another interesting approach in RAN has also shown that KPIs can be used to identify sleeping cells [FBA16]. The KPIs for each user equipment can then reveal which neighboring cells have deviations in the number of served UEs. By utilizing the geographical position of the user equipment, it is then possible to identify which cells have problems.

Topology graph-based analysis

KPIs can also measure the latency, response time, packets sent, or throughput between the interacting microservices. These metrics describe the interaction between services and can be used to create a topology similar to the one described in Section 3.2.2. A similar root cause technique can then be applied to group anomalies that occur and narrow down the root cause to a sub-graph of the topology. Additional help can be given to troubleshooters if the identified sub-graphs are stored together with information on where the real root cause is found. As new faults occur, the sub-graph can be compared with the root causes that have already been identified to zero in on failing services.

Causality graph-based analysis

Many of the KPI values in RAN are dependent on each other. If, for example, the number of successful calls increases, the number of released calls likely increases. A pairwise examination in time series can determine the dependency between KPIs. These dependencies can then form a causality graph that provides valuable information in root cause analysis. As anomalies occur in one KPI, the graph can be traversed to identify how KPIs are affected in normal cases. If the KPI-service dependency also is known, it can narrow down the area of the root cause.

KPIs can also be compared between services to create causality graphs. When determining the root cause of a performance anomaly, it can be useful to know the dependencies between services in terms of response time, error count, CPU, or queries per second. As the performance is degraded in one service,

the causality graph can be used to track which service may have caused the problem.

Chapter 4

Observability

To determine the root cause of an anomaly, it is necessary to understand the system's behavior. Observability is about understanding the system's internal states based on external outputs. This chapter discusses observability challenges in RAN and how combined ML and statistical methods can improve observability.

4.1 Challenges

The quality of the data decides whether the system's behavior can be understood or not. The data used in observability are often mentioned as the three pillars of observability:

- **Logs** - Events that reflect the system's behavior.
- **Metrics** - Periodical metrics that present the performance and resource utilization of the system.
- **Traces** - Records how requests are sent through the system.

These are the same categories of data analyzed to detect anomalies and root causes in Chapter 2 and Chapter 3. These areas are also closely related to each other. By monitoring a system, it is possible to detect anomalies; the root cause of the anomaly is then found by observing the system. However, there are several challenges associated with observing behavior in a RAN:

Choosing what to store. Not everything is observable in large real-time systems. Recording the change of all variables, states, and signals sent would produce thousands of events every second. Recording would then slow down the system's performance. The challenge as a developer is to choose what to record. If there is too little data, parts of the behavior might be missed, and too much data could cause confusion or slow down

the system. A larger system also requires more structure in the data than a small system. Without a structure that enables the sorting and filtering of interesting parts, it is difficult to understand the system's behavior.

Human Capacity. Another problem is related to human and CPU capacity. The approach RAN troubleshooters use has remained relatively unchanged for the last 20 years. As functionality fails, troubleshooters manually analyze metrics and logs to find the root cause of the problem. But RAN has become more complex, and CPU capacity has increased much faster than the capacity of the human brain. As more events per second occur, troubleshooters also have more data to analyze. The data is also scattered in more units today than ever before. Both of these aspects make it even more important to enable the sorting, filtering, and explaining of data.

Quantify the value of Observability. Even if some developers and troubleshooters realize that observability needs to be improved, management must also be aware of this need. A large industry survey shows that this is a large problem across many fields [Nie+19]. The survey also found a strong need for an organizational concept, including strategy, roles, and responsibilities. The conclusion of the survey, which is in line with our findings it that there is a need to support the development of systematic observability and monitoring for distributed systems.

RAN transformation. A new challenge in 5G RAN is the ongoing transformation of the 4G monolithic architecture into the cloud-native microservice architecture. The purpose of the new architecture is to simplify distribution, redundancy, update speed, scalability, etc. But the transformation also has a downside. Re-engineering the 4G solution into 5G is difficult, and if not properly systemized, it can result in latency problems and poor performance. Understanding the complex microservice interaction can be crucial for developers that need to troubleshoot the system.

Microservice diversity. Thousands of developers work together to develop the software used in a RAN. Even if several parts of RAN exchange information, it is up to the individual developers to decide which information to store for debugging purposes. for that reason tracking the interaction and understanding the behavior can be a challenging process.

4.2 Observability approaches

Even if researchers have started exploring how ML can help troubleshoot and improve observability, few industries have applied this research [Li+22a]. The following three sections explore how ML can improve observability using logs, metrics, and traces.

4.2.1 Observability Through Logs

Logs are traditionally written and read by humans, no matter what system they originate from. In the era of ML, it has become popular to search through large logs to find anomalies and present them to troubleshooters. But this is more about monitoring than providing an insight into the system's behavior. New techniques are needed to aid troubleshooters to see through the data and discover what matters. It can be more important to know how the events are related to each other, which events are essential, what functionality or scenarios are ongoing, or how the events deviate from the normal.

Previous research has explored different mining techniques to extract relations between services from system logs [Gor+22]. Information about signaling, for example, can visualize the interaction between services and provide a better understanding of the architecture. One interesting mining approach also combines static code analysis with the occurrence of events [IAC22]. The combination enables the creation of a business process heat map that can explain where performance issues occur and visualize where there is a high load in the software.

The root cause causality graph-based approaches discussed in Section 3.2.1 provide another insight into system behavior. The graphs reveal dependencies that may be unknown to the end user and help troubleshooters find the root cause of anomalies.

The behavior can also be analyzed from a time perspective as discussed in Section 3.2.1. The end user can then get valuable insight into where delays occur in RAN.

Our largest contribution to observability concerns the procedural identity information discussed in Section 3.2.1. The software instrumentation guidelines proposed in Paper IV introduce a concept called procedural information, which connects all events in the system log to systemized procedures. The combination of procedural information and equipment identifiers, cell information, and signals sent enables tracking of sequences of events through RAN. Each procedure is also given a name that enables the grouping of events and improves observability (see Fig 4.1).

The figure presents a top-level overview of which procedures occur for each user equipment. The microservice interaction can be explored for each procedure, and the events are grouped for each microservice and procedure. Anomalies found can then be highlighted at all levels, and the behavior can be compared to better understand the deviations.

4.2.2 Observability Through Metrics

Metrics are monitored to reveal the system's performance during a certain period of time. Often, they are presented in a time series, which enables analysis over time. It is common to present average, min, or max values together with manually defined thresholds for the important metrics so the troubleshooter

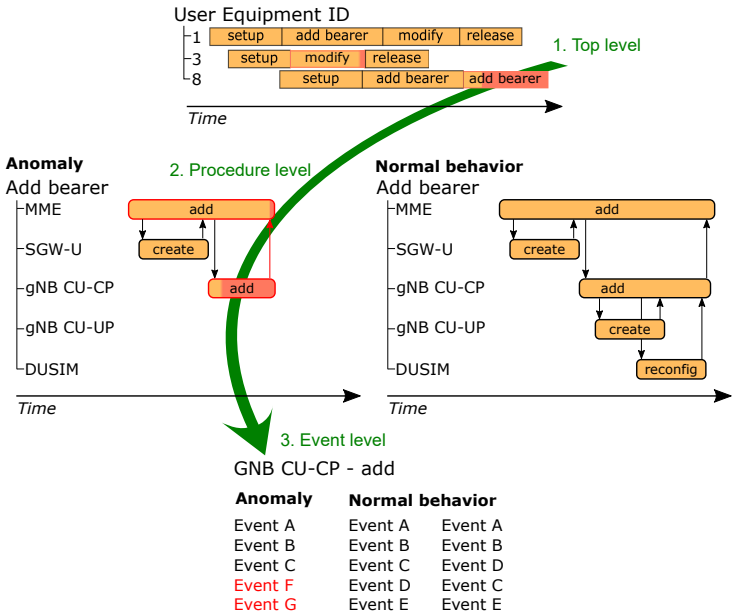

Figure 4.1: Procedure information provides valuable insights from an overview perspective down to events in the log. (Extended version from Paper IV).

can understand if the performance is normal. Statistical or ML methods can analyze the time series data to provide trend lines for future behavior or show when anomalies occur.

The metrics can also reveal hidden dependencies, which facilitates a better system understanding. The causality graphs discussed in Section 3.2.3 can then help to identify the metrics that may affect each other or find relations that were previously unknown.

Similarly, as discussed in Section 3.2.3, it is also possible to explore how the anomalies may affect the KPI data. The insights help identify the parts affected by anomalies in traces or logs (see Fig 4.2).

4.2.3 Observability Through Traces

Distributed tracing is a method of observing requests as they propagate through distributed cloud environments. Many industries have improved observability by implementing a tracing mechanism to track service requests [Li+22a] in their microservice system. The tracing information can provide valuable information about the system through a lifecycle or flow diagrams as shown in Fig 4.3.

The diagrams help to create an understanding of the microservice interaction and where most time is spent on a service request. Statistical or ML methods

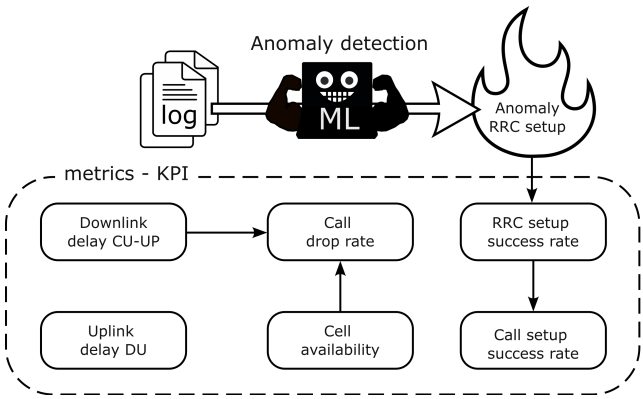

Figure 4.2: Causality graph visualizing the dependencies between KPIs and anomalies detected.

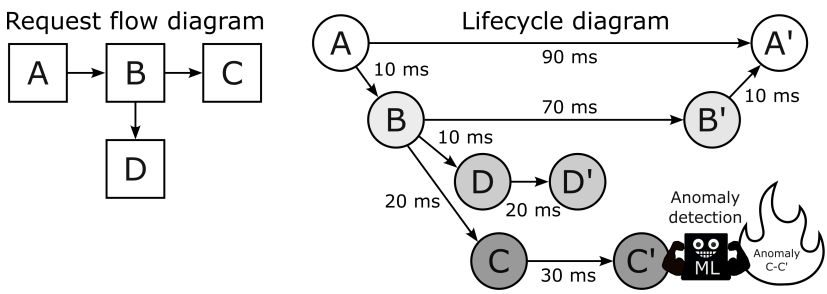

Figure 4.3: Service request flow diagram and lifecycle diagram based on tracing information. (Extended from Paper V)

can also provide additional information for the diagrams and highlight where the behavior deviates from normal.

4.2.4 Combined Approaches

The individual approaches above can help improve the system's observability, but they can also be combined to improve the understanding even more. An example of how the anomalies detected from one source of information can be combined with logs or traces was explored in Section 4.2.2. A more extensive approach is described by Tzanettis et al., who combine data from all pillars of observability to improve the system's observability [Tza+22]. The authors also suggest that their data fusion enables new ML methods to analyze, visualize, and orchestrate a microservice system. Based on my 20 years in the telecom industry I also believe that humans need assistance in these areas, and this is discussed further in Chapter 6.

Chapter 5

Summary of Contributions

This chapter describes how the five papers presented in this thesis contribute to addressing the research objectives. The main objective of all papers included in the thesis is to find new methods to help RAN troubleshooters detect anomalies and identify their root causes. The papers also highlight how observability can be improved through the structured instrumentation of software and can automatically prevent performance issues and reduce resource utilization.

Throughout the papers, the system's behavior is learned from events in RAN system logs. The logs are also the main source of information for developers that troubleshoot RAN. While troubleshooters can spend days manually analyzing the millions of events from a RAN, ML can do this in seconds. Paper I presents how an ensemble of Long Short-Term Memory (LSTM) modules can detect the majority of anomalies by learning the sequence of events and predicting the preceding ones. Paper II focuses on the temporal behavior between the events where tiny, unexpected delays can be detected in the control plane in RAN. Three techniques are combined to provide a detailed view of where unexpected delays occur in a RAN. Paper III analyses the behavior more in detail to identify the root cause of the anomalies. Two new methods then learn the relation between systemized events and kernel/function calls. The relations form patterns that can help find occasional software bugs and improve the understanding of the complex microservice interaction in a RAN. Paper IV proposes a novel software instrumentation guideline to further improve observability and anomaly detection. The paper highlights the necessity to maintain a structure in the system log and proposes a new way to track the sequences of events through a RAN. Observability and anomaly detection can in such a way be enhanced by adding systemized procedure information to each event in the log. In Paper V, the procedural information is combined with temporal information to identify performance bottlenecks. It also demonstrates how bottleneck identification can be utilized to dynamically lower energy consumption in a RAN while still satisfying latency requirements.

In all papers, I acted as the main author, Erik Elmroth as the main supervisor, Monowar Bhuyan as co-supervisor, and Johan Forsman as the industry supervisor. Each paper is summarily described in the following sections.

5.1 Paper I

T. Sundqvist, M. Bhuyan, J. Forsman, and E. Elmroth. Boosted Ensemble Learning for Anomaly Detection in 5G RAN. *IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI 2020)*, pp. 15-30, 2020.

RAN troubleshooters examine the system's behavior by manually examining system logs. The anomalies can be found by learning the order of events and identifying any deviations from the normal flow. Troubleshooters would save considerable time if they could identify where anomalies are in the log within a matter of seconds.

5.1.1 Paper Contributions

Paper I addresses **RO1** by proposing a method that detects anomalies in the sequential order of system log events. The intention was to create a method that could be used early in the RAN development chain where continuous integration loops verify every new delivery. As test cases are completed in a RAN, the events in system logs follow a pattern similar to how words and sentences are constructed. Inspired by ideas from natural language processing, a novel Adaboost ensemble of Long Short-Term Memory (LSTM) models is created to learn the order of events in system logs. The method learns from successful scenarios and can help troubleshooters identify anomalies as scenarios fail in the continuous integration environment.

The effectiveness of the LSTM ensemble is verified on data from an advanced 5G testbed developed by Tietoevry[1] and a popular HDFS data set. It also demonstrated how a human-in-the-loop can help improve the method if false positives are found.

5.2 Paper II

T. Sundqvist, M. Bhuyan, and E. Elmroth. Uncovering Latency Anomalies in 5G RAN - A Combination Learner Approach. *IEEE 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 621-629, 2022.

[1] https://www.tietoevry.com

Several services in RAN are time-critical, and it is essential to know where and why abnormal delays occur. The existing KPIs only provide an end-to-end latency for some services, and it is not enough to provide a detailed overview of where time is spent. There is also little knowledge of how new software updates timely affect existing functionality.

5.2.1 Paper Contributions

Paper II also addresses **RO1** but analyzes the system logs from a time perspective. Three new methods are created to analyze the time between events in the system log from a statistical, probabilistic, and unsupervised ML approach. As the sequences of events form special patterns, the methods can learn the statistical difference for sequential events and detect delays between different test runs. The three methods were combined into a learner that can reveal more delays than the individual methods. The combined learner was evaluated in a 5G testbed in which delays ranging from 1 ms to 1000 ms were injected. The learner could detect almost all anomalies, and thanks to the detailed latency overview, It also discovered previously unknown problems in the control plane scenarios. The detailed latency overview also partly addresses **RO2**, as it helps developers narrow down the delays' root cause. The combination learner can also be useful in continuous integration loops, where abnormal delays can be detected in new software releases.

5.3 Paper III

T. Sundqvist, M. Bhuyan, and E. Elmroth. Unsupervised Root-cause Identification of Software Bugs in 5G RAN. *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC 2022)*, IEEE, pp. 624-630, 2022.

Most problems in RAN are related to software bugs introduced during the software development or correction of earlier problems. Some of these faults are occasional and can be difficult to reproduce. The faults can occur for many reasons, and troubleshooters often have to analyze the software code to determine possible scenarios that may have caused the problem. The wide range of scenarios can be challenging to foresee, which is also why occasional bugs occur. Troubleshooters could better understand the range of potential scenarios and more quickly identify software bugs if a method is used that can learn the software function interaction and identify behavior deviations.

5.3.1 Paper Contributions

Paper III, addresses **RO3** by proposing two new methods that learn the call chains between events instrumented by the developers and function/kernel calls. The new methods analyze the time spent in each call chain and can detect

deviations in both time and call chain interaction. Call chains that seldom occur are presented in a top-candidate list that the end users can analyze further to identify the software bugs. The methods were also constructed to handle the large amount of data a RAN produces. Collecting all kernel and function calls can generate millions of events every minute in a RAN, and there is sometimes a need to analyze hours of data. The methods were evaluated in a 5G testbed by introducing different software bugs and triggering these to occur randomly. The results show that the methods could identify the software bugs much faster and more accurately than previous research.

5.4 Paper IV

T. Sundqvist, M. Bhuyan, and E. Elmroth. Robust Procedural Learning for Anomaly Detection and Observability in 5G RAN. *IEEE Transactions on Network and Service Management*, IEEE, 2023 (submitted for publication).

A problem in RANs and other large systems is the diverse software implementation of the many different development teams. Each team is responsible for a small part of the software they instrument to fulfill their troubleshooting needs. This diversity makes it difficult to grasp the big picture when several services interact. Tracking and understanding behavior throughout the whole system is crucial when troubleshooting, and a systemized and generic approach to software instrumentation would improve observability. The instrumentation diversity and unstructured data are also problematic for statistical and ML methods that need to parse and extract features from the logs. The poor data structure results in bad parsing accuracy and difficulty tracking the chain of events, ultimately leading to poor performance in the methods that learn the system's behavior.

5.4.1 Paper Contributions

Paper IV introduces a new software instrumentation guideline and addresses **RO3**. The paper highlights the importance of structured data and introduces the concept of procedural information. As features are developed in a RAN, a system document often describes the interaction between the many objects. These interactions can be tracked by assigning a unique procedure identifier for each systemized request. The identifier is stored together with each event and enables a new kind of procedure grouping. By storing the feature's name with the identifier, it becomes easier to track the behavior and map it to the systemization.

The benefit of the guideline is evaluated from two perspectives. The first compares the system understanding of developers that troubleshoot the system. This also addresses **RO2**, as the new guidelines help developers understand the system better and locate the root cause of anomalies faster. The second addresses

RO1 by implementing a new method, which uses procedural information to identify anomalies. The new procedural learning method outperforms previous state-of-the-art methods in both speed and accuracy. Hopefully, these findings can help developers and managers recognize the importance of structured and aligned data.

5.5 Paper V

T. Sundqvist, M. Bhuyan, and E. Elmroth. Bottleneck Identification and Failure Prevention with Procedural Learning in 5G RAN. *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, accepted for publication, 2023.

Many may not know that the new 5G telecom network is reusing software components from the existing 4G network. The challenge in the 5G transformation is building a new microservice architecture based on the previous monolithic architecture. The new architecture uses the microservice advantage to allow more flexible scaling, improve capacity, and accelerate the time-to-market. A challenge with this transformation is deciding how to split the old functionality while simultaneously increasing capacity. As the new architecture takes shape, there is also a need to detect performance bottlenecks for the implemented functionality.

5.5.1 Paper Contributions

Paper V addresses **RO2** by combining the latency analysis from Paper II and procedural learning from Paper IV. The result is a method called LogGenie that can track procedures through a RAN and measure the latency at procedure, microservice, and event pair levels. LogGenie detects performance bottlenecks by comparing the latency as the load increases in the system. The largest latency increase for microservices and event pairs is then presented in a top-candidate list. End users can also view a list of the procedures that suffer the most from the load increase.

The bottlenecks identified by LogGenie are then utilized to create a mechanism to automatically scale the system and prevent it from overloading. LogGenie decides where to change the capacity and triggers appropriate scaling actions as the system load varies. The paper further demonstrates how this mechanism can help lower resource utilization and still keep the latency requirements of the system.

Chapter 6

Future Research

Some of the methods in this thesis have already been implemented in cloud RAN production, and many interesting paths remain unexplored. This chapter outlines areas that may benefit from this thesis research in the future.

6.1 Evaluation in Production Environment

In parallel with my research, I am collaborating with Ericsson[1] to explore the benefits of the research methods presented in this thesis. At the time of writing, an integrated solution using the method from Paper I now detects anomalies in cloud RAN. The results are promising; the LSTM ensemble can learn the sequential patterns from continuous integration tests and identify abnormal events. An important lesson learned is that it is important to limit the scope of the methods such that each model learns the behavior for only one microservice and scenario. Parts from Paper IV have also been integrated, where the procedural information is added to the log events. The early results show that anomaly detection is enhanced, but further evaluation with more scenarios and microservices is needed. The potential to also utilized the time aspect in the procedure learning will be further explored in a master's thesis at Umeå University[2]. The goal is to reinforce anomaly detection and the system's observability.

An important finding is also that the end users must understand why the ML methods think there is an anomaly, otherwise, it is difficult to gain trust in the methods. This challenge is addressed by providing a solution where the normal and abnormal behavior is visualized in a sequence diagram. The abnormal behavior is also further explained in a complimentary report.

[1] https://www.ericsson.com/en
[2] https://www.umu.se/en/

6.2 Observability in RAN

Distributed tracing and system logs can be utilized to find the interactions between the microservices in a RAN. This interaction has been visualized in a text format in terminal windows and as UML sequence diagrams. RAN troubleshooters could benefit from displaying the anomalies in such diagrams. The question is how to present anomalies to the end users. The normal and abnormal behavior could be presented side by side or in the same diagram, but one must also consider the large variation in normal behavior.

Visualizing the affected objects, such as cells, radio bearers, tunnels, user equipment, and other RAN-specific objects, would also be advantageous. The structure allows the separation of concurrent actions and improves the system's observability. The procedural information in Paper IV would also be useful if applied to help isolate ongoing procedures.

6.3 Learning from Live Nodes

Most methods described in this thesis aim to help troubleshooters as early as possible in the development cycle, but the findings can also help troubleshoot logs from live nodes. A recent proof of concept explores how the methods from Paper I can learn the behavior from test cases in continuous integration loops and analyze logs from live nodes. The early findings demonstrate that the methods can assist in finding anomalies in live node logs, but the situation is more complex in the live network. Many alternative scenarios occur, and the timing of events is more random.

It would be interesting to evaluate whether ML can be utilized to learn the behavior from live logs instead of in the continuous integration loops. This would require a more structured system log, where the ML models can automatically learn the many alternative scenarios. The procedural learning from Paper IV may be a path in the right direction.

Collecting data from many nodes and training models based on the node type, software version, and configuration would also be beneficial. The ML methods could then identify the nodes that behave differently and find nodes with a faulty configuration or nodes that suffer from hardware failure or software bugs.

6.4 RAN Performance Scaling

RANs experience large daily and weekly fluctuations in traffic intensity. The nodes and antennas are geographically bounded, but citizens travel and change their mobile usage depending on, for example, whether they are asleep, at work, at school, or engaged in leisure. Future nodes must be capable of handling the large fluctuations in traffic while keeping energy consumption low. The many hardware and software configurations in a RAN can make it difficult

to manually specify rules for how the system should scale. If ML models can handle and optimize the scaling, this would be a tremendous benefit. Paper V demonstrated one way to scale the system depending on where the ML methods detected performance anomalies. Still, much work is left to be done before it can be applied in a RAN. One challenge is to ensure that all requirements are fulfilled while the system scales, another is to find a robust solution that works for the many configurations of a RAN.

ML models can also help balance the system's load to increase scaling efficiency in RAN. A current collaboration between Tietoevry and Karlstad University[3] explores how to use reinforcement learning for load optimization in a 5G test bed. Perhaps it can be combined with the research from this thesis to also consider abnormal behavior in the load balancer.

6.5 Finding Duplicate Faults

Occasional faults are often difficult to troubleshoot in a RAN. They can appear in live nodes but also in the test suits that are run in the continuous integration loops. Some of these faults result in erroneous messages or reject signals in the system log and can provide a clue to the problem. These printouts are commonly used to decide whether the fault has been seen before or not. Troubleshooters can also set up regular expressions to automatically detect when test cases contain the same error printouts.

Another interesting approach would be to utilize the anomalies detected by the ML methods described in this thesis. The detected anomalies form a unique pattern for a certain fault, and similar patterns can be grouped. The fault grouping can then be automatized and will not need to rely on the manual regular expressions added by troubleshooters. There is also less chance of missing similar faults and it becomes possible to detect problems earlier in the development process. To explore this more in practice, a joint collaboration between Umeå University, Tietoevry, and Ericsson is now evaluating how ML methods can group the faults.

6.6 Final Words

The research goal of this thesis is to aid RAN developers in understanding system behavior and detecting anomalies in system logs. I have already seen the benefits of the methods from Paper I and Paper IV in cloud RAN, and I am eager to integrate solutions from the remaining of my research.

It is also worth noting that the accuracy of the methods described in this thesis is highly dependent on the quality of the system log data but the methods should also work for other systems as well. As long as it is possible to track

[3] https://www.kau.se/en

the sequence of events in the system log, and the information is relevant for troubleshooting, the system is likely to benefit from this work.

Bibliography

[3gp22] 3gpp.org. *Finding AI in 3GPP*. 2022. URL: https://www.3gpp.org/technologies/finding-ai-in-3gpp.

[ana22] IoT analytics. *IoT 2022: Connected Devices Growing 18% to 14.4 Billion Globally*. 2022. URL: https://www.iotforall.com/state-of-iot-2022.

[Can+21] Dinis Canastro, Ricardo Rocha, Mário Antunes, Diogo Gomes, and Rui L Aguiar. "Root Cause Analysis in 5G/6G Networks". In: *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2021, pp. 217–224.

[CCP12] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. "Event logs for the analysis of software failures: A rule-based approach". In: *IEEE Transactions on Software Engineering* 39.6 (2012), pp. 806–821.

[Co19] Huawei Technologies Co. *Enable Autonomous Driving Network*. 2019. URL: https://carrier.huawei.com/~/media/cnbgv2/download/adn/huawei-naie-white-paper.pdf.

[Dam07] Lars-Ola Damm. "Early and cost-effective software fault detection: measurement and implementation in an industrial setting". PhD thesis. Blekinge Institute of Technology, 2007.

[Du+17] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning". In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, pp. 1285–1298.

[Eri20] Ericsson. *5 key facts about 5G radio access networks*. 2020. URL: https://www.ericsson.com/en/public-policy-and-government-affairs/5-key-facts-about-5g-radio-access-networks.

[FBA16] Sergio Fortes, Raquel Barco, and Alejandro Aguilar-Garcia. "Location-based distributed sleeping cell detection and root cause analysis for 5G ultra-dense networks". In: *EURASIP Journal on Wireless Communications and Networking* 2016 (2016), pp. 1–18.

[Gor+22] Mia E Gortney, Patrick E Harris, Tomas Cerny, Abdullah Al Maruf, Miroslav Bures, Davide Taibi, and Pavel Tisnovsky. "Visualizing Microservice Architecture in the Dynamic Perspective: A Systematic Mapping Study". In: *IEEE Access* (2022).

[GYW21] Haixuan Guo, Shuhan Yuan, and Xintao Wu. "Logbert: Log anomaly detection via bert". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[He+17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. "Drain: An online log parsing approach with fixed depth tree". In: *2017 IEEE international conference on web services (ICWS)*. IEEE. 2017, pp. 33–40.

[He+21] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. "A survey on automated log analysis for reliability engineering". In: *ACM computing surveys (CSUR)* 54.6 (2021), pp. 1–37.

[Hu+21] Shuyan Hu, Xiaojing Chen, Wei Ni, Ekram Hossain, and Xin Wang. "Distributed machine learning for wireless communication networks: Techniques, architectures, and applications". In: *IEEE Communications Surveys & Tutorials* 23.3 (2021), pp. 1458–1493.

[Hua+20] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. "Hitanomaly: Hierarchical transformers for anomaly detection in system log". In: *IEEE transactions on network and service management* 17.4 (2020), pp. 2064–2076.

[IAC22] Md Rofiqul Islam, Abdullah Al Maruf, and Tomas Cerny. "Code Smell Prioritization with Business Process Mining and Static Code Analysis: A Case Study". In: *Electronics* 11.12 (2022), p. 1880.

[Jae20] Jaegertracing.Io. *Jaegertracing*. 2020. URL: `https://www.jaegertracing.io/`.

[Jit+21] Manocha Jitendra, Terrill Stephen, Mattsson Ulf, Filipovic Zlatko, Westerberg Erik, and Kopplin Dirk. *Accelerating the adoption of AI in programmable 5G networks*. 2021. URL: `https://www.ericsson.com/4a3998/assets/local/reports-papers/white-papers/08172020-accelerating-the-adoption-of-ai-in-programmable-5g-networks-whitepaper.pdf`.

[JLL22] Andrea Janes, Xiaozhou Li, and Valentina Lenarduzzi. "Open Tracing Tools: Overview and Critical Comparison". In: *arXiv preprint arXiv:2207.06875* (2022).

[Lan+22] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. "Deep Learning for Anomaly Detection in Log Data: A Survey". In: *arXiv preprint arXiv:2207.03820* (2022).

[Li+22a] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. "Enjoy your observability: an industrial survey of microservice tracing and analysis". In: *Empirical Software Engineering* 27 (2022), pp. 1–28.

[Li+22b] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. "SwissLog: Robust Anomaly Detection and Localization for Interleaved Unstructured Logs". In: *IEEE Transactions on Dependable and Secure Computing* (2022).

[Lin+16] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. "Log clustering based problem identification for online service systems". In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE. 2016, pp. 102–111.

[Lou+10] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. "Mining invariants from console logs for system problem detection". In: *2010 USENIX Annual Technical Conference (USENIX ATC 10)*. 2010.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[MHH18] Qian Mao, Fei Hu, and Qi Hao. "Deep learning for intelligent wireless networks: A comprehensive survey". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 2595–2621.

[Nie+19] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. "On observability and monitoring of distributed systems–an industry interview study". In: *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*. Springer. 2019, pp. 36–52.

[Ope20a] Opentelemetry.Io. *Opentelemetry*. 2020. URL: `https://opentelemetry.io/`.

[Ope20b] Opentracing.Io. *Opentracing*. 2020. URL: `https://opentracing.io/`.

[OS07] Adam Oliner and Jon Stearley. "What supercomputers say: A study of five system logs". In: *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE. 2007, pp. 575–584.

[Pef+07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. "A design science research methodology for information systems research". In: *Journal of management information systems* 24.3 (2007), pp. 45–77.

[SB22] Jacopo Soldani and Antonio Brogi. “Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey”. In: *ACM Computing Surveys (CSUR)* 55.3 (2022), pp. 1–39.

[Sol+17] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. “Survey on models and techniques for root-cause analysis”. In: *arXiv preprint arXiv:1701.08546* (2017).

[Sun+19] Yaohua Sun, Mugen Peng, Yangcheng Zhou, Yuzhe Huang, and Shiwen Mao. “Application of machine learning in wireless networks: Key techniques and open issues”. In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3072–3108.

[Tza+22] Ioannis Tzanettis, Christina-Maria Androna, Anastasios Zafeiropoulos, Eleni Fotopoulou, and Symeon Papavassiliou. “Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications”. In: *Sensors* 22.5 (2022), p. 2061.

[VSP17] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. “Long short-term memory based operation log anomaly detection”. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2017, pp. 236–242.

[WXG18] Mengying Wang, Lele Xu, and Lili Guo. “Anomaly detection of system logs based on natural language processing and deep learning”. In: *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*. IEEE. 2018, pp. 140–144.

[Xu+09] Wei Xu, Ling Huang, Armando Fox, David A. Patterson, and Michael Jordan. *Large-Scale System Problems Detection by Mining Console Logs*. Tech. rep. UCB/EECS-2009-103. EECS Department, University of California, Berkeley, July 2009. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-103.html`.

[YKD20] Rakesh Bahadur Yadav, P Santosh Kumar, and Sunita Vikrant Dhavale. “A survey on log anomaly detection using deep learning”. In: *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE. 2020, pp. 1215–1220.

[Yu+21] Dongqing Yu, Xiaowei Hou, Ce Li, Qiujian Lv, Yan Wang, and Ning Li. “Anomaly Detection in Unstructured Logs Using Attention-based Bi-LSTM Network”. In: *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*. IEEE. 2021, pp. 403–407.

[Zha+19] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. "Robust log-based anomaly detection on unstable log data". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 807–817.

[Zho+21] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study". In: *IEEE Transactions on Software Engineering* 47.2 (2021), pp. 243–260.

[Zhu+18] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. "Tools and Benchmarks for Automated Log Parsing". In: *CoRR* abs/1811.03509 (2018). arXiv: `1811.03509`.

[Zip20] Zipkin.Io. *Zipkin*. 2020. URL: `https://zipkin.io/`.

[ZJM22] Xiaoqing Zhao, Zhongyuan Jiang, and Jianfeng Ma. "A Survey of Deep Anomaly Detection for System Logs". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–8.

[ZPH19] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. "Deep learning in mobile and wireless networking: A survey". In: *IEEE Communications surveys & tutorials* 21.3 (2019), pp. 2224–2287.