Technische Universität Berlin

Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

DOCUMENT BUILD DATE: March 25, 2025
DOCUMENT STATUS: Final

Multi-Agent Root-Cause Analysis Framework for Private 5G Networks

Bachelor's Thesis
am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der Telekommunikation (AOT)
Prof. Dr.-Ing. habil. Sahin Albayrak
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

vorgelegt von
Petar Iliev

Gutachter:	Prof. Dr.-Ing. habil. Sahin Albayrak
	Prof. Dr. Odej Kao

Petar Iliev
Matrikelnummer: 398743

Erklärung der Urheberschaft

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Ort, Datum Berlin, 25.03.2025 Unterschrift [illegible]

Declaration of authorship

I hereby confirm that I prepared this thesis independently and by exclusive reliance on the literature or tools indicated herein.

Ort, Datum Berlin, 25.03.2025 Unterschrift [illegible]

Abstract

As 5G networks become more complex due to increased diversity in use cases, the challenges in monitoring and managing network performance grow. The software required to do that usually demands specialized expertise, making it difficult to detect and address issues such as radio link failure or distributed denial-of-service (DDoS) attacks on time. This complexity highlights the need for solutions that enable an efficient and scalable analysis of network data.
This thesis proposes a multi-agent framework that utilizes NWDAF EventSubscription service to enable intelligent event analysis and communication across a 5G network. The framework dynamically collects, correlates, and processes event data from different network sectors, allowing for efficient anomaly detection and root cause analysis. Specialized agents subscribe to NWDAF's event notifications, receiving relevant data for relevant monitored events based on each network layer. A central aggregation agent collects and integrates these event reports, formats the information, and applies predefined analysis rules to extract actionable insights.
The framework offers a flexible, 5G-agnostic approach, allowing operators to create customized agents suited to their unique needs. This solution enables network operators and non-specialists to monitor the network's status more intuitively and with increased clarity. It reduces the time needed to notice an issue and take a corrective approach.

Abstrakt

Da 5G-Netze aufgrund der zunehmenden Vielfalt der Anwendungsfälle immer komplexer werden, wachsen die Herausforderungen bei der Überwachung und Verwaltung der Netzleistung. Die dafür erforderliche Software erfordert in der Regel spezielles Fachwissen, was die rechtzeitige Erkennung und Behebung von Problemen wie dem Ausfall von Funkverbindungen oder verteilten Denial-of-Service-Angriffen (DDoS) erschwert. Diese Komplexität verdeutlicht den Bedarf an Lösungen, die eine effiziente und skalierbare Analyse von Netzwerkdaten ermöglichen.

Diese These schlägt ein Multi-Agenten-Framework vor, das den NWDAF EventSubscription Service nutzt, um eine intelligente Ereignisanalyse und Kommunikation in einem 5G-Netzwerk zu ermöglichen. Das Framework sammelt, korreliert und verarbeitet dynamisch Ereignisdaten aus verschiedenen Netzwerksektoren und ermöglicht so eine effiziente Anomalieerkennung und Ursachenanalyse. Spezialisierte Agenten abonnieren die NWDAF-Ereignisbenachrichtigungen und erhalten relevante Daten für relevante überwachte Ereignisse auf der Grundlage der einzelnen Netzwerkschichten. Ein zentraler Aggregationsagent sammelt und integriert diese Ereignisberichte, formatiert die Informationen und wendet vordefinierte Analyseregeln an, um verwertbare Erkenntnisse zu gewinnen.

Das Framework bietet einen flexiblen, 5G-agnostischen Ansatz, der es Betreibern ermöglicht, maßgeschneiderte Agenten zu erstellen, die auf ihre individuellen Bedürfnisse zugeschnitten sind. Diese Lösung ermöglicht es Netzbetreibern und Nichtfachleuten, den Netzstatus intuitiver und übersichtlicher zu überwachen. Sie verkürzt die Zeit, die benötigt wird, um ein Problem zu bemerken und Abhilfemaßnahmen zu ergreifen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter discusses several pivotal challenges and opportunities presented by the evolving landscape of 5G technologies and network management practices.

1.1 Motivation

5G networks enable applications like IoT, essential communications, and V2X by bringing ultra-low latency, large data speeds, and ever-expanding interconnectedness. Therefore, these advancements have significantly increased the levels of complication. Multiple technology architectures have become progressively more daunting for network operators and administrators alike, and without specialized tools, it becomes tough to manage these networks. Traditional systems for network management tend to present overly technical data, limiting their accessibility and usability for broader audiences [19].

The NWDAF, as standardized by 3GPP, plays a crucial role in leveraging AI and machine learning to enhance 5G core network functionalities. By analyzing data collected from various 5G network functions (NFs), NWDAF supports anomaly detection, load balancing, and UE behavior analytics [18]. While studies have shown the NWDAF's capabilities, focusing on anomaly detection and cybersecurity, such as identifying DDoS attacks and unexpected UE mobility [19], its current iteration lacks accessible tools for operators and does not provide flexibility for developing new analytical use cases [14]. A multi-agent framework, proposed in this thesis, overcomes these issues and improve NWDAF's flexibility and usability. The framework uses a modular architecture to deploy specialized agents at the backend, user, and physical network levels. These agents collaborate to filter and correlate event data, reducing information overload and improving anomaly detection capabilities [14]. An aggregation agent consolidates this data into actionable insights, visualized through intuitive dashboards and maps, making network management accessible even for operators with limited technical expertise [18].

The approach aims to improve the operational workflow and simplify the sophisticated 5G functionality for network management.

1.2 Research Questions

While the NWDAF framework effectively leverages AI and machine learning for network analytics, its greatest strength is also one of its weaknesses: it gathers and outputs so much data that its usability requires a lot of background knowledge and expertise. This thesis aims to explore the following research questions:

1. How can a multi-agent framework enhance the accessibility and usability of NWDAF?
2. What strategies can effectively correlate anomalies across the physical, user, and backend network layers to improve root cause analysis?
3. How can integrating real-time data aggregation and visualization tools improve the comprehensibility of complex 5G network data for non-technical users?
4. What are the design considerations for ensuring the multi-agent framework remains scalable and adaptable to future analytical use cases?

This research designs a viable modular multi-agent system that works with NWDAF to address these problems. Enhancing the system with a web-based user interface (UI) that visualizes the data for better readability and usability of the outcome.

1.3 Goal

The main goal is to design and implement a multi-agent system integrated with NWDAF to enhance 5G network monitoring. The system includes three agents: the Physical Layer Inspector for detecting physical layer issues like radio link failures, the User Monitor Agent for identifying frequent service access attempts, and the Strategic Advisor for backend anomalies such as potential DDoS attacks. Hermes, the central agent, aggregates and analyzes data from all agents and forwards it to the database. This configuration aids in correlating network problems and the real-time identification of disturbances such as jamming, traffic irregularities, and cyberattacks (see Figure 1.1). Users may swiftly make well-informed decisions with an integrated dashboard showing network performance, risks, and health.

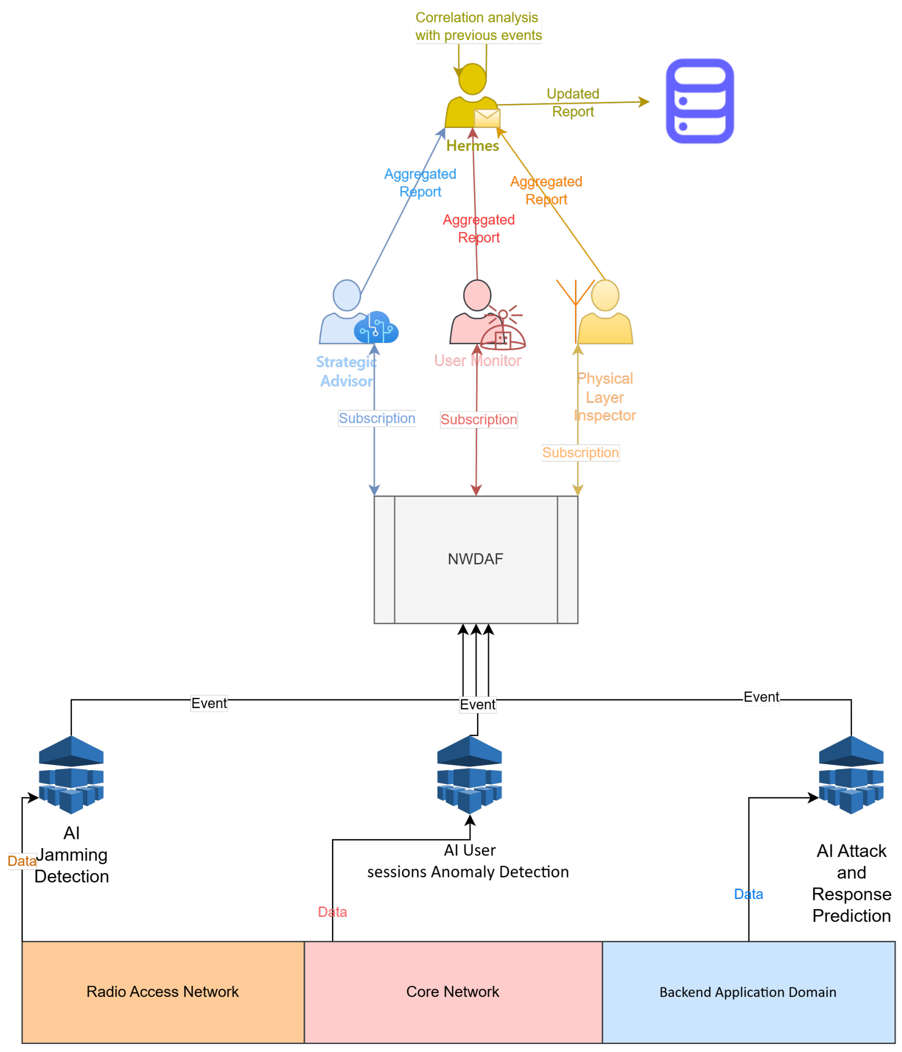


Figure 1.1: Illustration of the Multi-Agent System Architecture

1.4 Contribution

This thesis seeks to improve data readability and operational efficiency to streamline 5G network administration. The multi-agent system uses data aggregation and visualization to show how automated root cause analysis (RCA) can effectively discover and resolve network anomalies. RCA offers significant benefits over traditional strategies by automating the identification of root causes, reducing downtime, and minimizing the dependency on human expertise [17] [15]. Studies have demonstrated the importance of RCA in addressing the complexity of dynamic and heterogeneous environments like 5G and 6G. For instance, Canastro et al. employed graph-based dependency analysis for automating RCA, allowing the correlation of events across multiple layers and domains to identify underlying issues accurately [15]. Adaptive methods, such as the Bayesian-based approaches presented by Mfula and Nurminen enable RCA systems to handle incomplete or uncertain data while continuously improving through domain-specific learning [17]. The suggested architecture's emphasis on RCA fits nicely with the 5G network management automation requirement, significantly improving NWDAF's usability for real-time anomaly detection. By improving NWDAF's current RCA, a gap in the solutions is filled, and faster issue resolution is possible by increasing the readability of the network state. The framework's flexibility allows for integrating future analysis tools into the multi-agent system. The multi-agent system empowers operators to make timely, educated decisions by improving diagnosis and completely visualizing related anomalies. Connecting technical knowledge with practical results encourages greater accessibility and participation in 5G network administration.

1.5 Structure of the Thesis

Chapter 2: Background: An extensive summary of the thesis's fundamental elements is given in this chapter. First, the fundamentals of 5G architecture are presented, with particular attention paid to the 5G Core Network (5GC), Service-Based Architecture (SBA), and network functions, including AMF, SMF, and UPF. The chapter explains how they help with data management, connection, and ensuring efficient resource management while introducing a crucial analytical component of the 5G ecosystem: the Network Data Analytics Function (NWDAF). NWDAF's integration with the SBA and its functionalities, including data collection, real-time analytics, and anomaly detection, are discussed in detail and supported by diagrams of its architecture and event subscription mechanisms.

Chapter 3: Approach: This chapter outlines the methodological framework adopted to develop a multi-agent system for root-cause analysis in private 5G networks. The chapter begins by explaining the system architecture, which integrates specialized agents like the Physical Layer Inspector, Strategic Advisor, User Anomaly Agent, and

Hermes Agent. It emphasizes how the anomaly detection and event subscription features of NWDAF have been integrated into the system. The chapter elaborates on the development of every one of the agents, their roles, communication protocols among the agents, and how they work together in improving network administration and monitoring. Provide a complete description of the role of the Hermes Agent in data aggregation and correlation analysis, supplemented with flow diagrams illustrating the data processing and interaction between the agents.

Chapter 4:Implementation: Chapter 4 discusses implementing the multi-agent framework. It presents the technologies and software stacks used, such as Python, FastAPI, MongoDB, Docker, and React.js, and describes their roles in the system architecture. It provides the development of the File Data Reporting System, the deployment of expert agents, and how all these components cooperate to simulate NWDAF's functionality effectively.

Chapter 5: Simulation Setup: This chapter outlines the simulation environment, such as the hardware and software settings, deployment of agents, file creation procedures, and data workflow processes. It outlines how the simulated NWDAF EventSubscription system was configured to handle event notifications and correlation analysis. It outlines the configuration of the visualization dashboard, providing interactive monitoring and analysis of network events and anomalies.

Chapter 6: Evaluation: The evaluation chapter assesses the effectiveness of the utilized multi-agent framework based on metrics such as anomaly detection accuracy, correlation effectiveness, processing performance, and scalability. The assessment validates the system's capability to accurately detect and correlate network anomalies and maintain stability under varying load patterns. It demonstrates its viability and potential for use in real-world applications.

Chapter 2

Background

2.1 Essential 5G Architecture Components

Knowing how the 5G network works and its interconnectedness requires understanding some of its most important elements and concepts. These elements underlie data handling, connectivity, and networks and are the basis for 5G's improved service capability. [7]

The 5G Core Network (5GC) serves as the central hub of the 5G ecosystem, enabling the advanced functionalities and flexibility that define the next generation of telecommunications. Built on a Service-Based Architecture (SBA), the 5GC uses modular components that interact seamlessly via standardized APIs over HTTP/2. This structure ensures dynamic scalability and supports diverse applications, from streaming high-definition videos to enabling real-time communications for autonomous vehicles [7].

The simulation uses the File Data Reporting System described in TS 28.532 [6] to supply OAM data in Performance, Proprietary, Analytics, and Trace files. It acts as a middleman for storing and retrieving event data to guarantee that agents can efficiently access and process it for additional analysis. There are many endpoints in the service:

- GET /files: Reads information about available files.
- POST /subscriptions: Creates a subscription to be notified when files become available.
- DELETE /subscriptions/subscriptionId: Deletes a subscription.

The service has also been extended with additional endpoints:

- POST /files: Creates a file report.
- GET /files/fileId: Reads the contents of a single file.

- DELETE /files/fileId: Deletes a file.
- POST /files/create_many: Bulk creates file reports.
- GET /subscriptions/subscriptionId: Reads a subscription.

These capabilities allow the File Data Reporting System to manage event notifications efficiently, supporting the agents in retrieving, filtering, and analyzing the event data.

It administers user authentication, connection initiation, and mobility. The **Access and Mobility Management Function (AMF)** is a network component responsible for authenticating users in the network, guarding connections, and founding mobility transitions along service areas. For instance, the AMF makes it easier for cell towers to handover when a person is taking a train between cities, guaranteeing continuous internet and phone service [5] [7].

The **Session Management Function (SMF)** oversees the establishment and continuity of user sessions, including assigning IP addresses and reserving network resources for active sessions. For example, when a user begins streaming a movie, the SMF controls the data flow and keeps the connection alive, even in high network traffic, to guarantee that the session stays steady [5] [7].

The **User Plane Function (UPF)** is the traffic controller of the network, responsible for routing and forwarding user data efficiently while enforcing Quality of Service (QoS) policies. For instance, the UPF manages other users' internet traffic on the same network while prioritizing the video call data during a video conference to reduce latency and guarantee a smooth and easy experience [8].

The **Policy Control Function (PCF)** is the brain behind the network's resource management, enforcing rules for data usage, resource allocation, and QoS. It ensures users receive the appropriate service quality based on their subscription plans. For instance, if a mobile user has a premium data plan, the PCF ensures they have prioritized access to high-speed internet even during peak usage hours while enforcing data caps for standard users [5] [7].

The **Unified Data Management (UDM)** acts as the central database for user subscription data and authentication information. The UDM guarantees a smooth network connection without needing extra setup by retrieving users' subscription information when they put their SIM cards into a new device. To maintain secure connections, it protects user-profiles and oversees crucial authentication procedures [7].

The repository of all running network functions is the **Network Repository Function (NRF)**. It maintains a running service record and enables other components to locate them more easily. The NRF refreshes its database whenever a new network function is deployed so that integration is smooth throughout all the components, ensuring dynamic and consistent network performance. [7].

The **Network Exposure Function (NEF)** provides a secure interface for external applications to interact with the network. It exposes network capabilities, such as location data or traffic information, to authorized third-party applications. To give consumers precise projected arrival times while adhering to stringent data security requirements, ride-sharing software, for instance, can use the NEF to get real-time traffic information [7].

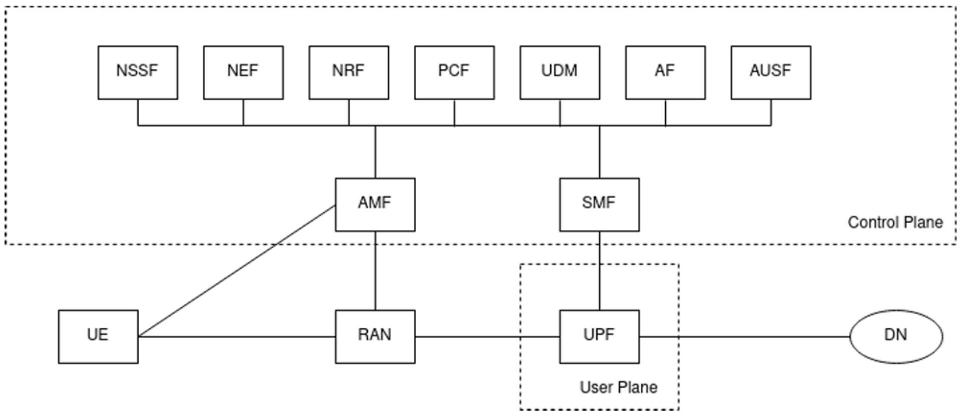


Figure 2.1: System architecture for the 5G System (5GS) (TS 23.501) [7], core components and network functions

These elements form the 5G Core Network's backbone, allowing the 5G ecosystem to function intelligently and scalable. The 5GC ensures that the network can adapt to the demands of contemporary digital services and changing user requirements by dynamically controlling connectivity, traffic, and resources.

2.2 Introducing the Network Data Analytics Function (NWDAF)

NWDAF is a component of the 5G architecture; operating within the SBA, NWDAF processes vast amounts of data from various network functions, transforming it into actionable insights that enhance the efficiency, adaptability, and overall performance of the 5G system [4]. Using standardized interfaces such as `Nnwdaf`, NWDAF integrates with other 5G core functions, contributing to dynamic optimization and Quality of Service (QoS) management [1]. Rather than being a standalone entity, NWDAF operates as a service that interacts with the broader 5G ecosystem. NWDAF collects data from essential network functions, including the AMF, the SMF, and the UPF; this data is then processed and analyzed in real-time, which can then be output as insight regarding the network operation and how it can be supported [4, 1].

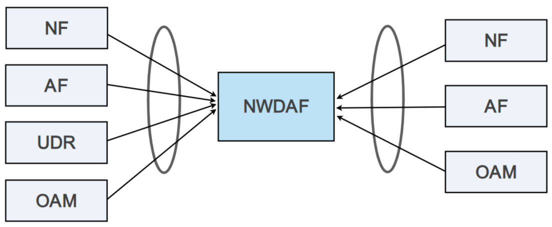


Figure 2.2: Simplified NWDAF model [16]

NWDAF's functionality encompasses three main processes. Data collection first entails integrating with network functions to obtain essential metrics, like mobility status from AMF, session details from SMF, and traffic loads from UPF [5]. This extensive data offers a thorough understanding of network behavior and performance. Second, data analysis processes the gathered data using sophisticated algorithms and machine learning models. NWDAF can detect anomalies like radio link failures, forecast traffic demands, or identify security threats such as DDoS attacks [1]. Finally, NWDAF manages notification and integration, communicating insights to relevant network functions or external systems [2]. These notifications enable network components to adapt dynamically, respond to anomalies, and maintain optimal real-time performance.

NWDAF comprises several modular components to achieve these capabilities. The Data Collection Module retrieves raw data from network functions, forming the foundation for further analysis [4]. The Analytics Engine applies statistical and machine learning techniques to generate insights and detect patterns within the data [1]. Lastly, the Notification Manager ensures that the results are distributed to subscribers, whether internal network functions or external applications, ensuring consistent integration across the network [1].

NWDAF's analytics capabilities empower various critical applications within the 5G network. For example, it facilitates anomaly detection, which finds anomalies like unexpected user behavior or radio connection failures and allows quick remedial action [3]. NWDAF helps to reduce congestion and optimize resource allocation by projecting future consumption patterns using traffic forecasting. Because of its function in QoS optimization, network parameters are dynamically changed to preserve service quality. The mobility management feature recognizes and fixes handovers and connection drop problems.

NWDAF also integrates with other components of the 5G architecture, relying on standardized communication protocols and interfaces [2]. The SBA enables NWDAF to operate flexibly and scalably, using APIs over HTTP/2 for interactions between network functions. NWDAF uses the `Nnwdaf` interface to send and receive data, making it an indispensable part of the 5G ecosystem [4].

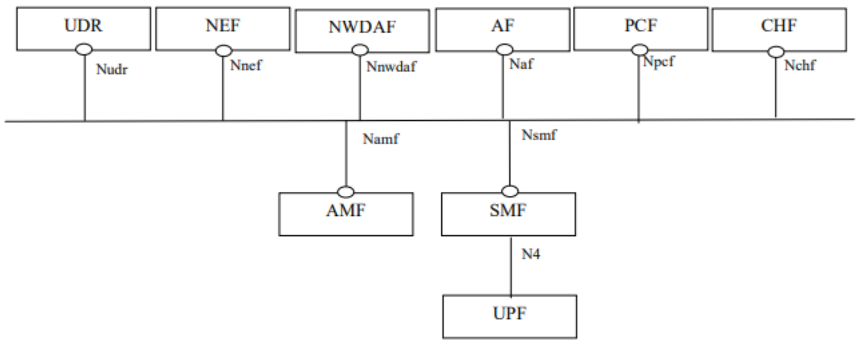


Figure 2.3: Nnwdaf Data Management Service architecture

Notifying and subscription to events are among the most essential roles of NWDAF. Network capabilities such as AMF, SMF, or PCF can subscribe to NWDAF for specific event-based analytics. For instance, NWDAF can monitor odd traffic patterns, abrupt user mobility, or network congestion. [1] Upon these events, NWDAF informs the subscribers through their designated `notificationURI`. With these notifications, network operations can promptly respond and evade possible problems or optimize resources according to necessity.

There are multiple steps in the event subscription process. Specifying the kind of analytics needed and the thresholds or criteria for triggering notifications in a request sent to NWDAF by a network function [1]. Once subscribed, NWDAF continuously monitors the requested metrics. When meeting the predefined conditions, NWDAF sends notifications to the subscriber. Subscriptions can be modified or deleted, offering flexibility for network operations.

2.3 NWDAF Event Subscription & Anomaly Detection

NWDAF Event Subscription is a feature in 5G networks that allows different network functions to request and receive real-time analytics on specific network conditions [1]. This mechanism helps improve network performance by detecting congestion, anomalies, and QoS issues before they affect users. Instead of passively collecting data, NWDAF enables network components to subscribe to important analytics and receive notifications only when meeting predefined conditions.

The process begins when a network function, such as the AMF or SMF, sends a subscription request to NWDAF. This request specifies what kind of analytics is needed, the conditions under which notifications should be sent, and where (a URL or endpoint) the messages should be delivered. For example, an SMF might request updates on traffic congestion so it can take action before network slowdowns occur.

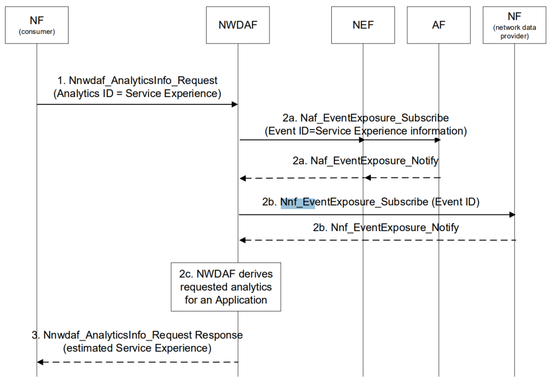

NF (consumer)
NWDAF
NEF
AF
NF (network data provider)
1. Nnwdaf_AnalyticsInfo_Request (Analytics ID = Service Experience)
2a. Naf_EventExposure_Subscribe (Event ID=Service Experience information)
2a. Naf_EventExposure_Notify
2b. Nnf_EventExposure_Subscribe (Event ID)
2b. Nnf_EventExposure_Notify
2c. NWDAF derives requested analytics for an Application
3. Nnwdaf_AnalyticsInfo_Request Response (estimated Service Experience)

Figure 2.4: steps involved in NWDAF event exposure [4]

After subscribing, NWDAF continuously monitors and evaluates network data. It notifies the requesting function when the predetermined criteria are satisfied. Increased traffic volume, odd user behavior, or an abrupt decline in signal strength are a few examples of these alerts. These alerts enable the network to respond rapidly, for instance, by real-time resource optimization or traffic rerouting to avoid congestion.

If a request contains errors, such as missing parameters or an unsupported event type, NWDAF returns an error message so the request can be corrected.

NWDAF supports different types of event subscriptions, each designed to enhance specific aspects of network management [1]:

- **Congestion Monitoring**: Helps prevent overload by notifying network operators when parts of the network experience high traffic.
- **Anomaly Detection**: Identifies unusual behavior, such as suspicious spikes in data traffic that could indicate security threats (e.g., DDoS attacks).
- **Quality of Service (QoS) Optimization**: Ensures that users receive the best possible service by detecting issues like slowdowns in data speed or dropped connections [1].
- **Mobility Tracking**: Monitors user movements to detect handover failures or connection losses, helping maintain stable connectivity.

A practical example of NWDAF in action is video streaming services. A PCF managing a video streaming platform can subscribe to NWDAF to monitor network congestion . Suppose NWDAF detects that video streaming quality is at risk due to limited bandwidth. In that case, the PCF can automatically adjust streaming quality (e.g., temporarily lowering video resolution) to maintain a smooth user experience without

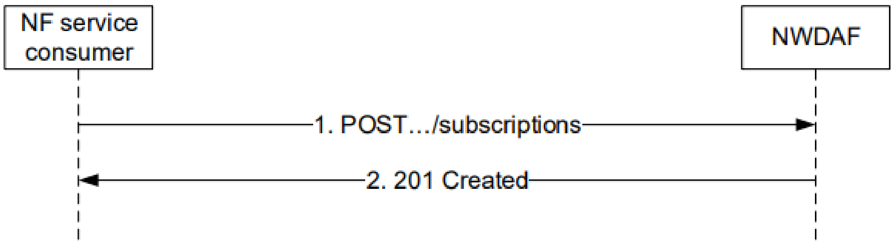


Figure 2.6: Flow of an NF service consumer subscribing to NWDAF notifications [1]

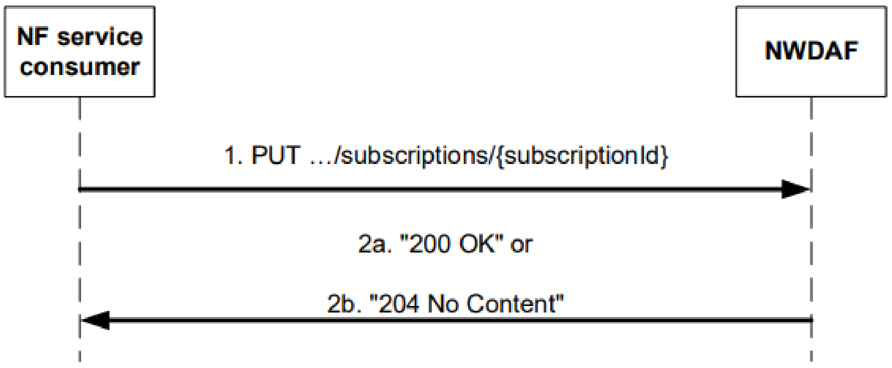


Figure 2.7: Process for updating an NWDAF subscription [1]

buffering. Similarly, a cloud gaming service could subscribe to latency analytics to adjust server resources in real-time to avoid lag during gameplay.

SBA employs specific interfaces to enable interaction between network procedures in 5G. Without disrupting ongoing operations, the AMF, SMF, PCF, and NSSF can request analytics through the Nnwdaf_EventsSubscription service. This structure allows the decision-making process to be Real-time and data-driven, verifying the network's continued operational efficiency and flexibility.

NWDAF enables real-time monitoring of network anomalies through event subscriptions, allowing network functions to detect and respond to unusual behavior patterns that may indicate network issues or security threats. By subscribing to anomaly-related events, network functions receive immediate notifications when meeting predefined conditions, facilitating proactive mitigation measures [1].

Anomaly detection subscriptions define the type of anomaly to monitor, trigger conditions, and the destination for notifications. For example, an SMF may subscribe to analytics for unexpected large-rate data flows, indicating a temporary surge in user activity or a potential DDoS attack . Similarly, an AMF could subscribe to unexpected user mobility patterns, helping detect misconfigurations, interference, or unauthorized access attempts.

To ensure accurate anomaly detection, NWDAF continuously monitors data streams from sources like UPF and PCF, applying statistical methods and machine learning algorithms to identify deviations from normal behavior. If an anomaly exceeds a predefined threshold, such as an unusual increase in handover failures or a surge in signaling messages, NWDAF generates an event notification. The subscribing function can then take immediate corrective action, such as adjusting mobility parameters or activating additional security measures.

For example, suppose NWDAF detects a persistent ping-ponging effect where a device frequently switches between multiple cells. In that case, an AMF receiving the anomaly notification can adjust handover parameters to stabilize the connection. Likewise, a PCF monitoring QoS degradation may receive notifications indicating abnormal service fluctuations, prompting adjustments to optimize network resources dynamically [1].

Network operations can swiftly adjust to conditions because of dynamic flexibility in changing or terminating anomaly detection subscriptions. This dynamism minimizes erroneous notifications and guarantees that metrics are pertinent to ongoing network operations.

NWDAF's integration of anomaly detection and event subscription systems allows for enhancements in quality of service, operational security, and practical resource allocation. Real-time anomaly identification and reaction are critical to 5G network stability and high-quality service delivery.

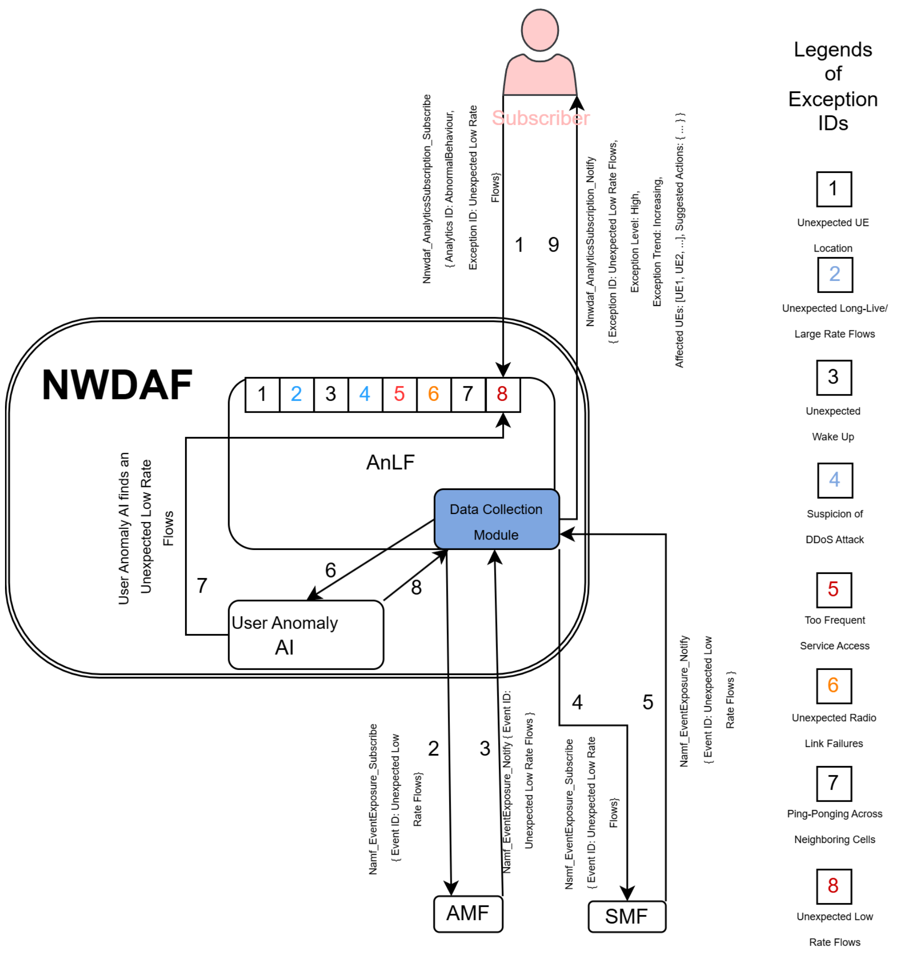

Subscriber
Nwdaf_AnalyticsSubscription_Subscribe { Analytics ID: AbnormalBehaviour, Exception ID: Unexpected Low Rate Flows}
1
9
Nwdaf_AnalyticsSubscription_Notify { Exception ID: Unexpected Low Rate Flows, Exception Level: High, Exception Trend: Increasing, Affected UEs: [UE1, UE2, ...], Suggested Actions: { ... } }
NWDAF
1 2 3 4 5 6 7 8
AnLF
Data Collection Module
User Anomaly AI finds an Unexpected Low Rate Flows
7
6
8
User Anomaly AI
Namf_EventExposure_Subscribe { Event ID: Unexpected Low Rate Flows}
2
3
Namf_EventExposure_Notify { Event ID: Unexpected Low Rate Flows }
4
5
Nsmf_EventExposure_Subscribe { Event ID: Unexpected Low Rate Flows}
Namf_EventExposure_Notify { Event ID: Unexpected Low Rate Flows }
AMF
SMF
Legends of Exception IDs
1 Unexpected UE Location
2 Unexpected Long-Live/ Large Rate Flows
3 Unexpected Wake Up
4 Suspicion of DDoS Attack
5 Too Frequent Service Access
6 Unexpected Radio Link Failures
7 Ping-Ponging Across Neighboring Cells
8 Unexpected Low Rate Flows

Figure 2.8: Process detecting an anomaly and notifying subscriber

Chapter 3

Methodology

This chapter presents the approach used to create the multi-agent system. The system improves network comprehension by utilizing the NWDAF's capabilities. The Physical Layer Inspector, Strategic Advisor, User Anomaly Agent, and Hermes Agent are the four specialist entities that make up the multi-agent architecture. To ensure a thorough and integrated approach to network management, each agent handles different facets of network monitoring, data processing, and visualization.

3.1 System Architecture Overview

The NWDAF system processes data through its Data Collection Module, gathering inputs from network functions like the AMF and the SMF. These inputs include exception types (e.g., "Unexpected Low-Rate Flows") and additional network statistics. This data can be analyzed using an implemented AI approach. This AI uses machine learning models within the AnLF to detect anomalies, such as unexpected low-rate flows or Too Frequent Service Access. The AI detects and forecasts anomalies based on predetermined metrics. In these situations, NWDAF allows external entities, such as the User Anomaly Agent, to subscribe to anomaly reports using Exception Subscription and Notification. The AMF, SMF, or other functions receive notifications that describe anomalies such as "Unexpected Low Rate Flows" and other problems unique to a particular UE. Figure 3.1 shows a detailed example of the data flow. Following receipt of the anomaly report to which it has subscribed, the data is forwarded to the Hermes agent, who compares it to previously submitted reports and looks for correlations. The report is then improved based on the findings and published in the database. The system architecture is schematically represented in Figure 3.2, which also shows how the agents communicate and exchange data.

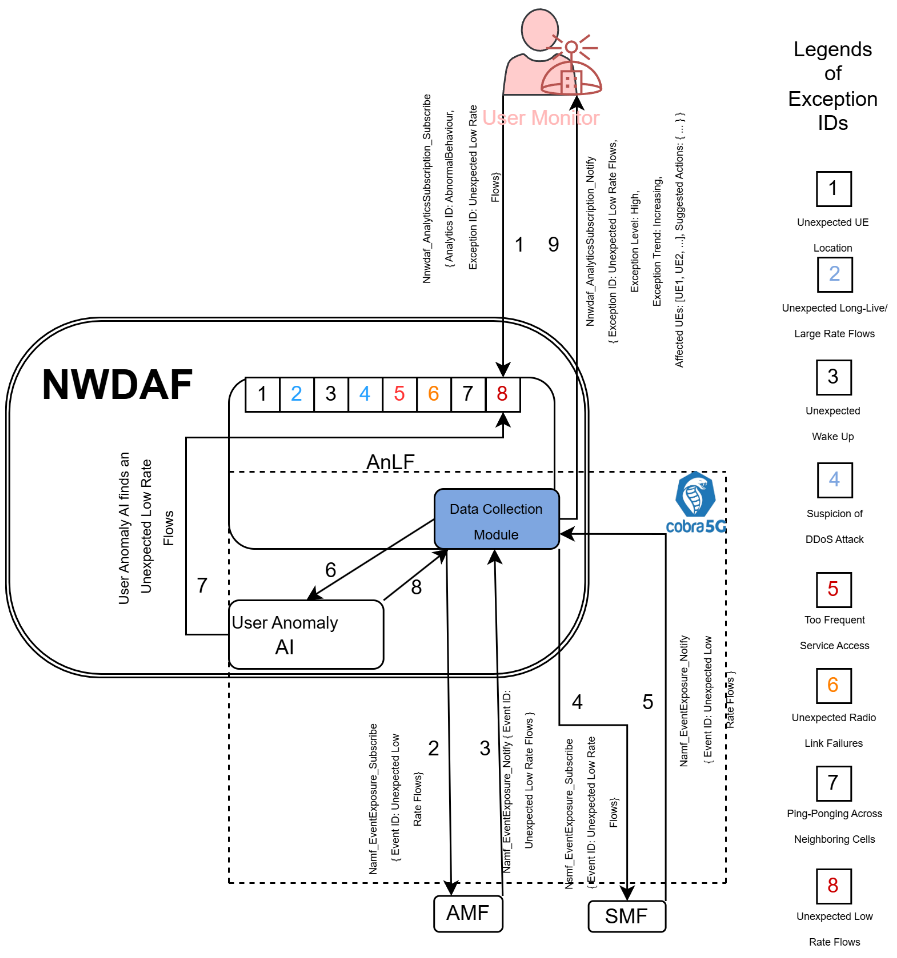

User Monitor
Nnwdaf_AnalyticsSubscription_Subscribe { Analytics ID: AbnormalBehaviour, Exception ID: Unexpected Low Rate Flows)
1
9
Nnwdaf_AnalyticsSubscription_Notify { Exception ID: Unexpected Low Rate Flows, Exception Level: High, Exception Trend: Increasing, Affected UEs: [UE1, UE2, ...], Suggested Actions: { ... } }
NWDAF
1 2 3 4 5 6 7 8
AnLF
User Anomaly AI finds an Unexpected Low Rate Flows
7
Data Collection Module
cobra5G
6
8
User Anomaly AI
Namf_EventExposure_Subscribe { Event ID: Unexpected Low Rate Flows}
2
3
Namf_EventExposure_Notify { Event ID: Unexpected Low Rate Flows }
4
Nsmf_EventExposure_Subscribe {Event ID: Unexpected Low Rate Flows}
5
Namf_EventExposure_Notify { Event ID: Unexpected Low Rate Flows }
AMF
SMF
Legends of Exception IDs
1 Unexpected UE Location
2 Unexpected Long-Live/ Large Rate Flows
3 Unexpected Wake Up
4 Suspicion of DDoS Attack
5 Too Frequent Service Access
6 Unexpected Radio Link Failures
7 Ping-Ponging Across Neighboring Cells
8 Unexpected Low Rate Flows

Figure 3.1: Illustration of what happens within NWDAF

3.2 Problem Definition

The conventional approach to network diagnostics often involves manual investigation, which is inefficient and time-consuming. The introduction of the NWDAF within the 5G architecture aims to enhance network intelligence through real-time data analytics and event subscriptions. Nevertheless, practical experiments need real-life deployments that are not found in most research environments, and NWDAF implementations are not always present. To avoid the lack of a true NWDAF, this system emulates a simulated NWDAF EventSubscription system based on a File Data Reporting system. This simulation allows for organized anomaly detection and root cause analysis in private 5G networks.

3.3 Agent Development

The choice of a multi-agent architecture is motivated by its scalability, modularity, and potential for future AI integration. Each agent is responsible for a specific type of anomaly, ensuring targeted filtering and schema adherence while reducing the complexity of the anomaly detection system. Each agent within the multi-agent system facilitates the creation of asynchronous and cooperative agents. The agents communicate through predefined protocols, enabling robust data exchange and coordinated actions. Below is a detailed exposition of each agent's development, functionality, and interaction mechanisms. All three Agents interact with Hermes in a similar fashion.

3.3.1 Physical Layer Inspector Agent

The Physical Layer Inspector Agent subscribes to the NWDAF's EventSubscription Anomalies, including "Unexpected Radio Link Failures" and "Unexpected UE Location." Its primary responsibilities include continuously tracking signal strength, interference levels, and radio link quality to identify anomalies such as jamming or hardware failures. The agent collects and processes data related to these physical layer events, forwarding the information to the Hermes Agent for further analysis. The agent does not perform correlation analysis. Moreover, it mainly receives the data from NWDAF with its specific filter and then sends it to Hermes.

Implementing the Physical Layer Inspector Agent begins with subscribing to relevant data streams and ingesting event notifications. To ensure real-time updates on important events like unexpected radio link failures, the agent subscribes to NWDAF event notifications using a POST request to $create\_nwdaf\_event\_subscription$. The agent can function effectively asynchronously since these alerts are obtained via an API.

3.3.2 Strategic Advisor Agent

Like the Physical Layer Inspector, the Strategic Advisor Agent receives data from the NWDAF's event subscription service based on anomaly filters like Suspicion of DDoS Attack and Unexpected Long-Live/Large Rate Flows. This agent does not perform any correlation analysis; instead, it focuses on collecting and forwarding data to the Hermes Agent, who is responsible for all analytical tasks.

The Strategic Advisor Agent focuses on significant parameters a backend attack could influence, such as anomaly reports, service quality measurements, and network usage patterns, enabling proper evaluation of network health and operation. Having past data also makes trend analysis easier. For example, it can identify patterns that indicate an imminent network traffic surge or recurring anomalies that impede service availability.

The Strategic Advisor interfaces with the Hermes agent, ensuring all relevant data is available for analysis.

3.3.3 User Anomaly Agent

The User Anomaly Agent monitors user network activity. It first records user sessions and behaviors to assist Hermes in detecting abnormalities like excessive service access attempts or odd patterns that can point to underlying network problems or security risks.

In addition to reacting to anomalies, the data from the User Anomaly Agent is used to perform correlation analysis to find relationships between user-related anomalies and other network events, including examining correlations such as the link between too frequent service access and unexpected radio link failures, unexpected low-rate flows, and suspicion of DDoS attacks.

Upon detecting anomalies, the User Anomaly Agent forwards the collected data to the Hermes Agent for further analysis.

3.3.4 Hermes Agent

The Hermes Agent is the data correlation and analysis hub of the multi-agent environment, as depicted in figure 3.2. It listens to event notifications, analyzes the received data, and applies correlation rules to determine correlations among anomalies.

When receiving event data, the agent normalizes the timestamps to ensure a consistent format for analysis. It retrieves recent events from MongoDB to compare new events with previously recorded anomalies. The correlation rules are predefined, based on which the agent evaluates potential relationships between anomalies based on typical user identifiers, temporal proximity, and geographical closeness. Additional metrics, such as signal quality and confidence levels, further refine the correlation assessment.

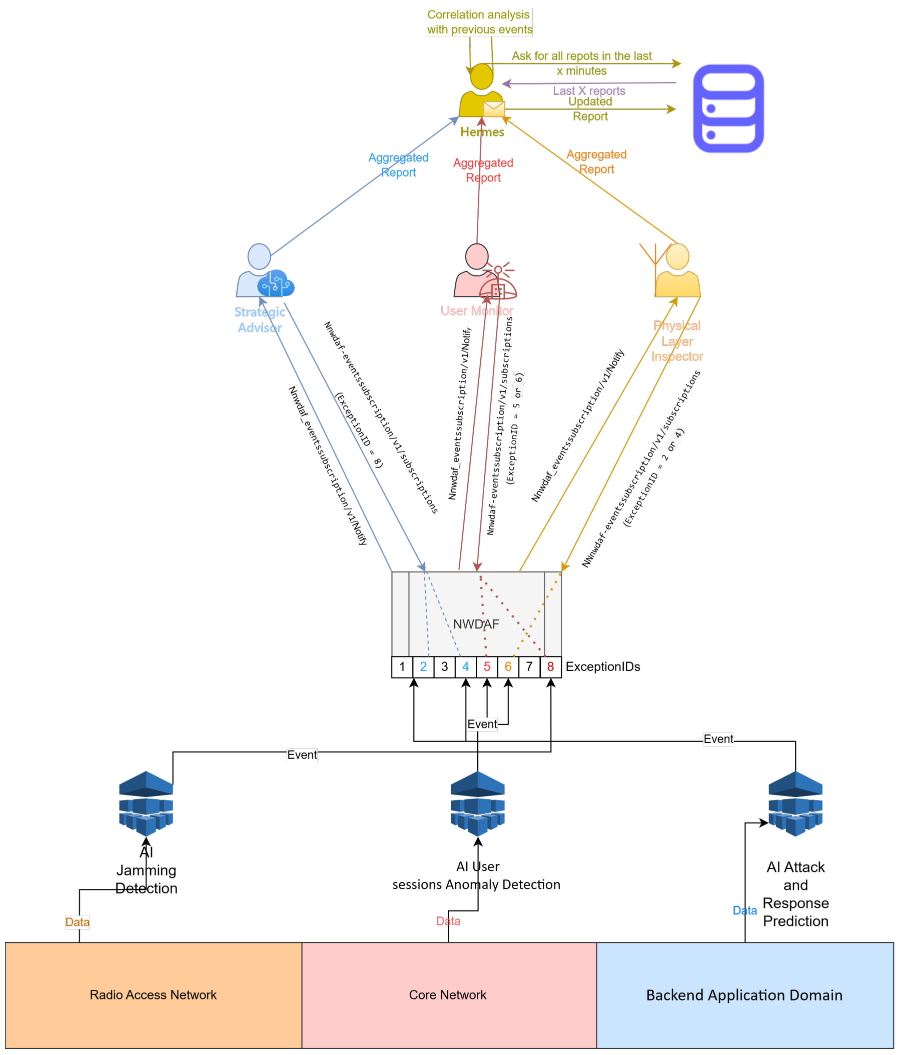

Correlation analysis with previous events
Ask for all repots in the last x minutes
Last X reports
Updated Report
Hermes
Aggregated Report
Aggregated Report
Aggregated Report
Strategic Advisor
User Monitor
Physical Layer Inspector
Nnwdaf_eventssubscription/v1/Notify
Nnwdaf-eventssubscription/v1/subscriptions (ExceptionID = 8)
Nnwdaf_eventssubscription/v1/Notify
Nnwdaf-eventssubscription/v1/subscriptions (ExceptionID = 5 or 6)
Nnwdaf_eventssubscription/v1/Notify
NNwdaf-eventssubscription/v1/subscriptions (ExceptionID = 2 or 4)
NWDAF
1 2 3 4 5 6 7 8
ExceptionIDs
Event
Event
Event
AI Jamming Detection
AI User sessions Anomaly Detection
AI Attack and Response Prediction
Data
Data
Data
Radio Access Network
Core Network
Backend Application Domain

Figure 3.2: Illustration of the Multi-Agent System Architecture

On detection, the correlation score is added to the event JSON. The event is then processed for data related to the AMF, SMF, and UPF, and upon finding it, a standalone file is created with the corresponding schema and reposted back to the File Data Reporting System. The event is then posted to MongoDB for downstream correlation analysis and visualization. This facilitates long-term observation of network anomalies and detecting patterns that repeat themselves. Network operators can use the data stored to analyze previous incidents and make decisions about network optimization and security.

The Hermes Agent includes a continuously running FastAPI server that processes event notifications asynchronously. This improves the possibility of correlation detection and guarantees that the real-time correlation analysis can manage several concurrent occurrences without losing data due to interference. It also keeps a history of abnormalities, improves network performance and dependability, and offers insights for troubleshooting.

When the Hermes Agent receives an anomaly, if a correlation is found (e.g., Unexpected Radio Link Failures linked to Too Frequent Service Access), the event is enhanced with correlation metadata before being formatted according to its respective NWDAF schema and uploaded to the File Data Reporting System.

The Hermes Agent stores AMF, SMF, and UDM events in their respective formats:

AMF Events (Trace Category): Events related to UE mobility and registration issues (e.g., Unexpected Radio Link Failures) are stored following TS 29.518 schemas [12].

SMF Events (Analytics Category): PDU session and traffic anomalies (e.g., Unexpected Low Rate Flows) are formatted according to TS 29.502 [10].

UDM Events (Proprietary Category): Authentication and reachability issues (e.g., Too Frequent Service Access) adhere to TS 29.503 [11].

The Hermes Agent dynamically determines the correct schema and file category, ensuring subscribing agents can retrieve and interpret data correctly. This process maintains structured event processing while preserving correlation insights for anomaly investigation.

Furthermore, to enhance subscribers' visibility, each file uploaded includes a new 'fileComponent' field, which explicitly identifies whether the data belongs to AMF, SMF, or UDM. This addition allows external agents to filter and retrieve relevant data efficiently.

By integrating automated correlation detection and structured data uploads, the Hermes Agent bridges the gap between raw anomaly detection and meaningful root-cause analysis, making it a central component of the simulated NWDAF EventSubscription system.

3.4 Integration and Communication Protocols

Effective communication and seamless data integration among the agents are pivotal to the system's functionality. The agents interact through asynchronous messaging, ensuring non-blocking operations and real-time data processing.

Each agent operates within its container, maintaining a subscription with the NWDAF EventSubscription and communicating with the Hermes agent when receiving an event. The communication flow is structured as follows:

1. **Data Collection**: Data gathering agents such as the Physical Layer Inspector and User Anomaly Agent identify anomalies and gather data according to their expert role.

2. **Issue Reporting**: The agents report anomalies or correlations by passing on the data processing to the Hermes Agent.

3. **Data Aggregation and Visualization**: The Hermes Agent aggregates the received data, cleans it, and updates the database used for visualization, providing a comprehensive and accessible overview of network health.

4. **Strategic Insights**: The Strategic Advisor Agent analyzes aggregated data to identify long-term trends and could generate strategic recommendations in future implementations, which are also communicated to the Hermes Agent for visualization.

Chapter 4

Implementation

The multi-agent system ensures that data integration goes smoothly, communication is effective, and network anomalies are discovered and analyzed in real-time. This process happens across a stable architecture employing asynchronous programming—an absolute necessity when many specialized agents do something different. However, all of them (nearly all the time) need to be able to interact. The multi-agent system ensures seamless data integration. Every agent conducts its operations autonomously within its container and communicates with other agents via asynchronous messaging protocols provided by the aiomas framework, guaranteeing real-time data sharing, as communication among agents is non-blocking and, therefore, cannot be delayed by one agent waiting for another to finish a task. The User Anomaly Agent and Physical Layer Inspector gather and analyze data specific to their domains, such as unexpected network events and signal quality. When they detect something that is out of the ordinary, they alert the Hermes Agent. The Hermes Agent does for anomalies what the other agents do for data in their domains—it compiles, cleans, and saves them in a database accessible by any of the agents. The solution summarizes network health measures and supports advanced data visualization through an intuitive user interface. This interface enables interested parties to delve deeply into any particular issue while displaying important real-time metrics, such as the frequency and severity of anomalies. The workhorse behind the scenes is the ”Strategic Advisor Agent”—”simply put, a cluster of Axon-based customized agents.” This setup allows for long-term trend and optimization analysis on a scale not humanly possible. The primary programming language employed in the implementation is Python. It was chosen mainly due to its breadth of support for asynchronous programming, essential for the built multi-agent system. Its aromas—the structural core of the system—affect asynchronous inter-agent coordination and communications. Asyncio helps the agents control several concurrent operations and maintain the system responsiveness under high data loads. For HTTP-based interactions—such as safe management of NWDAF event subscriptions and alerts—they also employ aiohttp. The design combines these elements:

- **Modularity**
- **Scalability**
- **Real-time responsiveness**

They allow proper supervision, auditing, and portrayal of the intricate operations within 5G networks.

4.1 Software Stack

NWDAF EventSubscription simulation deployment based on the multi-agent system relies on an appropriately chosen software stack that supports scalability, modularity, and efficiency of real-time processing. This chapter describes the core software components and their role in the system.

4.1.1 Programming Languages and Frameworks

- **Python**: The primary programming language for implementing the agents, leveraging its extensive support for asynchronous programming and data processing libraries.
- **TypeScript**: Used for the File Data Reporting System to ensure type safety and maintainability in the backend services.
- **FastAPI**: A high-performance web framework in Python for developing asynchronous API endpoints for the agents.
- **Express.js**: Utilized for building RESTful API endpoints in the File Data Reporting System.

4.1.2 Data Storage and Management

- **MongoDB**: The primary database for storing event notifications, correlation results, and agent reports. Its flexible schema and indexing capabilities support efficient querying for correlation analysis.
- **Mongoose**: A MongoDB object modeling tool used in the File Data Reporting System for schema validation and data integrity.
- **MongoDB Memory Server**: Facilitates testing by providing an in-memory instance of MongoDB for integration tests.

4.1.3 Asynchronous and Messaging Support

- **aiomas**: A library for building asynchronous multi-agent systems in Python, enabling agents to operate concurrently without blocking each other.
- **asyncio**: A Python library for managing event loops, essential for handling asynchronous HTTP requests and real-time data processing.
- **aiohttp**: Used for making asynchronous HTTP requests between agents and the File Data Reporting System.

4.1.4 API Management and Documentation

- **OpenAPI**: Defines API specifications for the File Data Reporting System, ensuring standardized communication between agents and backend services.
- **Swagger UI**: Provides an interactive API documentation for testing and validating API endpoints.
- **Express OpenAPI Validator**: Ensures incoming File Data Reporting System requests comply with the OpenAPI schema.

4.1.5 Data Processing and Analysis

- **Pandas**: Utilized for data manipulation and analysis in the Hermes Agent, mainly for handling event notifications and correlation analysis.
- **Geopy**: A Python library used for geospatial analysis within the Hermes Agent to calculate distances between reported UE locations.
- **Numpy**: Assists in numerical computations for correlation scores and proximity analysis.

4.1.6 Web Dashboard and Visualization

- **React.js**: Used to develop the visualization dashboard for presenting real-time insights into network anomalies.
- **Mapbox**: Integrated for geospatial visualization of anomaly locations on interactive maps.
- **Axios**: A promise-based HTTP client for React.js, facilitating API requests to backend services for real-time data updates.
- **Bootstrap**: Employed for responsive UI design and layout in the web dashboard.

4.1.7 Development and Testing Tools

- **TypeScript**: Enhances code maintainability and type safety for the File Data Reporting System.
- **Jest and Mocha**: Testing frameworks for conducting unit and integration tests in the File Data Reporting System.
- **ESLint and Prettier**: Enforce coding standards and style consistency across the project.
- **Nodemon**: Facilitates automatic server restarts during development when file changes are detected.

4.1.8 Logging and Monitoring

- **Winston**: A logging library used in the File Data Reporting System for structured and level-based logging.
- **Grafana and Prometheus**: Monitoring tools for tracking system performance, resource utilization, and detecting bottlenecks during simulations.

4.1.9 Containerization and Orchestration

- **Docker**: Used for containerizing agents, the File Data Reporting System, and the Hermes Agent to ensure consistent deployment across different environments.
- **Docker Compose**: Manages multi-container deployments, defining the network and volume configurations for the entire system.

4.1.10 Schema and Standards Compliance

- **TS 28.532** [6]: Defines the File Data Reporting service specifications and the schema for event notifications.
- **TS 29.520** [9]: Provides the schema for NWDAF EventSubscription outputs, ensuring consistency and compliance with 3GPP standards.

The design of this software supports a modular and scalable architecture that can effectively handle high-rate event notices. Using a combination of asynchronous programming, containerization, and real-time data processing, the system effectively emulates the NWDAF EventSubscription model for anomaly detection and root cause analysis.

4.2 File Data Reporting system as NWDAF EventSubscription

No working implementation of a fully functional NWDAF was available for this thesis. Without a functional NWDAF, the NWDAF EventSubscription is simulated using a File Data Reporting mechanism. The framework provides a way to generate, store, and retrieve event notifications that mimic the role of the NWDAF to detect anomalies in private 5G networks. Structured file-based OAM data handling is out of band, with the four main types being trace and analytics. Proprietary and Performance (File Data Reporting) Endpoints that expose file management and subscription handling, enabling external agents to subscribe for specific event categories or dynamic data. A file generation script periodically creates simulated NWDAF EventSubscription outputs corresponding to various anomaly types observed in the network. Each generated file contains event notifications with attributes such as Subscriber Permanent Identifier (SUPI), timestamps, confidence levels, and additional contextual details depending on the anomaly type. The anomaly types are mapped to file categories as follows: Unexpected UE Location (Trace), Unexpected Long-Live Flows (Analytics), Suspicion of DDoS Attack (Proprietary), Too Frequent Service Access (Performance), Unexpected Radio Link Failures (Trace), Unexpected Low Rate Flows (Performance), and Unexpected Large Rate Flow (Analytics). The script sends the files to the File Data Reporting service HTTP POST requests, assigns unique IDs, and sets file expiration times. Periodically, the system creates several files to show real-time network abnormalities. The events contained in these files follow the schema defined in *TS29520_Nnwdaf_EventsSubscription.yaml*, ensuring consistency with standardized NWDAF event structures. The agent triggers the subscription process in File Data Reporting and then releases notices of the availability of relevant files to simulate the behavior of an NWDAF EventSubscription. Built with aiohttp and FastAPI, the agent proactively keeps track of file notices and handles related event information correctly. For example, the PhysicalLayerInspectorAgent subscribes to Trace-type files to detect unexpected radio link failures. Upon receiving a file notification, the agent retrieves file details, extracts relevant attributes, and forwards the structured information to the Hermes agent. The Hermes agent filters out irrelevant anomalies, forwarding only actionable insights for further processing. As a result, data processing and analytics workflows can be established even without NWDAF deployment. For clarity from this point on, when talking about the simulated NWDAF EventSubscription, it will be written as NWDAF EventSubscription unless talking specifically about the File Data Reporting system.

4.3 Implementation of Agent-Based Event Processing System

Python and FastAPI are used to implement the agent-based event processing system, which offers an asynchronous framework for managing event-driven network anomaly detection. The system simulates the NWDAF EventSubscription model by leveraging multiple specialized agents that subscribe to the File Data Reporting system, process relevant anomaly data, and forward structured insights to a central aggregation system. Each agent is implemented as an independent service, running on designated ports and communicating with the Hermes analytics system via HTTP.

4.3.1 Base Agent Architecture

The `BaseExaminerAgent` is built-upon asynchronous server built with FastAPI and is the primary building block of the agent structure. The agent can create dynamic HTTP endpoints to handle event notifications and has configurable settings like host, port, and file types to which it subscribes. By sending a POST request to the `/subscriptions` endpoint upon initialization, the agent specifies the file types it is interested in and subscribes to the File Data Reporting service. The agent uses an asynchronous HTTP GET request to retrieve all file details after extracting the pertinent data fields and validating the structure in response to an HTTP POST request notification. After being extracted, the data is formatted and sent to the Hermes system via aiohttp for additional analysis.

4.3.2 Physical Layer Inspector Agent

The `PhysicalLayerInspectorAgent` is a concrete subclass of the `BaseExaminerAgent` that has been designed specifically to handle `Trace`-type files. The agent listens on a target port and automatically opens up a route to deal with trace file notifications. When it receives a notification, it parses the JSON body to retrieve the `eventNotifications` element. It systematically traverses notifications and has a filter mechanism to find those matching the anomaly type `UNEXPECTED_RADIO_LINK_FAILURES`. Once it finds a corresponding match, it builds a structured packet of data sent asynchronously to the Hermes Agent using an HTTP POST. The entire mechanism is built using Python's asyncio framework to eliminate potential blocks.

4.3.3 User Monitor Agent

The `UserMonitorAgent` extends `BaseExaminerAgent` and subscribes to both `Trace` and `Performance` file types. It overrides the event handling mechanism to process anomalies such as TOO_FREQUENT_SERVICE_ACCESS and UNEXPECTED_LOW_RATE_FLOWS. When it receives a notification of a file, it issues an HTTP GET to fetch detailed information. Then, it examines the collection of anomalies to identify substantial attributes like SUPI values, timestamps, and confidences. The agent classifies a relevant anomaly once it is identified. The agent then structures information and forwards it to Hermes using aiohttp. The entire communication cycle is logged using Python's logging module for debugging and monitoring.

4.3.4 Backend Advisor Agent

The BackendAdvisorAgent subscribes to Analytics and Proprietary file types and specializes in detecting service-related anomalies such as UNEXPECTED_LONG_LIVE_FLOWS and SUSPICION_OF_DDOS_ATTACK. Implemented as a FastAPI service, it registers routes dynamically for handling file notifications and uses an asynchronous HTTP client to fetch file contents upon receiving a notification. The extracted data is parsed to identify patterns of prolonged session durations or excessive network traffic spikes. If a match is found, the agent formats the data and sends it to Hermes for further analysis. Leveraging FastAPI's event loop and concurrency model, the architecture efficiently manages high-throughput event streams—a Scalable, Asynchronous Approach to Anomaly Detection and Signal Aggregation using an Agent-based Framework. Non-blocking I/O operations based on network and file communications guarantee the low-latency processing of requests. The agent operates independently, providing modularity and flexibility. The framework closely emulates the NWDAF behavior within private 5G networks.

4.3.5 Hermes Agent for Event Correlation

The HermesAgent acts as a central processing unit aggregating and cross-checking anomaly reports received from other agents. The system is built as a FastAPI application that listens on a defined port and connects to a MongoDB database to guarantee stable persistence of information. When it receives anomaly information through an agent, the Hermes agent analyzes the received event, extracts vital parameters like SUPI, timestamps, and confidences, and uses pre-defined correlation rules to identify the correlation between anomalies.

The Hermes agent maintains a set of correlation rules that define conditions for associating different anomaly types. For example, it can identify if UNEXPECTED_RADIO_LINK_FAILURES frequently co-occur with TOO_FREQUENT_SERVICE_ACCESS within a short time window. This is achieved by querying the MongoDB database for recent events and comparing them based on time proximity, shared UE identifiers, and spatial relationships derived from location data. The system employs geospatial analysis using the Geopy library to calculate distances between UE locations, allowing the agent to assess whether anomalies such as `UNEXPECTED_UE_LOCATION` and `RADIO_LINK_FAILURES` are occurring nearby. The correlation engine then calculates a degree of confidence to assign to one of the discovered relationship types and structures the enriched information to prepare it for subsequent processing. When it finds a high-confidence correlation, the Hermes agent fires a new event notice and passes it to the File Data Reporting system via an HTTP POST request. This operation ensures that all such patterns found are recorded and available for additional investigation.

The Hermes agent offers an API endpoint that enables access to historical anomaly information so that exterior analytical software can consider previous trends. Its incorporation of real-time event correlation, asynchronous processing, and high-capacity storage results in functions that closely resemble those of NWDAF in private 5G networks. The feature enables efficient documentation and study of intricate network anomaly interrelations.

4.4 Visualization Approach

The visualization tool was built using `React`, `Bootstrap`, and `Mapbox`. This technological infrastructure supports effective monitoring and management by converting analyzed NWDAF EventSubscription data into actionable insights. Key features include real-time dashboards designed to identify anomalies like DDoS attacks, radio link failures, and high rates of accessed services. An interactive cartographic visualization shows the spatial distribution of anomalies, while filtering options allow for time, region, or type-based analyses. Extensive dashboards and metrics give valuable insights about the relations that reflect correlations between anomalies, confidence scores, and impact estimations. The dashboard has four parts: an anomaly trend line chart, a geographic distribution heat map, and persistent anomaly correlation alerts. This intuitive and accessible design enables most users to monitor and optimize 5G network performance effectively regardless of background.

Chapter 5

Simulation Setup

This chapter presents the necessary configurations, software, and system settings used in this study, along with the essential elements required for achieving and replicating the results. Then, a simulation is run to show the possible experiences of the multi-agent system in a disaster situation of a breakdown of 5G technology.

5.1 Simulation Environment

The simulation includes three main parts: the File Data Reporting System, a multi-agent system for root-cause analysis, and a web dashboard for visualization. These parts mimic the role of NWDAF EventSubscription and enable systematic anomaly detection and analysis.

The simulation does not have strict hardware requirements since it does not depend on a specific operating system. The only requirement for the NWDAF EventSubscription simulation is that it can run with the File Data Reporting System on both Windows and Linux platforms. The architecture is designed to operate in a Docker-enabled container, making it easily deployable and reproducible.

5.2 File Data Reporting System Setup

File Data Reporting System is an application used to make it easy to store, retrieve, and subscribe to notifications of network events. It follows TS 28.532 specification to provide structured administrative and management data reporting. It efficiently handles file storage and retrieval, supports event subscription capabilities, checks against the OpenAPI schema, and keeps the database consistent.

They are available to the public as links through which file notification subscriptions can be managed, existing subscribers can be deleted, metadata about accessible report files can be cleared and changed, reports can be submitted, file contents can be read, and files can be erased from the system. By facilitating smooth communication between agents and the file storage system, these endpoints guarantee adherence to NWDAF event subscription models.

The File Data Reporting System relies on dependencies that facilitate its operation. Incoming requests conform to the API schema.

After setting up, event-based anomaly reporting can be efficiently performed with a reproducible and scalable procedure. The procedure will strictly follow the multi-agent framework used to analyze events automatically. It will be designed to be integrated and consistent in the overall simulation framework based on containerized environments, consistent API definitions, and centralized logging.

5.3 File Generator Setup

The file generation mechanism simulates NWDAF EventSubscription notifications by generating structured event reports that align with 3GPP standards. The generator produces randomized event notifications adhering to the schema defined in TS 29.520, specifically for the `ABNORMAL_BEHAVIOUR` event type. These notifications are encapsulated within files and uploaded to the File Data Reporting System, which subscribing agents process.

Each generated event contains key parameters such as SUPI, timestamps, confidence levels, and additional contextual information. The generator organizes aberrant activity by mapping each anomaly type to a particular file category. To provide modular event management, anomalies like Unexpected UE Location, Too Frequent Service Access, and Suspicion of DDoS Attacks are categorized.

The script regularly creates and uploads event files at predetermined intervals using a periodic execution approach. To guarantee a realistic distribution of aberrant behaviors, the anomaly type for every file is chosen at random using weighted probability. Additionally, the generator populates extended details relevant to the specific anomaly, such as UE location data, network performance indicators, and unexpected flow characteristics.

To ensure compliance with the NWDAF subscription model, each event file is a JSON object containing structured event notifications. Metadata parameters, including event identifiers, reporting timestamps, and expiration conditions, are included in these messages. The generator dynamically uploads new event reports through HTTP POST requests to the File Data Reporting System.

This automated file production technique validates the multi-agent framework's ability to identify, examine, and correlate network events. It also allows for realistic simulation of network anomalies. The system is a good substitute for settings lacking direct NWDAF access because it guarantees compatibility with NWDAF event structures by following predefined schemas.

5.4 Multi-Agent System Deployment

The multi-agent system is responsible for processing event data retrieved from the File Data Reporting System and forwarding structured insights to the Hermes Agent. Each agent operates independently while subscribing to specific anomaly categories and filtering relevant event data.

The Physical Layer Inspector Agent processes Trace files to detect radio link failures. It listens for file notifications, retrieves the relevant data, filters anomalies related to unexpected radio link failures, and forwards structured insights to the Hermes Agent. The User Monitor Agent subscribes to Trace and Performance files, identifying user behavior-related anomalies such as Too Frequent Service Access and Unexpected UE Location. It ensures that only relevant anomalies are processed and passed to the Hermes Agent for further correlation. The Backend Advisor Agent monitors Analytics and Proprietary files to detect abnormalities like Unexpected Long Live Flows and Suspicion of DDoS Attacks. It processes network performance-related issues and forwards structured insights to Hermes for further analysis.

Every agent adheres to one standard process for event subscription and processing to maintain modularity and efficiency in detecting network anomalies. Agents move asynchronously, gathering event details, applying unique filtering criteria, and transmitting processed results to the Hermes Agent. The method is decentralized, thus allowing efficient event analysis while maintaining a scalable and adaptable system architecture.

5.5 Hermes Agent for Event Correlation

The Hermes Agent is the central analytics component responsible for aggregating anomaly data from multiple agents, identifying correlations, and structuring the results for further analysis. It utilizes a database for persistent storage and implements predefined correlation rules to determine relationships between network anomalies. The Hermes Agent uses predefined logic to identify dependencies, such as the connection between Too Frequent Service Access and Radio Link Failures or the effect of DDoS attacks on network performance.

The Hermes Agent utilizes a structured correlation framework to ensure the anomalies are examined in the context of the more extensive network rather than handled separately. Geospatial analysis assesses time proximity, spatial linkages, and shared UE identifiers. The system calculates the proximity between reported UE locations using geospatial distance measurements, allowing it to detect anomalies occurring in close physical proximity. The Hermes Agent increases the correlation confidence if two anomalies occur within a predefined distance threshold.

Examining the timestamps of event occurrences is another way to determine temporal linkage. The approach increases the probability that anomalies are related if they occur within a given timeframe and have comparable characteristics, making it possible to classify events more accurately and better understand possible network cascade failures.

The results from the correlation process refine anomaly classification, enhance event reporting accuracy, and improve the root cause analysis workflow. By using correlation ratings and prioritizing strongly correlated anomalies, the approach assists network operators in concentrating on the most critical network problems. This technology enhances overall observability and resilience by helping to test automated detection tactics and providing operators with better insights into 5G network faults.

5.6 Visualization Web Dashboard Setup

The visualization online dashboard provides real-time insights into correlation findings, anomaly detections, and event notifications. It effectively allows people to examine unusual network activity through interactive charts, tables, and maps. The dashboard is implemented using React.js and connected to the backend services via REST APIs.

The dashboard has several parts, each with a distinct function. The main access point is the Event Dashboard, which compiles different insights about network anomalies. Its Geographical Insights module maps event locations so users may see anomalies spatially and identify grouping trends. Network operators can swiftly discover common network anomalies using the Abnormal Behaviors module, which summarizes identified anomalies grouped by Exception ID.

With the Timestamp Events module, users can examine trends over time by visualizing the temporal distribution of event notifications. This visualization element is crucial for enhancing the overall effectiveness of the NWDAF EventSubscription simulation and turning raw event data into insightful knowledge.

Each component fetches data dynamically from the backend services. The Abnormal Behaviors module retrieves structured event data from the Hermes Agent, displaying SUPIs, confidence levels, and additional measurement information. The Geographical Insights module interacts with the geospatial correlation system to retrieve events that include latitude and longitude information for use as map markers. Using the Correlation Insights module, users can visually investigate root-cause relationships by retrieving associated event pairs and their corresponding correlation scores.

The dashboard also features an interactive navigation menu that allows users to limit data visualization by event severity, time range, and Exception ID and navigate between other event categories. By combining these modules into a single dashboard, network operators may track, identify, and evaluate 5G network irregularities in real-time. The effectiveness of the NWDAF EventSubscription simulation depends on its visualization component, which transforms raw event data into insightful knowledge.

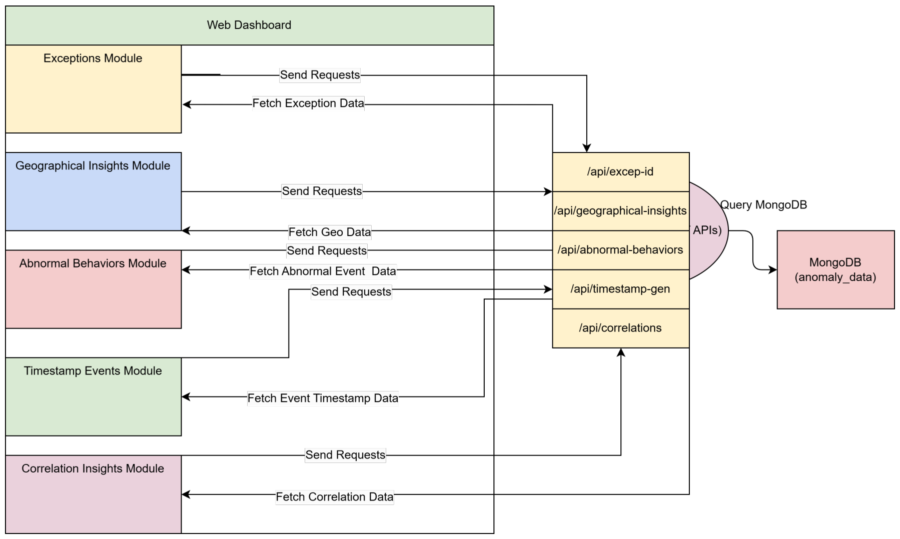


Figure 5.1: Illustration of the Visualization Dashboard

5.7 Data Flow and Processing Workflow

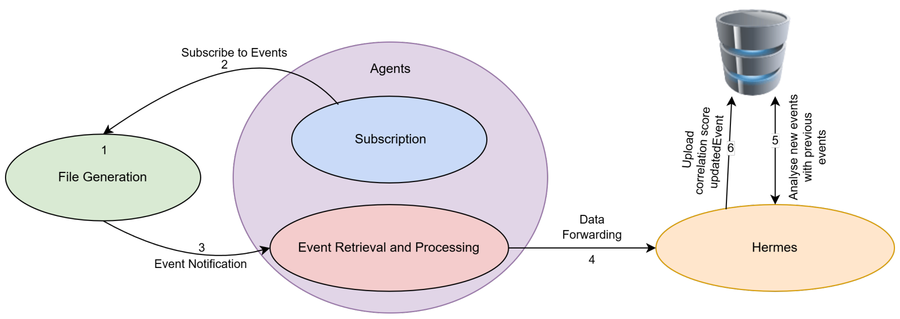

Subscribe to Events
2
Agents
Subscription
1
File Generation
3
Event Notification
Event Retrieval and Processing
Data Forwarding
4
Hermes
Upload correlation score updatedEvent
6
5
Analyse new events with previous events

Figure 5.2: The flow of data

File Generation: The File Data Reporting System generates NWDAF-compatible event notifications and categorizes them based on anomaly type.

Agent Subscription: Agents subscribe to specific file types and receive notifications when new anomalies are reported.

Event Retrieval and Processing: Agents fetch relevant files, extract meaningful insights, and format the data for analysis.

Data Forwarding: Processed anomaly data is transmitted to the Hermes Agent for aggregation and correlation.

Correlation and Analysis: Hermes applies predefined rules to detect relationships between anomalies and generates refined insights.

5.8 Conclusion

An organized and complete environment for simulating the NWDAF EventSubscription functionality is provided by this setup. To improve anomaly identification, root cause analysis, and interactive network event monitoring, a modular multi-agent architecture, a file-based event reporting system, and a real-time visualization dashboard are integrated. Insightful visualization, scalable agent-based processing, and automated event production ensure that network operators and researchers may efficiently evaluate anomalous activity in private 5G networks. The effectiveness of the suggested strategy will be examined in the upcoming chapters, and this setting lays the foundation for doing so.

Chapter 6

Evaluation

This chapter evaluates the performance of the proposed NWDAF EventSubscription simulation system, assessing its ability to detect, categorize, and correlate network anomalies in private 5G networks. The evaluation focuses on anomaly detection accuracy, correlation effectiveness, system performance, and scalability. The gathered data are examined to confirm that the system can detect anomalous network activity, measure its effects, and offer insightful information for root cause studies.

6.1 Evaluation Metrics and Methodology

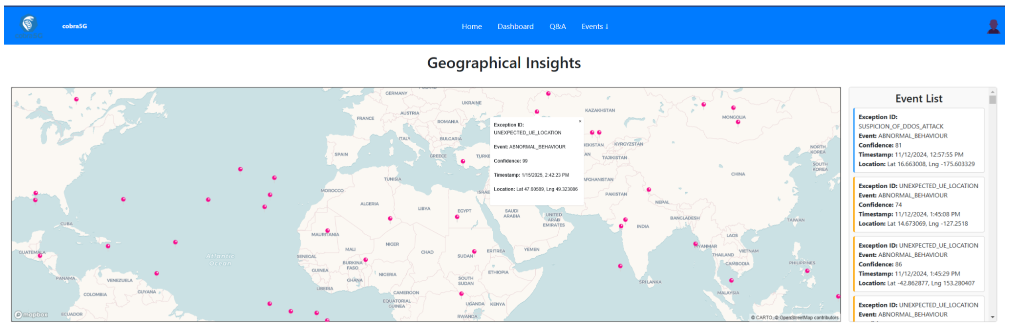
To assess the system's efficiency, several evaluation metrics were used, including:

- **Detection Accuracy**: The percentage of correctly identified anomalies compared to the ground truth.
- **Correlation Effectiveness**: The ability of the Hermes Agent to correctly link related anomalies.
- **Processing Time**: The time taken to process anomaly events from file generation to correlation analysis.
- **Scalability**: The system can handle increasing anomaly events and network load.

The evaluation of anomaly data was collected from the File Data Reporting System, which was processed by the multi-agent framework and aggregated by the Hermes Agent.

6.2 Anomaly Detection Performance

The system was tested on anomalies, including Unexpected UE Location, Radio Link Failures, Too Frequent Service Access, and Suspicion of DDoS Attacks. The evaluation revealed that the detection accuracy varied depending on the type of anomaly. Clear, unambiguous events like DDoS attacks displayed great detection confidence. Strong anomaly detection depends mainly on the detection model's capacity to differentiate between normal and abnormal behaviors.



Geographical Insights

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR
Confidence: 99
Timestamp: 1/15/2025, 2:42:23 PM
Location: Lat 47.60589, Lng 49.323086

Event List

Exception ID: SUSPICION_OF_DDOS_ATTACK
Event: ABNORMAL_BEHAVIOUR
Confidence: 81
Timestamp: 11/12/2024, 12:57:55 PM
Location: Lat 16.663008, Lng -175.603329

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR
Confidence: 74
Timestamp: 11/12/2024, 1:45:08 PM
Location: Lat 14.673069, Lng -127.2518

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR
Confidence: 86
Timestamp: 11/12/2024, 1:45:29 PM
Location: Lat -42.862877, Lng 153.280407

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR

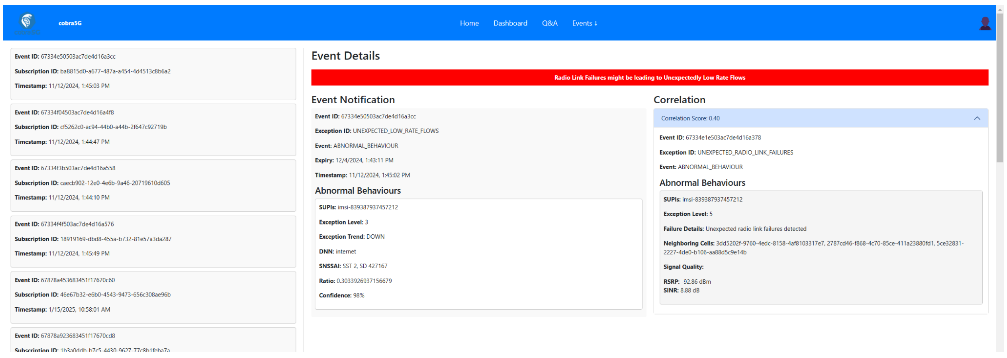
Figure 6.1: Geographical distribution of detected anomalies.

6.3 Correlation Analysis

The ability of the Hermes Agent to correlate several anomaly instances was assessed. Relationships such as radio link failures resulting in low rate flows and the effect of DDoS attacks on service availability were the main focus of the investigation. Weighted scoring is used in the correlation process, which considers several variables:

- **UE Identifier Matching**: Events sharing common UE IDs receive higher correlation scores.
- **Time Proximity**: Events occurring within a defined temporal threshold are more likely to be correlated.
- **Geospatial Analysis**: Events occurring within a certain distance are assigned a proximity-based correlation score.
- **Additional Metrics**: Signal quality, confidence levels, and exception severity levels contribute to the correlation evaluation.

The geospatial correlation mechanism uses latitude and longitude data to determine the physical distance between event sites. Two anomalies within a given distance threshold result in a proximity score assigned and included in the ultimate correlation score.



Event ID: 67334e50503ac7de4d16a3cc
Subscription ID: ba8815d0-a677-487a-a454-4d4513c8b6a2
Timestamp: 11/12/2024, 1:45:03 PM

Event ID: 67334f04503ac7de4d16a4f8
Subscription ID: cf5262c0-ac94-44b0-a44b-2f647c92719b
Timestamp: 11/12/2024, 1:44:47 PM

Event ID: 67334f3b503ac7de4d16a558
Subscription ID: caecb902-12e0-4e6b-9a46-20719610d605
Timestamp: 11/12/2024, 1:44:10 PM

Event ID: 67334f4f503ac7de4d16a576
Subscription ID: 18919169-dbd8-455a-b732-81e57a3da287
Timestamp: 11/12/2024, 1:45:49 PM

Event ID: 67878a453683451f17670c60
Subscription ID: 46e67b32-e6b0-4543-9473-656c308ae96b
Timestamp: 1/15/2025, 10:58:01 AM

Event ID: 67878a923683451f17670cd8
Subscription ID: 1b3a0ddb-b7c5-4430-9627-77c8b1feba7a

Event Details

Radio Link Failures might be leading to Unexpectedly Low Rate Flows

Event Notification

Event ID: 67334e50503ac7de4d16a3cc
Exception ID: UNEXPECTED_LOW_RATE_FLOWS
Event: ABNORMAL_BEHAVIOUR
Expiry: 12/4/2024, 1:43:11 PM
Timestamp: 11/12/2024, 1:45:02 PM

Abnormal Behaviours

SUPIs: imsi-839387937457212
Exception Level: 3
Exception Trend: DOWN
DNN: internet
SNSSAI: SST 2, SD 427167
Ratio: 0.3033926937156679
Confidence: 98%

Correlation

Correlation Score: 0.40

Event ID: 67334e1e503ac7de4d16a378
Exception ID: UNEXPECTED_RADIO_LINK_FAILURES
Event: ABNORMAL_BEHAVIOUR

Abnormal Behaviours

SUPIs: imsi-839387937457212
Exception Level: 5
Failure Details: Unexpected radio link failures detected
Neighboring Cells: 3dd5202f-9760-4edc-8158-4af8103317e7, 2787cd46-f868-4c70-85ce-411a23880fd1, 5ce32831-2227-4de0-b106-aa88d5c9e14b
Signal Quality:
RSRP: -92.86 dBm
SINR: 8.88 dB

Figure 6.2: Correlation analysis of network anomalies.

Determining how long it takes for events to cross helps the algorithm consider time-based correlation. Variations in signal strength and event intensity still improve correlation accuracy further. Even more improvement in correlation accuracy could be obtained with dynamic threshold adaptation.

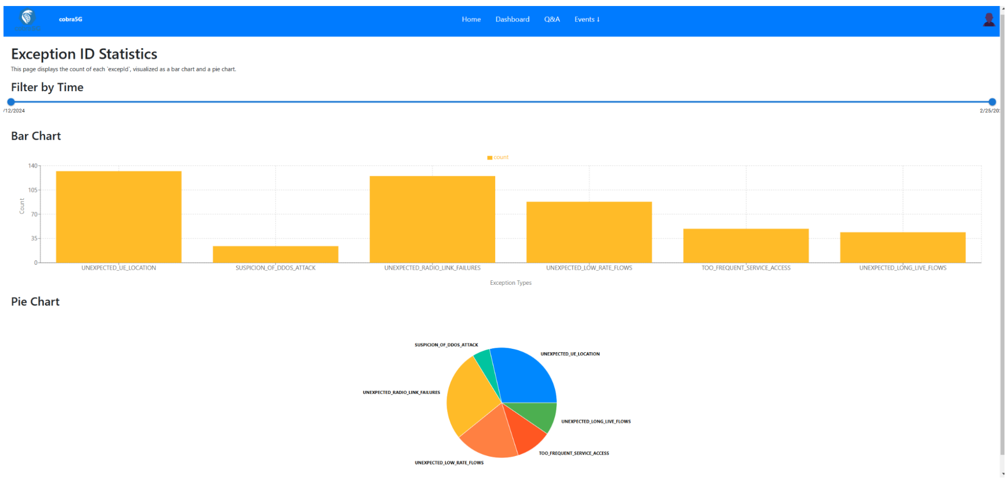

cobra5G
Home
Dashboard
Q&A
Events ↓
Exception ID Statistics
This page displays the count of each `excepId`, visualized as a bar chart and a pie chart.
Filter by Time
/12/2024
2/25/20
Bar Chart
count
140
105
70
35
0
Count
UNEXPECTED_UE_LOCATION
SUSPICION_OF_DDOS_ATTACK
UNEXPECTED_RADIO_LINK_FAILURES
UNEXPECTED_LOW_RATE_FLOWS
TOO_FREQUENT_SERVICE_ACCESS
UNEXPECTED_LONG_LIVE_FLOWS
Exception Types
Pie Chart
SUSPICION_OF_DDOS_ATTACK
UNEXPECTED_UE_LOCATION
UNEXPECTED_RADIO_LINK_FAILURES
UNEXPECTED_LONG_LIVE_FLOWS
TOO_FREQUENT_SERVICE_ACCESS
UNEXPECTED_LOW_RATE_FLOWS

Figure 6.3: Statistical distribution of detected exceptions.

6.4 System Performance and Scalability

Performance measurements were documented to gauge the system's reaction time to various event loads. The effect of higher event volume on system efficiency and the average processing time per anomalous event were assessed. The system showed effective event processing with low correlation and anomaly detection latency. Tests of scalability revealed that the system's performance remained constant even when processing one anomaly event per 0.5 seconds. The system's ability to manage this additional workload without experiencing noticeable slowdowns was verified by a stress test that produced two anomalies per second.

Correlation success rates differed depending on the level of randomness introduced in the data generation. Strict UE ID matching and timing limits caused most correlations to fail in a test where events were dispersed randomly across several places. This behavior was anticipated because randomized data lacks inherent links between abnormalities. In contrast, a second test in which all abnormalities happened in Berlin showed a considerable improvement in correlation efficacy. More successful event groupings were made possible by geographical proximity, and improved time proximity led to higher correlation scores. Even with this increase, inevitable correlation failures persisted because explicit UE ID matches were absent, indicating that improvements in fuzzy matching methods or adaptive correlation criteria may improve accuracy in practical applications.

6.5 Comparison with Other Approaches

Conventional log-based monitoring and ML-based anomaly detection against the multi-agent method are investigated. Unlike log-based approaches that depend on past data processing, the event-driven system reduces the time required for network operators to react. Compared to ML-based solutions, the structured event processing model avoids the need for extensive training datasets while maintaining high detection accuracy.

A relevant approach leveraging NWDAF for predictive analytics is the ML-based KPI prediction model proposed by Bayleyegn et al. [13]. Their approach predicts network traffic-related Key Performance Indicators (KPIs) by combining machine learning techniques with NWDAF. Proactive mitigation tactics are made possible by the system's AI models to predict possible network outages or congestion before they happen, contrasting with the event-driven system implemented in this research, which operates reactively by detecting anomalies as they arise.

Another comparative model is the anomaly detection approach discussed in Insight of Anomaly Detection with NWDAF in 5G [19]. Emphasizing statistical anomaly detection methods that classify network behavior aberrations using NWDAF's analytics framework, this method estimates expected network traffic. It detects performance anomalies using probabilistic approaches, including Gaussian-based classification.

Presenting a further advancement in anomaly detection in Junia PPGCC Master Thesis on 5G Anomaly Detection [14]. This research investigates deep learning-based anomaly classification models for 5G networks. Unlike rule-based or statistical methods, it uses convolutional and recurrent neural networks (CNNs and RNNs) to process temporal patterns in network behavior. Deep learning requires large labeled datasets for training and has substantial computational costs, even if it offers exceptional anomaly classification accuracy. By contrast, the event-driven system used in this work runs in real-time and calls no pre-trained models. A hybrid solution integrating deep learning for anomaly classification and event-driven detection for immediate response could improve detection capabilities in dynamic 5G environments.

Table 6.1: File Processing Statistics

Metric	**Value**
Total files generated	524
Files successfully processed (positives)	478
Failed files	46
Success rate (%)	$\frac{478}{524} \times 100 = 91.22\%$
Failure rate (%)	$\frac{46}{524} \times 100 = 8.78\%$
Average processing time per file (seconds)	0.07
Total processing time for positives (seconds)	$478 \times 0.07 = 33.46$
Delay between file generation (seconds)	1.00
Cumulative file generation time (seconds)	$524 \times 1 = 524$
Average throughput (files per second)	1

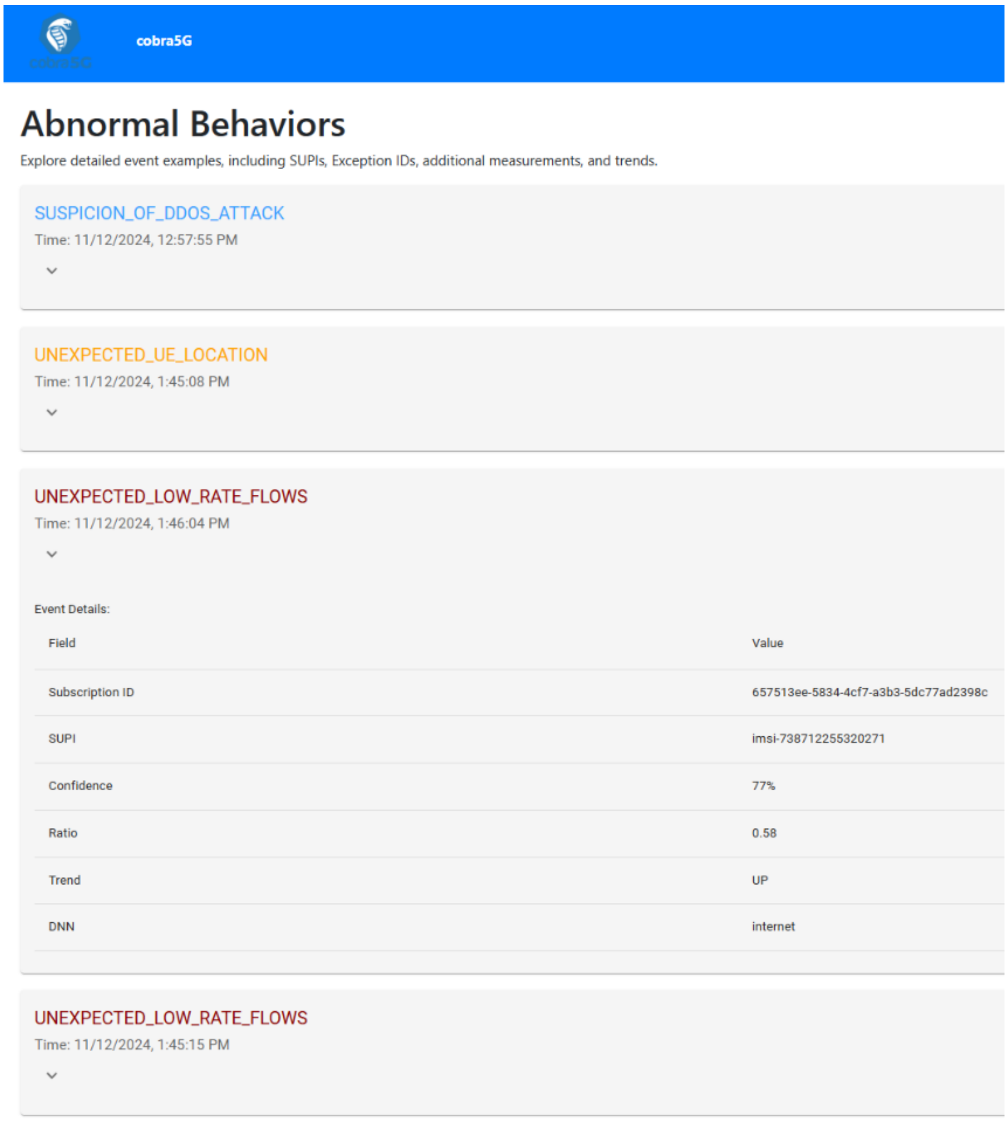
Table 6.2: File Processing Statistics

Metric	**Value**
Total files generated	900
Files successfully processed (positives)	886
Failed files	14
Success rate (%)	$\frac{886}{900} \times 100 = 98.44\%$
Failure rate (%)	$\frac{14}{900} \times 100 = 1.56\%$
Average processing time per file (seconds)	0.01
Total processing time for positives (seconds)	$886 \times 0.01 = 8.86$
Delay between file generation (seconds)	15.00
Cumulative file generation time (seconds)	$900 \times 15 = 13,500$
Average throughput (files per second)	0.06

Making a final comparison with the Real-Time Monitoring of 5G Networks: An NWDAF and ML-Based KPI Prediction Approach [13]. This approach combines NWDAF-driven analytics with machine learning to establish an end-to-end monitoring framework for 5G networks. The proposed design uses NWDAF as a centralized intelligence node that constantly evaluates network conditions and includes predictive modeling for network performance. Though this provides interesting material, it depends on high-volume data processing and persistent learning models, which can be computationally expensive. This work presents an event-driven paradigm emphasizing rapid responses to anomalies, offering a lightweight substitute. It performs best in a real-time setting where timely interaction takes the front stage. A critical aspect of whether or not an issue can be identified and solved in advance may be enabled by linking predictive control into the event-driven system.

6.6 Summary

The evaluation shows that the proposed NWDAF EventSubscription simulation faithfully reproduces actual anomaly detection and correlation systems. The multi-agent system offers real-time analysis, ordered event processing, and a scalable architecture fit for private 5G networks. The results validate the feasibility of an event-driven approach for anomaly detection, guiding future advancements in adaptive correlation models, improved exception handling, and artificial intelligence-assisted anomaly classification. These results highlight the possibilities of event-driven anomaly detection in private 5G networks, opening the path for future developments in intelligent network management.


cobra5G

Abnormal Behaviors

Explore detailed event examples, including SUPIs, Exception IDs, additional measurements, and trends.

SUSPICION_OF_DDOS_ATTACK

Time: 11/12/2024, 12:57:55 PM

UNEXPECTED_UE_LOCATION

Time: 11/12/2024, 1:45:08 PM

UNEXPECTED_LOW_RATE_FLOWS

Time: 11/12/2024, 1:46:04 PM

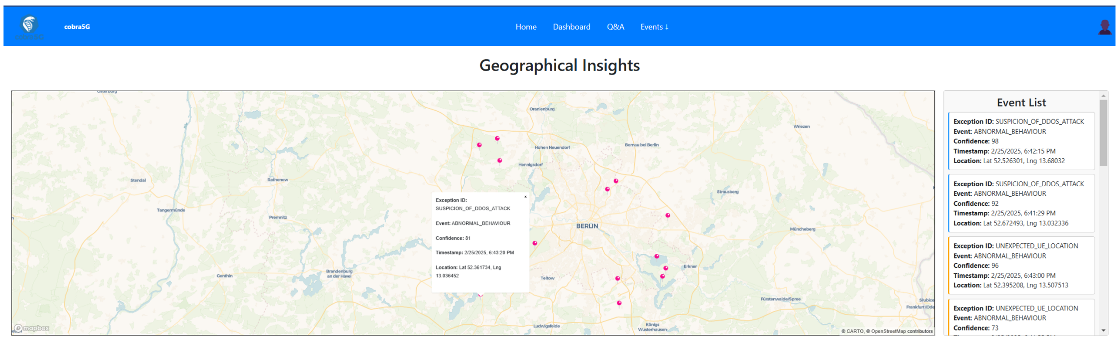
Event Details:

Field	Value
Subscription ID	657513ee-5834-4cf7-a3b3-5dc77ad2398c
SUPI	imsi-738712255320271
Confidence	77%
Ratio	0.58
Trend	UP
DNN	internet

UNEXPECTED_LOW_RATE_FLOWS

Time: 11/12/2024, 1:45:15 PM

Figure 6.4: Detailed abnormal behavior records with event attributes.



Geographical Insights

Exception ID: SUSPICION_OF_DDOS_ATTACK

Event: ABNORMAL_BEHAVIOUR

Confidence: 81

Timestamp: 2/25/2025, 6:43:20 PM

Location: Lat 52.361734, Lng 13.036452

Event List

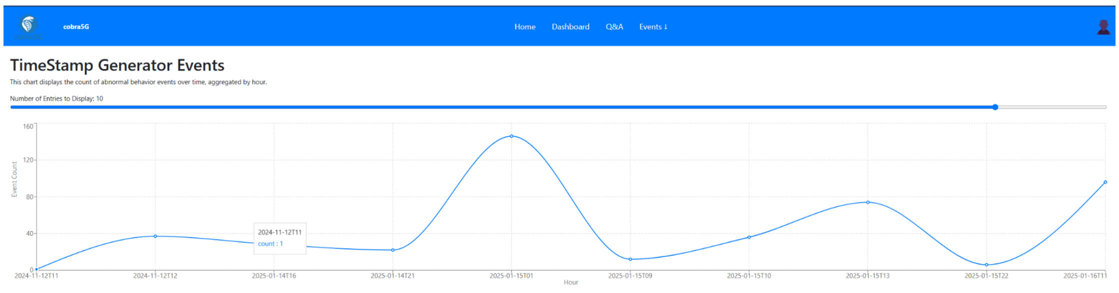
Exception ID: SUSPICION_OF_DDOS_ATTACK
Event: ABNORMAL_BEHAVIOUR
Confidence: 98
Timestamp: 2/25/2025, 6:42:15 PM
Location: Lat 52.526301, Lng 13.68032

Exception ID: SUSPICION_OF_DDOS_ATTACK
Event: ABNORMAL_BEHAVIOUR
Confidence: 92
Timestamp: 2/25/2025, 6:41:29 PM
Location: Lat 52.672493, Lng 13.032336

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR
Confidence: 96
Timestamp: 2/25/2025, 6:43:00 PM
Location: Lat 52.395208, Lng 13.507513

Exception ID: UNEXPECTED_UE_LOCATION
Event: ABNORMAL_BEHAVIOUR
Confidence: 73

Figure 6.5: Geographical distribution of detected anomalies for the 2nd Test located in Berlin.



TimeStamp Generator Events

This chart displays the count of abnormal behavior events over time, aggregated by hour.

Number of Entries to Display: 10

Figure 6.6: Event occurrence trends over time.

Chapter 7

Conclusion and Future Work

7.1 Summary

Chapter 2 introduced the core architecture of 5G systems, which highlights some of the most significant network functions, such as AMF, SMF, UPF, PCF, UDM, NRF, and NEF. Chapter 2 also described NWDAF's core role in 5G architecture. NWDAF's data analytics functions, primarily its event subscription and notification features, were elaborately described, highlighting their significance in anomaly detection and real-time network optimization.

Chapter 3 outlines the methodological framework for the multi-agent system. It describes the system architecture comprising specialized agents, including the Physical Layer Inspector Agent for detecting physical anomalies, the Strategic Advisor Agent for backend anomalies like DDoS attacks, and the User Anomaly Agent for unusual user behavior. All agents report anomalies to the Hermes Agent, which aggregates and correlates event data to provide actionable insights.

Chapter 4 detailed the implementation of the multi-agent framework using asynchronous Python programming and containerization with Docker. The File Data Reporting System was developed to simulate NWDAF EventSubscription due to the absence of an operational NWDAF. This simulation allowed structured anomaly detection and correlation in adherence to 3GPP standards, supported by tools such as FastAPI, MongoDB, and React.js for data processing, storage, and visualization.

Chapter 5 described the simulation environment and deployment processes. It outlined the hardware and software setup, the File Data Reporting System configuration, and the integration of the multi-agent framework. The methods used by the Hermes Agent to correlate anomalies in real time using geospatial and temporal analysis were covered. Furthermore, the visualization dashboard setup was described, showing how interactive interfaces provide insightful views of network anomalies.

Chapter 6 validated the system's effectiveness by measuring anomaly detection accuracy, correlation capability, and system scalability. The test confirmed event correlation and anomaly detection performance, guaranteeing the system's ability to efficiently detect network anomalies, correlate similar events, and maintain performance with increased network loading. The result indicated viable real-world applications in enhancing network monitoring and management.

7.2 Conclusion

This thesis has explored and implemented a multi-agent framework designed to enhance the capabilities of NWDAF EventSubscription within 5G networks. The results demonstrate remarkable improvement in real-time anomaly detection and correlation compared to standard network monitoring techniques.

The multi-agent platform could efficiently maximize the useability and accessibility of NWDAF by including specialized agent experts in detecting specific network anomalies, thus dramatically minimizing the complexity of anomaly data analysis for network administrators. Central to this framework is the Hermes Agent, which has successfully provided a dynamic correlation of anomalies against temporal, geographical, and user-specific dimensions. This agent-based approach significantly reduces latency for anomaly detection and root cause determination and, hence, is particularly well-suited for operational production.

The outcome points to the efficiency of systematic anomaly management and near-real-time correlation. By emulating NWDAF capability using a File Data Reporting system, the design exceeded the limitations of dealing with no working NWDAF. This was an in-real-life proof point of what NWDAF analytics could look like and provided an understanding of how such a system would perform under real-life circumstances.

Among the strengths emphasized are the established modularity and scalability of the developed multi-agent system. It could efficiently process high event loads without declining performance and demonstrated strong scalability potential. Furthermore, the visualization dashboard provided real-time and intuitive insights, making the complex network data more usable and accessible to users of varying technical know-how.

However, the design was also revealed to have some disadvantages. The efficiency and performance of the anomaly correlation are greatly influenced by pre-configured correlation rules, which need to be adjusted manually to deal appropriately with diverse network environments. In addition, the simulation-based design, although easy, might not exactly represent all realistic complexities and unexpected behaviors of a real 5G network in operation.

This paper presents a robust, scalable multi-agent solution that significantly enhances NWDAF's correlation and anomaly detection capability. It performed well in accessibility, responsiveness, and operational efficiency.

7.3 Future Work

Some areas have been identified and will be worked on in the future. First, the multi-agent architecture can be enhanced immensely by using sophisticated machine learning models to increase anomaly detection accuracy. In particular, using advanced machine learning or AI-based methods in agents like the Physical Layer Inspector and Strategic Advisor can make their detection more precise and responsive to dynamic network behavior.

Secondly, in later versions, correlation recognition based on deep learning may be incorporated within the Hermes agent to enhance the accuracy and robustness of anomaly correlation, particularly for more complex and dynamic network environments.

Another promising path would be adding an agent who can explain advanced network data and anomalies in simple, easy-to-grasp words. Leveraging the latest Natural Language Processing (NLP) technologies, such as large language models (LLMs), could bring the analytics output significantly closer to users with limited technical expertise and enable them to understand and take necessary action.

Finally, extending the current multi-agent framework to cover broader and evolving analytical use cases will ensure that the solution remains flexible and scalable, adapting effectively to the continuously evolving challenges and requirements of 5G network operations.

Bibliography

[1] Technical specification group core network and terminals; 5g system; network data analytics function (nwdaf). Technical Report TS 29.520, 3GPP, 2022. Defines the structure of NWDAF, including event subscriptions, analytics processing, and interaction with other network functions.

[2] Technical specification group core network and terminals; 5g system; procedures for the 5g system. Technical Report TS 29.501, 3GPP, 2022. Provides guidelines on NWDAF's integration with SBA and interworking with other network components.

[3] Technical specification group services and system aspects; management and orchestration; 5g system performance measurements. Technical Report TS 28.552, 3GPP, 2023. Defines key performance indicators (KPIs) for 5G networks, including real-time data collection and NWDAF integration with OAM.

[4] Technical specification group services and system aspects; study on enablers for network automation for 5g; phase 2. Technical Report TS 23.288, 3GPP, 2023. Describes how NWDAF integrates with network automation, focusing on event-driven analytics.

[5] Technical specification group services and system aspects; system architecture for the 5g system. Technical Report TS 23.502, 3GPP, 2023. Describes the interaction between NWDAF and network functions like AMF, SMF, and PCF, focusing on data collection mechanisms.

[6] 3GPP. 5G; Management and orchestration; Generic management services. Technical Specification (TS) 28.532, 3rd Generation Partnership Project (3GPP), August 2020. Release 16.

[7] 3GPP. 3rd generation partnership project; technical specification group services and system aspects; 5g system architecture. Technical Report TS 23.501, 2022. Covers 5G core architecture, including SBA and network functions.

[8] 3GPP. 3rd generation partnership project; technical specification group services and system aspects; user plane and control plane separation in 5g. Technical Report TS 23.503, 2023. Focuses on UPF, AMF, SMF, and control/user plane interactions.

[9] 3GPP. Ts 29520 nwdaf eventsubscription protocol. YAML File, 2023. Available for reference in NWDAF architecture implementation.

[10] 3rd Generation Partnership Project (3GPP). TS 29.502: 5G System; Session Management Services for 5G System (5GS). Technical Specification 29.502, 3GPP, 2023.

[11] 3rd Generation Partnership Project (3GPP). TS 29.503: 5G System; Unified Data Management Services for 5G System (5GS). Technical Specification 29.503, 3GPP, 2023.

[12] 3rd Generation Partnership Project (3GPP). TS 29.518: 5G System; Access and Mobility Management Services for 5G System (5GS). Technical Specification 29.518, 3GPP, 2023.

[13] Abebu Ademe Bayleyegn, Zaloa Fernández, and Fabrizio Granelli. Real-time monitoring of 5g networks: An nwdaf and ml based kpi prediction. In *NetSoft 2024 - 6th International Workshop on Performance Evaluation of Next Generation Virtualized Environments and Software-Defined Networks*, pages 31–36. IEEE, 2024.

[14] Júnia Maísa de Oliveira Pereira. A framework for 5g network data analytics function with emphasis on anomaly detection. Master's thesis, Federal University of Minas Gerais, 2024. Referenced in the Abstract, Introduction (Sections 1.1 and 1.2), and the Framework Design and Evaluation (Sections 4.2 and 5).

[15] Mario Antunes Diogo Gomes Rui L. Aguiar Dinis Canastro, Ricardo Rocha. Root cause analysis in 5g/6g networks. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 217–222. IEEE, 2021. Explores graph-based dependency analysis for RCA in 5G networks, emphasizing automation and scalability.

[16] Evangelos and Kafetzakis. Leveraging network data analytics function and machine learning for data collection, resource optimization, security and privacy in 6g networks. *IEEE Communications Standards Magazine*, 8(1):45–51, 2024.

[17] Harrison Mfula and Jukka K. Nurminen. Adaptive root cause analysis for self-healing in 5g networks. In *2017 International Conference on High Performance Computing & Simulation (HPCS)*, pages 136–142. IEEE, 2017. Discusses

Bayesian-based RCA for 5G networks and its role in automating fault detection and diagnosis.

[18] Jiwon Ock, Hyeon No, and Seongmin Kim. Exploring synthetic data generation for anomaly detection in the 5g nwdaf architecture. In *43rd International Conference on Distributed Computing Systems (ICDCS)*, 2023. Referenced in the Abstract and Introduction sections of the poster. Synthetic data generation and anomaly detection are detailed in Section III (Case Study).

[19] Yachao Yuan, Christian Gehrmann, Jakob Sternby, and Luis Barriga. Insight of anomaly detection with nwdaf in 5g. *IEEE*, 2022. Referenced in Section I (Introduction), Section II.A (NWDAF Standard), and Section II.B (Abnormal Behavior Related Analytics) of the paper.

Appendices

Appendix A: Abbreviations

3GPP	3rd Generation Partnership Project
5G	Fifth Generation Mobile Network
5GC	5G Core Network
5GS	5G System
ACL	Access Control List
AF	Application Function
AMF	Access and Mobility Management Function
AN	Access Network
API	Application Programming Interface
AR	Augmented Reality
AUSF	Authentication Server Function
CDN	Content Delivery Network
CU	Centralized Unit
DCCF	Data Collection Control Function
DDoS	Distributed Denial of Service
DN	Data Network
DNN	Data Network Name
DU	Distributed Unit
ETSI	European Telecommunications Standards Institute
eRSU	Extended Road Side Unit
gNB	gNodeB
IA	Information Assurance
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IO	Information Operation
IoT	Internet of Things
IDP	Intrusion Detection System
IPS	Intrusion Prevention System
KPI	Key Performance Indicator
MEC	Multi-access Edge Computing
MFA	Multi-factor Authentication
NEF	Network Exposure Function
NF	Network Function
NFV	Network Functions Virtualization
NGAP	Next-Generation Node Application Part
NG-RAN	New Generation Radio Access Network
NRF	NF Repository Function
NSSF	Network Slice Selection Function
NWDAF	Network Data Analytics Function
OAM	Operations, Administration, and Maintenance

Appendix B - File Generator Script

```python
<import requests
import random
import time
from faker import Faker
from datetime import datetime, timedelta, timezone
import json

# Initialize Faker instance
fake = Faker()

# API endpoint configuration
API_BASE_URL = 'http://localhost:8080/fileDataReportingMnS/v1'
FILES_ENDPOINT = f'{API_BASE_URL}/files'

# Anomaly cases with file type mappings
ANOMALY_FILE_TYPE_MAPPING = {
    'Unexpected_UE_Location': 'Trace',
    'Unexpected_Long_Live_Flows': 'Analytics',
    'Suspicion_of_DDoS_Attack': 'Proprietary',
    'Too_Frequent_Service_Access': 'Performance',
    'Unexpected_Radio_Link_Failures': 'Trace',
    'Unexpected_Low_Rate_Flows': 'Performance',
    'Unexpected_Large_Rate_Flow': 'Analytics'
}

# Compression and format options
FILE_COMPRESSIONS = ['zip']
FILE_FORMATS = ['JSON']

common_supis = [f"imsi-{fake.random_number(digits=15,␣fix_len=True)}" for _ in range(5)]
base_time = datetime.now(timezone.utc)

BERLIN_COORDS = {
    "lat_min": 52.3,  # Lower bound for latitude
    "lat_max": 52.7,  # Upper bound for latitude
    "lon_min": 13.0,  # Lower bound for longitude
    "lon_max": 13.7   # Upper bound for longitude
}
def generate_berlin_coordinates():
    return {
        "longitude": round(random.uniform(BERLIN_COORDS["lon_min"], BERLIN_COORDS["lon_max"]), 6),
        "latitude": round(random.uniform(BERLIN_COORDS["lat_min"], BERLIN_COORDS["lat_max"]), 6),
        "tac": fake.random_number(digits=5, fix_len=True)
    }
def create_anomaly_file():
    """
        Generates and sends a simulated anomaly file to the API endpoint. The file's content
        is based on random attributes and pre-defined anomaly types.

        The function:
        1. Selects a random anomaly type.
        2. Creates the corresponding event notification structure.
        3. Populates additional information based on the anomaly type.
        4. Sends the generated file as a POST request to the API.

        Returns:
            None
    """
    anomaly_type = random.choices(
        list(ANOMALY_FILE_TYPE_MAPPING.keys()),
        weights=[1, 1, 0.5, 1, 1, 0.5, 0.5]
    )[0]

    file_data_type = ANOMALY_FILE_TYPE_MAPPING[anomaly_type]
    file_ready_time = (base_time + timedelta(seconds=random.randint(0, 180))).isoformat(timespec='seconds')
    expiration_time = (base_time + timedelta(days=random.randint(1, 30))).isoformat(timespec='seconds')
    subscription_id = fake.uuid4()

    supi = random.choice([random.choice(common_supis), f"imsi-{fake.random_number(digits=15,␣fix_len=True)}"])

    # Base notification structure
    event_notification = {
        "subscriptionId": subscription_id,
        "eventNotifications": [
            {
                "event": "ABNORMAL_BEHAVIOUR",
                "expiry": expiration_time,
                "timeStampGen": file_ready_time,
                "abnorBehavrs": [
                    {
                        "supis": [supi],
                        "excep": {
                            "excepId": anomaly_type.upper().replace("␣", "_"),
                            "excepLevel": random.randint(1, 5),
                            "excepTrend": random.choice(["UP", "STABLE", "DOWN"])
                        },
                        "dnn": "internet",
                        "snssai": {
                            "sst": random.randint(1, 3),
                            "sd": "".join([str(random.randint(0, 9)) for _ in range(6)])
                        },
                        "ratio": random.uniform(0, 1),
                        "confidence": random.randint(70, 100)
                    }
                ]
            }
        ]
    }

    # Additional details based on anomaly type
    if anomaly_type == 'Unexpected_UE_Location':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"] = {
            "circums": [
                {
                    "freq": random.uniform(0, 1),
                    "tm": fake.iso8601(),
                    "locArea": generate_berlin_coordinates(),
                    "vol": random.randint(100, 1000)
                }
            ]
        }

    elif anomaly_type == 'Unexpected_Long_Live_Flows':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"
            ] = {
            "svcExps": [
                {
                    "svcExprc": {
                        "serviceQuality": random.uniform(0.1, 1.0),
                        "expectedPerformance": "Extended␣Duration"
                    },
                    "svcExprcVariance": random.uniform(0, 0.3),
                    "supis": [f"imsi-{fake.random_number(digits=15,␣fix_len=True)}"],
                    "snssai": {
                        "sst": random.randint(1, 3),
                        "sd": "".join([str(random.randint(0, 9)) for _ in range(6)])
                    },
                    "appId": fake.uuid4(),
                    "srvExpcType": "LONG_SESSION",
                    "ueLocs": [generate_berlin_coordinates()],
                    "upfInfo": {
                        "upfName": "ExtendedSessionUPF",
                        "upfIpAddress": fake.ipv4()
                    },
                    "dnai": "dna1-network",
                    "appServerInst": fake.domain_name(),
                    "confidence": random.randint(70, 100),
                    "dnn": "internet",
                    "networkArea": {
                        "areaCode": random.randint(1000, 9999),
                        "locationId": fake.uuid4()
                    },
                    "nsiId": fake.uuid4(),
                    "ratio": random.uniform(0.1, 1.0),
                    "ratFreq": {
                        "frequencyBand": random.choice(["700MHz", "2.6GHz"]),
                        "ratType": "LTE"
                    },
                    "pduSesInfo": {
                        "sessionId": fake.uuid4(),
                        "status": "ACTIVE"
                    }
                }
            ]
        }

    elif anomaly_type == 'Suspicion_of_DDoS_Attack':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"
            ] = {
            "ddosAttack": {
                "ipv4Addrs": [fake.ipv4()],
                "ipv6Addrs": [fake.ipv6()]
            },
            "circums": [
                {
                    "freq": random.uniform(0, 1),
                    "tm": fake.iso8601(),
                    "locArea": generate_berlin_coordinates(),
                    "vol": random.randint(100, 1000)
                }
            ]
        }

    elif anomaly_type == 'Too_Frequent_Service_Access':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"
            ] = {
            "svcExps": [
                {
                    "svcExprc": {
                        "serviceQuality": random.uniform(0.1, 1.0),
                        "expectedPerformance": "Normal"
                    },
                    "svcExprcVariance": random.uniform(0, 0.3),
                    "supis": [f"imsi-{fake.random_number(digits=15, fix_len=True)}"],
                    "snssai": {
                        "sst": random.randint(1, 3),
                        "sd": "".join([str(random.randint(0, 9)) for _ in range(6)])
                    },
                    "appId": fake.uuid4(),
                    "srvExpcType": "NORMAL_BANDWIDTH",
                    "ueLocs": [generate_berlin_coordinates()],
                    "upfInfo": {
                        "upfName": "ExampleUPF",
                        "upfIpAddress": fake.ipv4()
                    },
                    "dnai": "dnai-network",
                    "appServerInst": fake.domain_name(),
                    "confidence": random.randint(70, 100),
                    "dnn": "internet",
                    "networkArea": {
                        "areaCode": random.randint(1000, 9999),
                        "locationId": fake.uuid4()
                    },
                    "nsiId": fake.uuid4(),
                    "ratio": random.uniform(0.1, 1.0),
                    "ratFreq": {
                        "frequencyBand": random.choice(["700MHz", "2.6GHz"]),
                        "ratType": "LTE"
                    },
                    "pduSesInfo": {
                        "sessionId": fake.uuid4(),
                        "status": "ACTIVE"
                    }
                }
            ]
        }

    elif anomaly_type == 'Unexpected_Radio_Link_Failures':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"
            ] = {
            "nwPerfs": [
                {
                    "cellId": fake.uuid4(),
                    "failureCount": random.randint(1, 10),
                    "failureType": random.choice(['Signal loss', 'Failed handover']),
                    "signalQuality": {
                        "RSRP": f"{random.uniform(-120, -80):.2f} dBm",
                        "SINR": f"{random.uniform(-10, 20):.2f} dB"
                    },
                    "neighboringCells": [fake.uuid4() for _ in range(3)],
                    "networkSlice": random.choice(['SliceA', 'SliceB', 'SliceC']),
                    "details": "Unexpected radio link failures detected"
                }
            ]
        }

    elif anomaly_type == 'Unexpected_Low_Rate_Flow' or anomaly_type == 'Unexpected_Large_Rate_Flow':
        event_notification["eventNotifications"][0]["abnorBehavrs"][0]["addtMeasInfo"] = {
            "svcExps": [
                {
                    "svcExprc": {
                        "serviceQuality": random.uniform(0.1, 1.0),
                        "expectedPerformance": "Moderate"
                    },
                    "svcExprcVariance": random.uniform(0, 0.3),
                    "supis": [f"imsi-{fake.random_number(digits=15,␣fix_len=True)}"],
                    "snssai": {
                        "sst": random.randint(1, 3),
                        "sd": "".join([str(random.randint(0, 9)) for _ in range(6)])
                    },
                    "appId": fake.uuid4(),
                    "srvExpcType": "HIGH_BANDWIDTH",
                    "ueLocs": [generate_berlin_coordinates()],
                    "upfInfo": {
                        "upfName": "ExampleUPF",
                        "upfIpAddress": fake.ipv4()
                    },
                    "dnai": "dnai-network",
                    "appServerInst": fake.domain_name(),
                    "confidence": random.randint(70, 100),
                    "dnn": "internet",
                    "networkArea": {
                        "areaCode": random.randint(1000, 9999),
                        "locationId": fake.uuid4()
                    },
                    "nsiId": fake.uuid4(),
                    "ratio": random.uniform(0.1, 1.0),
                    "ratFreq": {
                        "frequencyBand": random.choice(["700MHz", "2.6GHz"]),
                        "ratType": "5G"
                    },
                    "pduSesInfo": {
                        "sessionId": fake.uuid4(),
                        "status": "ACTIVE"
                    }
                }
            ]
        }

    # Validate that 'timeStampGen' is present and in the correct format
    for notification in event_notification["eventNotifications"]:
        if "timeStampGen" not in notification or not isinstance(notification["timeStampGen"], str):
            print(f"ERROR:␣Missing␣or␣invalid␣'timeStampGen'␣for␣subscription␣ID:␣{subscription_id}")
        else:
            print(f"'timeStampGen'␣is␣valid:␣{notification['timeStampGen']}␣for␣subscription␣ID:␣{subscription_id}")
    # Construct the payload with the event notification
    payload = {
        'fileDataType': file_data_type,
        'fileContent': event_notification,
        'fileReadyTime': file_ready_time,
        'fileSize': random.randint(1024, 104857600),
        'fileExpirationTime': expiration_time,
        'fileCompression': random.choice(FILE_COMPRESSIONS),
        'fileFormat': random.choice(FILE_FORMATS)
    }

    print(json.dumps(payload, indent=2))
    try:
        response = requests.post(FILES_ENDPOINT, headers={'Content-Type': 'application/json'}, data=json.dumps(payload))
        if response.status_code == 201:
            file_id = response.json().get('fileId')
            print(f"Successfully␣created␣file␣for␣{anomaly_type}␣with␣ID:␣{file_id}␣at␣{datetime.now(timezone.utc)}")
        else:
            print(f"Failed␣to␣create␣file␣for␣{anomaly_type}.␣Status␣Code:␣{response.status_code}")
            print(f"Response:␣{response.text}")
    except Exception as e:
        print(f"An␣error␣occurred:␣{e}")


def main():
    """
        Main function to generate and send anomaly files at regular intervals.

        The function:
        1. Sets the number of files to generate.
        2. Calls the file generation function at specified intervals.

        Returns:
            None
    """
    num_files = 1000  # Adjust as needed
    time_interval = 15  # Adjust as needed

    for _ in range(num_files):
        create_anomaly_file()
        time.sleep(time_interval)


if __name__ == "__main__":
    main()
>
```

Appendix C - Hermes Agent Implementation

```python
<import asyncio
import logging
import requests
from fastapi import FastAPI, Request
from pymongo import MongoClient
import uvicorn
from datetime import datetime, timedelta, timezone
from geopy.distance import geodesic

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

FILE_DATA_REPORTING_URL = "http://localhost:8080/fileDataReportingMnS/v1/files"


class HermesAgent:
    """
        The HermesAgent class processes anomaly data, identifies correlations based on predefined rules,
        and stores the enriched data in MongoDB.
    """

    def __init__(self, host="127.0.0.1", port=8081, mongo_uri="mongodb://localhost:27017", db_name="anomaly_data"):
        """
                Initializes the HermesAgent instance, FastAPI app, MongoDB client, and correlation rules.

                Args:
                    host (str): The host address for the FastAPI application.
                    port (int): The port number for the FastAPI application.
                    mongo_uri (str): The MongoDB connection URI.
                    db_name (str): The MongoDB database name.
        """
        self.host = host
        self.port = port
        self.app = FastAPI()
        self.client = MongoClient(mongo_uri)
        self.db = self.client[db_name]
        self.collection = self.db["anomalies"]
        self.app.post("/receive_shared_data")(self.receive_shared_data)

        # Define correlation rules
        self.correlation_rules = [
            {
                "name": "Radio␣Link␣Failures␣might␣be␣leading␣to␣frequent␣service␣access␣attempts",
                "anomalies": ["UNEXPECTED_RADIO_LINK_FAILURES", "TOO_FREQUENT_SERVICE_ACCESS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 60}
                ]
            },
            {
                "name": "Unexpected␣UE␣Location␣could␣relate␣to␣Radio␣Link␣Failures",
                "anomalies": ["UNEXPECTED_UE_LOCATION", "UNEXPECTED_RADIO_LINK_FAILURES"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "location", "match": "proximity", "threshold_km": 5},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "A␣DDoS␣attack␣might␣be␣overloading␣network␣resources,␣leading␣to␣Radio␣Link␣Failures",
                "anomalies": ["UNEXPECTED_RADIO_LINK_FAILURES", "SUSPICION_OF_DDOS_ATTACK"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "Unexpected␣Large␣Rate␣Flows␣could␣be␣straining␣network␣resources␣and␣contribute␣to␣Radio␣Link␣Failures",
                "anomalies": ["UNEXPECTED_RADIO_LINK_FAILURES", "UNEXPECTED_LARGE_RATE_FLOWS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "Long-lived␣flows␣might␣be␣consuming␣network␣resources␣leading␣to␣Radio␣Link␣Failures",
                "anomalies": ["UNEXPECTED_RADIO_LINK_FAILURES", "UNEXPECTED_LONG_LIVE_FLOWS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "A␣DDoS␣attack␣might␣be␣overloading␣network␣resources,␣leading␣to␣Radio␣Link␣Failures",
                "anomalies": ["SUSPICION_OF_DDOS_ATTACK", "UNEXPECTED_RADIO_LINK_FAILURES"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 60}
                ]
            },
            {
                "name": "A␣DDoS␣attack␣might␣be␣causing␣Frequent␣Service␣Access␣Attempts",
                "anomalies": ["SUSPICION_OF_DDOS_ATTACK", "TOO_FREQUENT_SERVICE_ACCESS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time_interval", "match": "contains"}
                ]
            },
            {
                "name": "Frequent␣Service␣Access␣attempts␣might␣be␣causing␣Unexpected␣Large␣Rate␣Flows",
                "anomalies": ["UNEXPECTED_LARGE_RATE_FLOWS", "TOO_FREQUENT_SERVICE_ACCESS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time_interval", "match": "overlap"}
                ]
            },
            {
                "name": "Frequent␣service␣access␣might␣be␣causing␣Long-Lived␣Flows",
                "anomalies": ["UNEXPECTED_LONG_LIVE_FLOWS", "TOO_FREQUENT_SERVICE_ACCESS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time_interval", "match": "overlap"}
                ]
            },
            {
                "name": "Too␣Frequent␣Service␣Access␣might␣be␣due␣to␣Radio␣Link␣Failure",
                "anomalies": ["TOO_FREQUENT_SERVICE_ACCESS", "UNEXPECTED_RADIO_LINK_FAILURES"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 60}
                ]
            },
            {
                "name": "Too␣Frequent␣Service␣Access␣might␣be␣causing␣Unexpected␣Large␣Rate␣Flows",
                "anomalies": ["TOO_FREQUENT_SERVICE_ACCESS", "UNEXPECTED_LARGE_RATE_FLOWS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "Too␣Frequent␣Service␣Access␣might␣be␣resulting␣in␣Long-Live␣Flows",
                "anomalies": ["TOO_FREQUENT_SERVICE_ACCESS", "UNEXPECTED_LONG_LIVE_FLOWS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "Radio␣Link␣Failures␣might␣be␣leading␣to␣Unexpectedly␣Low␣Rate␣Flows",
                "anomalies": ["UNEXPECTED_LOW_RATE_FLOWS", "UNEXPECTED_RADIO_LINK_FAILURES"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 60}
                ]
            },
            {
                "name": "A␣DDoS␣attack␣might␣be␣leading␣to␣Low␣Rate␣Flows",
                "anomalies": ["UNEXPECTED_LOW_RATE_FLOWS", "SUSPICION_OF_DDOS_ATTACK"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            },
            {
                "name": "Large rate flows could temporarily be disrupting normal traffic, leading to unexpectedly Low Rate Flows",
                "anomalies": ["UNEXPECTED_LOW_RATE_FLOWS", "UNEXPECTED_LARGE_RATE_FLOWS"],
                "conditions": [
                    {"type": "ue_ids", "match": "intersection"},
                    {"type": "time", "match": "overlap", "threshold_minutes": 30}
                ]
            }
        ]

    def calculate_correlation_score(self, ue_ids_match, time_difference, time_threshold, rule_conditions, event,
                                    recent_event):
        """
        Calculates a correlation score based on the matching UE IDs, time difference,
        and rule-specific conditions.

        Args:
            ue_ids_match (bool): Whether UE IDs intersect.
            time_difference (timedelta): The difference between event and recent event timestamps.
            time_threshold (timedelta): The maximum allowable time difference for correlation.
            rule_conditions (list): The conditions defined in the correlation rule.
            event (dict): The current event being checked for correlation.
            recent_event (dict): The recent event being compared.

        Returns:
            float: The calculated correlation score.
        """
        # Initialize score components
        score = 0.0

        # Weight assignments
        weight_ue_ids = 0.3
        weight_time_proximity = 0.4
        weight_location_proximity = 0.3
        weight_additional_conditions = 0.1

        # print("\n--- Calculating Correlation Score ---")

        #        UE ID match contribution
        if ue_ids_match:
            score += weight_ue_ids
            # print(f"    UE ID Match: +{weight_ue_ids}")

        #        Time proximity contribution
        if time_difference <= time_threshold:
            time_proximity_score = 1 - (time_difference.total_seconds() / time_threshold.total_seconds())
            time_score = weight_time_proximity * time_proximity_score
            score += time_score
            # print(f"    Time Proximity: {time_proximity_score:.2f} * {weight_time_proximity} = +{time_score:.2f}")

        #        Rule-specific conditions
        for condition in rule_conditions:
            if condition["type"] == "location" and condition["match"] == "proximity":
                event_location = \
                event["eventNotifications"][0]["abnorBehavrs"][0].get("addtMeasInfo", {}).get("circums", [{}])[0].get(
                    "locArea", {})
                recent_location = \
                recent_event["eventNotifications"][0]["abnorBehavrs"][0].get("addtMeasInfo", {}).get("circums", [{}])[
                    0].get("locArea", {})

                if event_location and recent_location:
                    proximity_score = HermesAgent.calculate_proximity_score(event_location, recent_location,
                                                                            condition["threshold_km"])
                    loc_score = weight_location_proximity * proximity_score
                    score += loc_score
                    # print(
                    #     f"    Location Proximity: {proximity_score:.2f} * {weight_location_proximity} = +{loc_score:.2f}")

            if condition["type"] == "additional_metric":
                additional_metric_score = HermesAgent.calculate_additional_metric(event, recent_event, condition)
                add_score = weight_additional_conditions * additional_metric_score
                score += add_score
                # print(
                #     f"    Additional Metric: {additional_metric_score:.2f} * {weight_additional_conditions} = +{add_score:.2f}")

        # Ensure the score is within bounds (0 to 1)
        final_score = min(score, 1.0)
        # print(f"    Final Score: {final_score:.2f}")
        return final_score

    @staticmethod
    def calculate_proximity_score(location1, location2, threshold_km):
        """
        Calculate proximity score based on geographical locations.

        Args:
            location1 (dict): The first location with latitude and longitude.
            location2 (dict): The second location with latitude and longitude.
            threshold_km (float): Maximum distance in kilometers for full proximity score.

        Returns:
            float: A proximity score between 0 and 1.
        """
        coords1 = (location1.get("latitude"), location1.get("longitude"))
        coords2 = (location2.get("latitude"), location2.get("longitude"))

        distance_km = geodesic(coords1, coords2).kilometers
        return max(0.0, 1 - (distance_km / threshold_km))

    @staticmethod
    def calculate_additional_metric(event, recent_event, condition):
        """
        Calculates an additional metric score based on specific conditions such as signal quality,
        confidence, exception level, or ratio.

        Args:
            event (dict): The current event being checked.
```

```python
        recent_event (dict): The recent event being compared.
        condition (dict): Specific condition details.

    Returns:
        float: A score between 0 and 1.
    """
    score = 0.0

    try:
        if condition["type"] == "signal_quality":
            event_signal = event["eventNotifications"][0]["abnorBehavrs"][0].get(
                "addtMeasInfo", {}).get("nwPerfs", [{}])[0]
            recent_signal = recent_event["eventNotifications"][0]["abnorBehavrs"][0].get(
                "addtMeasInfo", {}).get("nwPerfs", [{}])[0]

            if event_signal and recent_signal:
                event_rsrp = float(event_signal.get("signalQuality", {}).get("RSRP", "-120 dBm").split()[0])
                recent_rsrp = float(recent_signal.get("signalQuality", {}).get("RSRP", "-120 dBm").split()[0])
                event_sinr = float(event_signal.get("signalQuality", {}).get("SINR", "0 dB").split()[0])
                recent_sinr = float(recent_signal.get("signalQuality", {}).get("SINR", "0 dB").split()[0])

                rsrp_diff = abs(event_rsrp - recent_rsrp)
                sinr_diff = abs(event_sinr - recent_sinr)

                score += max(0.0, 1 - rsrp_diff / 20) * 0.5
                score += max(0.0, 1 - sinr_diff / 20) * 0.5

        elif condition["type"] == "confidence":
            event_confidence = event["eventNotifications"][0]["abnorBehavrs"][0].get("confidence", 0)
            recent_confidence = recent_event["eventNotifications"][0]["abnorBehavrs"][0].get("confidence", 0)

            score += min(event_confidence, recent_confidence) / 100

        elif condition["type"] == "exception_level":
            event_level = event["eventNotifications"][0]["abnorBehavrs"][0]["excep"].get("excepLevel", 0)
            recent_level = recent_event["eventNotifications"][0]["abnorBehavrs"][0]["excep"].get("excepLevel", 0)

            level_diff = abs(event_level - recent_level)
            score += max(0, 1 - level_diff / 5)

        elif condition["type"] == "ratio":
            event_ratio = event["eventNotifications"][0]["abnorBehavrs"][0].get("ratio", 0)
            recent_ratio = recent_event["eventNotifications"][0]["abnorBehavrs"][0].get("ratio", 0)

            ratio_diff = abs(event_ratio - recent_ratio)
            score += max(0, 1 - ratio_diff)

    except Exception as e:
        print(f"Error calculating metric: {e}")

    return min(score, 1.0)

def check_correlation_rules(self, event, recent_events):
    """
    Checks correlations between the current event and recent events based on predefined rules.

    Args:
        event (dict): The current event being evaluated.
        recent_events (list): A list of recent events.

    Returns:
        list: A list of correlations matching the predefined rules.
    """
    correlations = []

    # Assume 'timeStampGen' is valid and in the correct format
    event_time_str = str(event['eventNotifications'][0]['timeStampGen'])
    event_time = datetime.fromisoformat(event_time_str).astimezone(timezone.utc)

    event_type = event['eventNotifications'][0]['abnorBehavrs'][0]['excep']['excepId']
    event_ue_ids = set(event['eventNotifications'][0]['abnorBehavrs'][0].get('supis', []))

    for rule in self.correlation_rules:
        if event_type in rule['anomalies']:
            for recent_event in recent_events:
                recent_event_time_str = str(recent_event['eventNotifications'][0]['timeStampGen'])
                recent_event_time = datetime.fromisoformat(recent_event_time_str).replace(tzinfo=timezone.utc)
                recent_event_type = recent_event['eventNotifications'][0]['abnorBehavrs'][0]['excep']['excepId']
                recent_event_ue_ids = set(recent_event['eventNotifications'][0]['abnorBehavrs'][0].get('supis', []))

                # Skip same-type correlations
                if event_type == recent_event_type:
                    continue

                # Calculate time difference
                time_difference = abs(event_time - recent_event_time)

                time_threshold = timedelta(minutes=float(
                    next((cond["threshold_minutes"] for cond in rule["conditions"] if cond["type"] == "time"), 60)
                    # Default to 60 minutes if missing
                ))

                # Check UE IDs and time conditions
                ue_ids_match = event_ue_ids & recent_event_ue_ids

                time_condition_met = any(
                    cond["type"] == "time" and cond["match"] == "overlap" and time_difference <= time_threshold
                    for cond in rule["conditions"]
                )




                # Calculate correlation score
                correlation_score = self.calculate_correlation_score(
                    ue_ids_match=bool(ue_ids_match),
                    time_difference=time_difference,
                    time_threshold=timedelta(minutes=float(
                        next((cond["threshold_minutes"] for cond in rule["conditions"] if cond["type"] == "time"),
                             60)  # Default to 60 minutes if missing
                    )),
                    rule_conditions=rule["conditions"],
                    event=event,
                    recent_event=recent_event
                )

                if recent_event_type in rule['anomalies'] and ue_ids_match and time_condition_met:
                    correlations.append({
                        "rule_name": rule["name"],
                        "correlated_event_id": recent_event["_id"],
                        "correlation_score": correlation_score
                    })
                # else:
                #     logger.info(f"No correlation: Rule '{rule['name']}', "
                #                 f"Event Type: {event_type}, Recent Type: {recent_event_type}, "
                #                 f"UE ID Match: {bool(ue_ids_match)}, Time Condition Met: {time_condition_met}")
    return correlations

def generate_file_content(self, event, correlation_data):
    """
    Generates the appropriate file structure for AMF, SMF, or UDM based on the event type.
    """
    event_notifications = event['eventNotifications'][0]
    abnormal_behavior = event_notifications['abnorBehavrs'][0]
    event_type = abnormal_behavior['excep']['excepId'].upper()  # Convert to uppercase
    supi = abnormal_behavior.get('supis', ["unknown"])[0]
    event_time = event_notifications['timeStampGen'].isoformat()

    # Extract dynamically instead of hardcoding
    location_info = abnormal_behavior.get('addtMeasInfo', {}).get('circums', [{}])[0]
    nw_perf_info = abnormal_behavior.get('addtMeasInfo', {}).get('nwPerfs', [{}])[0]

    tac = str(location_info.get('locArea', {}).get('tac', "000000"))
    plmn_id = location_info.get('locArea', {}).get('plmnId', {"mcc": "000", "mnc": "00"})
    eutra_cell_id = nw_perf_info.get('cellId', "000000")

    # Define mappings for AMF, SMF, and UDM
    AMF_EVENTS = {"UNEXPECTED_UE_LOCATION", "UNEXPECTED_RADIO_LINK_FAILURES"}
    SMF_EVENTS = {"UNEXPECTED_LONG_LIVE_FLOWS", "UNEXPECTED_LOW_RATE_FLOWS", "UNEXPECTED_LARGE_RATE_FLOW",
                  "SUSPICION_OF_DDOS_ATTACK"}
    UDM_EVENTS = {"TOO_FREQUENT_SERVICE_ACCESS"}

    # Generate file content based on event category
    if event_type in AMF_EVENTS:
        file_category = "Trace"
        file_component = "AMF"
        file_content = {
            "fileDataType": file_category,
            "fileComponent": file_component,
            "fileContent": {
                "eventId": "AMF_UE_MOBILITY",
                "supi": supi,
                "eventTime": event_time,
                "location": {
                    "tai": {
                        "plmnId": {"mcc": plmn_id.get("mcc", "000"), "mnc": plmn_id.get("mnc", "00")},
                        "tac": tac
                    },
                    "ecgi": {
                        "plmnId": {"mcc": plmn_id.get("mcc", "000"), "mnc": plmn_id.get("mnc", "00")},
                        "eutraCellId": eutra_cell_id
                    }
                },
                "correlation_data": correlation_data
            }
        }

    elif event_type in SMF_EVENTS:
        file_category = "Analytics"
        file_component = "SMF"
        file_content = {
            "fileDataType": file_category,
            "fileComponent": file_component,
            "fileContent": {
                "eventId": "SMF_SESSION_ANOMALY",
                "supi": supi,
                "pduSessionId": abnormal_behavior.get('pduSessionId', "10"),
                "dnn": abnormal_behavior.get('dnn', "internet"),
                "snssai": abnormal_behavior.get('snssai', {"sst": 1, "sd": "000000"}),
                "sessionType": "IPV4",
                "trafficAnomaly": {
                    "type": event_type,
                    "ratio": abnormal_behavior.get('ratio', 0),
                    "confidence": abnormal_behavior.get('confidence', 0)
                },
                "correlation_data": correlation_data
            }
        }

    elif event_type in UDM_EVENTS:
        file_category = "Proprietary"
        file_component = "UDM"
        file_content = {
            "fileDataType": file_category,
            "fileComponent": file_component,
            "fileContent": {
                "eventId": "UDM_UE_REACHABILITY",
                "supi": supi,
                "reachabilityStatus": "UNREACHABLE",
                "timestamp": event_time,
                "correlation_data": correlation_data
            }
        }

    else:
        return None  # No match found

    # Add metadata
    file_content.update({
        "fileReadyTime": datetime.now(timezone.utc).isoformat(),
        "fileExpirationTime": (datetime.now(timezone.utc) + timedelta(days=30)).
            isoformat(),
        "fileCompression": "zip",
        "fileFormat": "JSON"
    })

    return file_content


def upload_file_to_reporting_system(self, file_data):
    """Uploads the generated file to the File Data Reporting System."""
    headers = {"Content-Type": "application/json"}
    try:
        response = requests.post(FILE_DATA_REPORTING_URL, headers=headers, json=
            file_data)
        if response.status_code == 201:
            logger.info(f"Successfully uploaded file. File ID: {response.json().
                get('fileId')}")
        else:
            logger.error(f"Failed to upload file. Status Code: {response.
                status_code}, Response: {response.text}")
    except Exception as e:
        logger.error(f"Error uploading file: {e}")

async def receive_shared_data(self, request: Request):
    """
            Handles incoming event notifications, computes correlations, and
                stores the data.

            Args:
                request (Request): The incoming HTTP request containing event
                    data.

            Returns:
                dict: A response indicating the success of the operation.
    """

    # Processing Start Time
    processing_start_time = datetime.now(timezone.utc)
    logger.info(f"[PROCESSING_START] Hermes Received data at: {
        processing_start_time}")

    shared_data = await request.json()

    # Convert timeStampGen to timezone-aware datetime in UTC
    for notification in shared_data['eventNotifications']:
        notification['timeStampGen'] = datetime.fromisoformat(str(notification['
            timeStampGen'])).astimezone(
            timezone.utc)

    # Retrieve recent events to compare for correlation
    time_threshold = datetime.now(timezone.utc) - timedelta(minutes=60)

    recent_events = list(self.collection.find({
        "eventNotifications.timeStampGen": {
            "$gte": time_threshold
        }
        }))



        # Calculate correlations based on rules
        correlation_data = self.check_correlation_rules(shared_data, recent_events)

        # Add correlation data to the event document
        shared_data['correlation_data'] = correlation_data

        if correlation_data:
            file_data = self.generate_file_content(shared_data, correlation_data)
            self.upload_file_to_reporting_system(file_data)

        # Insert updated event data into MongoDB
        try:
            self.collection.insert_one(shared_data)
            # logger.info("Event stored with correlation data.")

            #  Processing End Time
            processing_end_time = datetime.now(timezone.utc)
            logger.info(f"[PROCESSING_END] Hermes Processing completed at: {
                processing_end_time}")

            #  Calculate and Log Total Processing Time
            processing_duration = (processing_end_time - processing_start_time).
                total_seconds()
            logger.info(f"[PROCESSING_TIME] Total Hermes processing time: {
                processing_duration:.2f} seconds")
        except Exception as e:
            logger.error(f"Failed to insert data: {e}")



        return {"status": "success"}

    async def start(self):
        server = uvicorn.Server(uvicorn.Config(self.app, host=self.host, port=self.
            port, log_level="info"))
        await server.serve()


# Usage example
if __name__ == "__main__":
    hermes_agent = HermesAgent(host="127.0.0.1", port=8081)


    async def standalone_start():
        await hermes_agent.start()


    try:
        asyncio.run(standalone_start())
    except Exception as ex:
        logger.error(f"An error occurred: {ex}")
>
```

Appendix D - PhysicalLayerInspector Agent Script

```
<import asyncio
import logging
from base_agent import BaseExaminerAgent
from fastapi import HTTPException, Request
from datetime import datetime, timedelta, timezone
import aiohttp

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


class PhysicalLayerInspectorAgent(BaseExaminerAgent):
    def __init__(self, host, port, hermes_agent_url):
        # Initialize BaseExaminerAgent with only the "Trace" file type
        super().__init__(host, port, ["Trace"], hermes_agent_url)
        logger.info(f"PhysicalLayerInspector␣initialized␣with␣URL:␣{self.host}:{self.port}")

    async def send_to_hermes(self, shared_data):
        async with aiohttp.ClientSession() as session:
            #logger.info(f"Sending data to Hermes: {shared_data}")
            async with session.post(self.hermes_agent_url, json=shared_data) as response:
                if response.status == 200:
                    logger.info("Data␣successfully␣sent␣to␣Hermes.")
                else:
                    logger.error(f"Failed␣to␣send␣data␣to␣Hermes.␣Status␣code:␣{response.status}")

    def create_file_handler(self, file_type):
        """Creates a specialized handler function for Trace file notifications."""
        async def handle_file_notification(request: Request):
            """Handles notifications for Trace files."""
            try:
                # Processing start time
                processing_start_time = datetime.now(timezone.utc)
                logger.info(f"[PROCESSING␣START]␣File␣␣Physical␣Agent␣notification␣received␣at:␣{processing_start_time}")


                notification = await request.json()
                #logger.info(f"Received Trace file notification: {notification}")

                file_info_list = notification.get("fileInfoList", [])
                if not file_info_list or not isinstance(file_info_list, list):
                    raise HTTPException(status_code=422, detail="Invalid␣fileInfoList␣structure")

                for file_info in file_info_list:
                    file_location = file_info.get("fileLocation")
                    if not file_location:
                        #logger.error("File location missing in file information.")
                        continue

                    # Fetch file details
                    file_details = await self.fetch_file_details(file_location)
                    if not file_details:
                        continue

                    # Process only relevant exceptions
                    excep_id = (
                        file_details.get("eventNotifications", [{}])[0]
                        .get("abnorBehavrs", [{}])[0]
                        .get("excep", {})
                        .get("excepId")
                    )
                    if excep_id != "UNEXPECTED_RADIO_LINK_FAILURES":
                        #logger.info("Ignoring non-UNEXPECTED_RADIO_LINK_FAILURES notification.")
                        return {"message": "Non-relevant Trace file ignored"}

                    # Prepare data to send to Hermes
                    shared_data = {key: value for key, value in file_details.items() if key != "fileInfo"}
                    await self.send_to_hermes(shared_data)

                    # Processing end time
                    processing_end_time = datetime.now(timezone.utc)
                    logger.info(f"[PROCESSING_END] Physical Agent Processing completed at: {processing_end_time}")

                    # Calculate and log processing duration
                    processing_duration = (processing_end_time - processing_start_time).total_seconds()
                    logger.info(f"[PROCESSING_TIME] Total Physical Agent processing time: {processing_duration:.2f} seconds")

                return {"message": "Trace notification processed"}

            except Exception as e:
                logger.error(f"Error processing Trace notification: {e}")
                raise HTTPException(status_code=422, detail="Invalid notification format")

        return handle_file_notification

# Running this as a standalone instance
if __name__ == "__main__":
    host = '127.0.0.16'
    port = 5558  # Use a unique port for the Trace agent
    hermes_agent_url = "http://localhost:8081/receive_shared_data"

    agent = PhysicalLayerInspectorAgent(host, port, hermes_agent_url)

    try:
        asyncio.run(agent.main())
    except Exception as ex:
        logger.error(f"An error occurred: {ex}")
>
```

Appendix E - Agent Orchestration Main Script

```python
<import asyncio
import logging
from backend_advisor_agent import BackendAdvisorAgent
from user_monitor_agent import UserMonitorAgent
from physical_layer_inspectorAgent_agent import PhysicalLayerInspectorAgent
from hermes_agent import HermesAgent  # Import HermesAgent



# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

async def start_agent(agent):
    """Helper function to start each agent."""
    await agent.main()

async def main():
    host = '172.24.176.1'
    hermes_agent_url = f'http://{host}:8081/receive_shared_data'  # Update Hermes URL for all agents

    # Instantiate each agent with a unique port
    backend_agent = BackendAdvisorAgent(host, 5555, hermes_agent_url)
    user_agent = UserMonitorAgent(host, 5556, hermes_agent_url)
    physical_layer_agent_agent = PhysicalLayerInspectorAgent(host, 5557, hermes_agent_url)

    # Instantiate Hermes agent
    hermes_agent = HermesAgent(host=host, port=8081)

    # Run all agents concurrently, including Hermes
    await asyncio.gather(
        start_agent(backend_agent),
        start_agent(user_agent),
        start_agent(physical_layer_agent_agent),
        hermes_agent.start()
    )

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except Exception as ex:
        logger.error(f"An␣error␣occurred:␣{ex}")
>
```

Appendix F: Example Agent Logs

Below are sample log excerpts collected during a system performance test.

```
00:00
INFO:user_monitor_agent:[PROCESSING TIME] Total User Monitor processing
INFO:     172.24.176.1:61774 - "POST /receive_shared_data HTTP/1.1" 200
```

```
INFO:     172.24.176.1:61772 - "POST /handle_performance_file_notificat
INFO:user_monitor_agent:[PROCESSING START] User Monitor File notificati
INFO:physical_layer_inspectorAgent_agent:[PROCESSING START] File  Physi
INFO:     172.24.176.1:61782 - "POST /handle_trace_file_notification HT
INFO:     172.24.176.1:61786 - "POST /receive_shared_data HTTP/1.1" 200
INFO:     172.24.176.1:61783 - "POST /handle_trace_file_notification HT
INFO:base_agent:Fetched file details: {'_id': '67d054351c0dd38f49b69739
INFO:base_agent:Fetched file details: {'_id': '67d054351c0dd38f49b69739
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
INFO:physical_layer_inspectorAgent_agent:Data successfully sent to Herm
INFO:physical_layer_inspectorAgent_agent:[PROCESSING END] Physical Ager
INFO:physical_layer_inspectorAgent_agent:[PROCESSING TIME] Total Physic
INFO:backend_advisor_agent:[PROCESSING START] Backend Advisor File noti
INFO:base_agent:Fetched file details: {'_id': '67d054441c0dd38f49b69743
INFO:backend_advisor_agent:Sending data to Hermes: {'_id': '67d054441c0
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
INFO:backend_advisor_agent:Data successfully sent to Hermes.
INFO:backend_advisor_agent:[PROCESSING END] Backend Advisor Processing
INFO:backend_advisor_agent:[PROCESSING TIME] Total  Backend Advisor pro
INFO:     172.24.176.1:61795 - "POST /receive_shared_data HTTP/1.1" 200
INFO:     172.24.176.1:61793 - "POST /handle_analytics_file_notificatio
INFO:user_monitor_agent:[PROCESSING START] User Monitor File notificati
INFO:base_agent:Fetched file details: {'_id': '67d054531c0dd38f49b6974b
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
INFO:user_monitor_agent:Data successfully sent to Hermes.
INFO:user_monitor_agent:[PROCESSING END] User Monitor Processing comple
INFO:user_monitor_agent:[PROCESSING TIME] Total User Monitor processing
INFO:     172.24.176.1:61817 - "POST /receive_shared_data HTTP/1.1" 200
INFO:     172.24.176.1:61815 - "POST /handle_performance_file_notificat
INFO:user_monitor_agent:[PROCESSING START] User Monitor File notificati
INFO:physical_layer_inspectorAgent_agent:[PROCESSING START] File  Physi
INFO:base_agent:Fetched file details: {'_id': '67d054621c0dd38f49b69753
INFO:base_agent:Fetched file details: {'_id': '67d054621c0dd38f49b69753
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
```

```
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
INFO:physical_layer_inspectorAgent_agent:Data successfully sent to Herm
INFO:physical_layer_inspectorAgent_agent:[PROCESSING END] Physical Ager
INFO:physical_layer_inspectorAgent_agent:[PROCESSING TIME] Total Physic
INFO:     172.24.176.1:61833 - "POST /handle_trace_file_notification HT
INFO:     172.24.176.1:61837 - "POST /receive_shared_data HTTP/1.1" 200
INFO:     172.24.176.1:61834 - "POST /handle_trace_file_notification HT
INFO:physical_layer_inspectorAgent_agent:[PROCESSING START] File  Physi
INFO:base_agent:Fetched file details: {'_id': '67d054711c0dd38f49b6975c
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
INFO:physical_layer_inspectorAgent_agent:Data successfully sent to Herm
INFO:physical_layer_inspectorAgent_agent:[PROCESSING END] Physical Ager
INFO:physical_layer_inspectorAgent_agent:[PROCESSING TIME] Total Physic
INFO:     172.24.176.1:61852 - "POST /receive_shared_data HTTP/1.1" 200
INFO:     172.24.176.1:61850 - "POST /handle_trace_file_notification HT
INFO:backend_advisor_agent:[PROCESSING START] Backend Advisor File noti
INFO:base_agent:Fetched file details: {'_id': '67d054811c0dd38f49b69765
INFO:backend_advisor_agent:Sending data to Hermes: {'_id': '67d054811c0
INFO:hermes_agent:[PROCESSING START] Hermes Received data at: 2025-03-1
INFO:hermes_agent:[PROCESSING END] Hermes Processing completed at: 2025
INFO:hermes_agent:[PROCESSING TIME] Total Hermes processing time: 0.00
```

These logs demonstrate the message flow between agents, notification handling, and processing durations for different types of file events in the simulated 5G NWDAF environment.

Appendix G: Docker and Compose Configuration

Listing 1: docker-compose.yml excerpt

```
version: '3.9'
services:
  file-reporter:
    build: .
    ports:
      - "8080:8080"
    volumes:
      - .:/app
    networks:
      - monitoring
  user-monitor-agent:
    image: my_agent_image
    ports:
      - "5556:5556"
    depends_on:
      - file-reporter
networks:
  monitoring:
    driver: bridge
```

Listing 2: Dockerfile snippet for agent build

```
# Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["python", "main.py"]
```