UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

Master of Science

Computer Science and Networking

Deployment and Testing of an Enterprise-Facing Private 5G Solution (Aether) in a VM- Based Environment

Supervisors:

Prof: Rosario G. Garroppo

Prof: Alessio Giorgetti

Candidate:

Hassan Shabir

ACADEMIC YEAR 2024/2025

Abstract

This thesis investigates the deployment and performance evaluation of *Aether*, an open-source, enterprise-grade private 5G platform developed by the Open Networking Foundation (ONF), within virtualized environments. The study explores the feasibility of deploying Aether in constrained setups such as single-node or dual-node virtual machines, which are common in academic and small enterprise scenarios. Two configurations were examined: a Quick Start deployment using `gNBsim` for validating core functionality, and a Full Aether deployment incorporating multiple blueprints, runtime control via ROC, and Quality of Service testing using `UERANSIM`. The results demonstrate Aether's modularity, resilience under load, and suitability for reproducible research and educational use. Additionally, the study identifies deployment challenges, including CPU and memory bottlenecks, multi-interface networking complexity, and telemetry limitations, while also contributing enhancements to monitoring and provisioning mechanisms. Overall, this work advances the understanding of scalable, cost-effective private 5G solutions and serves as a reference model for future academic experimentation and enterprise deployment.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Prof. Rosario G. Garroppo and Prof. Alessio Giorgetti, for their invaluable guidance, encouragement, and support throughout this research journey. Their expertise and insightful feedback have been instrumental in shaping the direction and quality of this work.

I am especially thankful to the Aether project team at the Open Networking Foundation (ONF) for providing an open and collaborative platform that made this work possible. In particular, I would like to sincerely thank Gabriel Arrobo and Suresh Marikkannu for their technical guidance, timely feedback, and continuous support during the deployment and testing phases of this project.

I am also grateful to my colleagues and friends for their encouragement, constructive discussions, and collaboration. Their support helped me stay motivated throughout the thesis process.

Finally, I thank my family for their unconditional love, patience, and unwavering belief in me. Their support has been the foundation of my academic journey.

Table of Contents

Table of Contents

List of Figures

List of Tables

Glossary of Terms

Term	Definition
5G	The fifth generation of wireless communication technologies, offering ultra-low latency, high throughput, and support for massive IoT deployments.
Private 5G	A standalone 5G network deployed by an organization for internal use, ensuring greater security, control, and customization.
Aether	An open-source platform by the Open Networking Foundation (ONF) for managing private 5G networks using cloud-native and modular components.
Blueprint	A reusable configuration package in Aether, defined using Ansible and Helm, that describes the deployment of network components.
SD-Core	Aether's cloud-native core network supporting 4G/5G functions like AMF, SMF, UPF, and UDM with microservices and APIs.
SD-RAN	Software-defined RAN aligned with O-RAN split 7.2x, including components like nRT-RIC, xApps, and simulated CU/DU nodes.
AMP	Aether Management Plane providing runtime control and monitoring through ROC and observability tools.
ROC	Runtime Operational Control subsystem enabling dynamic configuration of policies, slices, and devices using APIs and GUI.
OnRamp	Aether's deployment tool that uses Ansible and Helm to configure all components in VM or cloud setups.

Term	Definition
Container	A lightweight virtualized environment for packaging apps with dependencies, used for microservice deployment.
VM	A software-based emulation of a computer that hosts containerized services on general-purpose infrastructure.
NFV	Network Function Virtualization—virtualizing network services that were traditionally run on hardware appliances.
SDN	Software-Defined Networking—separates the network control plane from the data plane for programmability.
Kubernetes	A container orchestration system used to deploy and manage Aether microservices.
Helm	A Kubernetes package manager used in Aether for deploying containerized applications via Helm charts.
Ansible	An automation tool that defines infrastructure as code for setting up Aether environments.
gNB	The 5G base station that manages radio communications between UE and the core network.
gNBsim	A simulator that emulates gNB and UE behavior for testing Aether SD-Core functionality.
UERANSIM	A tool to simulate User Equipment and RAN for testing 5G Core deployments.
CU/DU	Centralized and Distributed Units that split base station functions between edge and cloud.
nRT-RIC	Near-Real-Time RAN Intelligent Controller hosting xApps to optimize radio network performance.
xApps	Modular control applications running on nRT-RIC to manage RAN policies and behaviors.
Prometheus	Monitoring system used to collect metrics from Aether services.
Grafana	Visualization tool used with Prometheus to create real-time dashboards.

Term	Definition
Rancher	Kubernetes management tool used for container lifecycle and cluster operations.
CI/CD	Continuous Integration/Deployment—automated workflows for testing and deploying applications.
Edge Cloud	A computing environment near data sources (e.g., factories or campuses) for low-latency processing.
Local Breakout	A deployment model where traffic is routed locally to reduce latency, rather than via centralized data centers.

Chapter 1

Introduction

The rapid advancement of 5G technology is transforming industries by enabling ultra-reliable, low-latency, and high-bandwidth communication. While public 5G networks are widely deployed, enterprises increasingly seek private 5G solutions for enhanced security, reliability, and customization. However, deploying such solutions in virtualized environments presents technical challenges, requiring robust evaluation and optimization. This thesis explores the deployment and testing of Aether, an open-source private 5G solution, within a virtualized testbed.

1.1 Motivation

Private 5G networks provide enterprises with dedicated connectivity, offering improved security, low-latency communication, and network customization. Unlike traditional Wi-Fi or public 5G, private 5G enables:

- **Secure and isolated network environments:** Ensuring that sensitive data remains within the organization's control.
- **Higher Quality of Service (QoS) for critical applications:** Allowing prioritization of mission-critical operations.
- **Greater control over network policies and resource allocation:** Facilitating tailored network configurations to meet specific business needs.

However, deploying private 5G solutions in virtualized infrastructure poses unique challenges, such as:

- **Performance overhead introduced by virtualization:** Virtualization can lead to increased latency and reduced throughput.
- **Complex network configuration and integration:** Integrating 5G network functions within a virtualized environment requires careful planning and execution.
- **Ensuring scalability while maintaining reliability:** As the network scales, maintaining consistent performance and reliability becomes challenging.

This study aims to explore these challenges by evaluating Aether, an enterprise-focused open-source private 5G solution.

1.2 Problem Statement

This research evaluates the feasibility and performance of deploying Aether—a modular, open-source private 5G platform—in a VM-based testbed. The study focuses on identifying deployment challenges, assessing network performance, and proposing optimizations to support Vm based implementations. Virtual machines (VMs) provide flexibility and scalability, but they also introduce complexities such as:

- **Performance bottlenecks due to virtualization overhead:** The additional layer of abstraction can impact network performance.
- **Difficulty in configuring 5G network functions in a multi-tenant environment:** Ensuring isolation and optimal performance for multiple tenants is challenging.
- **Challenges in integrating software-defined networking (SDN) with private 5G:** Harmonizing SDN principles with 5G architecture requires careful consideration.

This research evaluates the feasibility and performance of deploying Aether, an open-source private 5G platform, in a VM-based testbed. The study aims to identify key deployment challenges, assess network performance, and propose optimizations for enterprise-scale implementations.

1.3 Objectives

The primary objectives of this study are:

- **Deploy Aether on a VM-Based Infrastructure:** Set up a controlled testbed to evaluate Aether's deployment in a virtualized environment, including essential components like SD Core, Aether Run time Operation Control and Aether Monitor.
- **Test Core Functionalities and Performance Metrics:** Validate key 5G functionalities such as UE registration, PDU session establishment, bandwidth, and QoS using tools like UERANSIM and gNBSim.
- **Identify Challenges and Propose Solutions:** Investigate deployment bottlenecks (e.g., network configurations, VM setups) and suggest optimizations to improve scalability and reliability.
- **Support Teaching Activities:** Integrate the deployment process into educational activities, providing students with hands-on experience in configuring and troubleshooting private 5G networks.

These objectives aim to demonstrate the feasibility of deploying private 5G solutions in virtualized environments while contributing to both academic research and practical education.

1.4 Scope of the Study

This study focuses on the deployment and testing of Aether within a virtualized environment. The scope includes:

- **Setting up a VM-based testbed:** Utilizing virtualization technologies to emulate a realistic enterprise network environment, ensuring scalability and reproducibility for private 5G deployments.
- **Implementing Aether's core components:** Deploying essential network functions, including the Software-Defined Core Network (SD-Core), Aether Runtime Operational Control (ROC), and Aether Monitor, to facilitate comprehensive network management and monitoring.
- **Evaluating network performance:** Conducting careful testing to measure key performance indicators such as latency, throughput, packet loss, and system reliability under different workload conditions.

- **Addressing Deployment Challenges and Contributing to Platform Enhancements:** Identifying and resolving various issues encountered during deployment on virtual machines, implementing optimizations, and contributing valuable solutions to the Aether platform's codebase to improve its stability and functionality.

1.5 Thesis Organization

This thesis is organized as follows:

- **Chapter 2** provides an overview of private 5G networks, including virtualization, SDN, and the Aether platform.
- **Chapter 3** describes the methodology used to deploy Aether in a VM-based testbed and the testing framework adopted.
- **Chapter 4** details the implementation process, covering VM configurations, network setup, and 5G function deployment.
- **Chapter 5** presents the experimental results and discusses key findings, performance analysis, and observed challenges.
- **Chapter 6** concludes the study by summarizing contributions, limitations, and future directions for private 5G in virtualized environments.

Chapter 2

Background and Literature Review

2.1 Overview of Private 5G Networks

Private 5G networks are dedicated wireless communication systems designed to serve specific organizations, locations, or use cases. Unlike public 5G networks operated by telecom providers, private 5G networks are owned and managed by enterprises, offering greater control, customization, and security. These networks are gaining attraction due to their ability to meet the growing demands for high-speed, low-latency connectivity in various industries [5].

Private 5G networks have several defining characteristics. They can operate on licensed, unlicensed, or shared spectrum, ensuring interference-free and reliable communication. One key capability is network slicing, which allows the creation of multiple virtualized and isolated logical networks over the same physical infrastructure, each optimized for specific services such as IoT, video surveillance, or AR/VR. Combined with ultra-low latency and high bandwidth, these features make private 5G ideal for mission-critical applications such as industrial automation, autonomous vehicles, and smart infrastructure [6].

The benefits of private 5G are widely recognized. These networks offer enhanced security by isolating traffic from public infrastructure, improved reliability and performance compared to Wi-Fi, and greater control over network policies and traffic flows. In addition, private 5G facilitates integration with legacy systems while supporting advanced technologies like AI/ML, edge computing, and massive IoT deployments [7].

Use cases span various sectors. In manufacturing, private 5G enables smart factories with real-time monitoring and robotics. In healthcare, it supports telemedicine and connected medical devices. Logistics and warehousing benefit from AGV coordination and inventory management, while educational institutions leverage private 5G for research and immersive learning. Smart cities deploy private 5G for public safety, traffic optimization,

and infrastructure automation [8].

Despite these advantages, several challenges hinder widespread adoption. The high cost of infrastructure, the complexity of deployment and orchestration, and difficulties integrating with legacy systems remain critical barriers. Resource limitations in SMEs and academic institutions further complicate deployment efforts. Finally, the lack of skilled professionals familiar with 5G, SDN, and NFV technologies represents a significant constraint [9].

2.2 Overview of the Aether Platform

Aether is a leading open-source private 5G platform developed by the Open Networking Foundation (ONF) to deliver scalable, flexible, and secure enterprise connectivity. It provides a modular, cloud-native framework for deploying and managing private 5G networks, offering advanced features such as network slicing, runtime policy control, and real-time observability [4]. Aether is particularly well-suited for both enterprise deployment and academic experimentation, thanks to its Kubernetes-based architecture, open APIs, and extensible blueprint system.

Key Components of Aether

Aether consists of several core components that integrate to provide end-to-end private 5G services:

SD-Core (Software-Defined Core): SD-Core, also known as the Aether Core, is a cloud-native mobile core designed to support 4G LTE, 5G non-standalone (NSA), and 5G standalone (SA) deployments within a unified architecture [1]. It provides a modular service-based design, aligning with 3GPP-defined principles for next-generation networks.

The control plane of SD-Core includes the Access and Mobility Management Function (AMF), Session Management Function (SMF), User Plane Function (UPF), User Data Management (UDM), and related functions—all implemented as containerized microservices. The user plane is powered by BESS (Berkeley Extensible Software Switch), enabling high-performance packet forwarding at the edge.

SD-Core exposes a rich runtime API that allows network operators and third-party systems to:

- Dynamically provision and manage subscribers.
- Configure and apply QoS policies across network slices.

- Control runtime behavior of core functions (e.g., SMF, UPF).
- Stream telemetry data for monitoring and closed-loop automation.

This API-driven approach makes SD-Core highly suitable for edge computing environments and allows integration with intelligent applications that can dynamically influence the network.

For development and testing purposes, SD-Core includes `gNBsim`, an open-source radio access emulator. This tool enables simulation of large-scale UE attach, session establishment, and control plane signaling without requiring physical radios, making it ideal for CI/CD pipelines and lab environments [10].

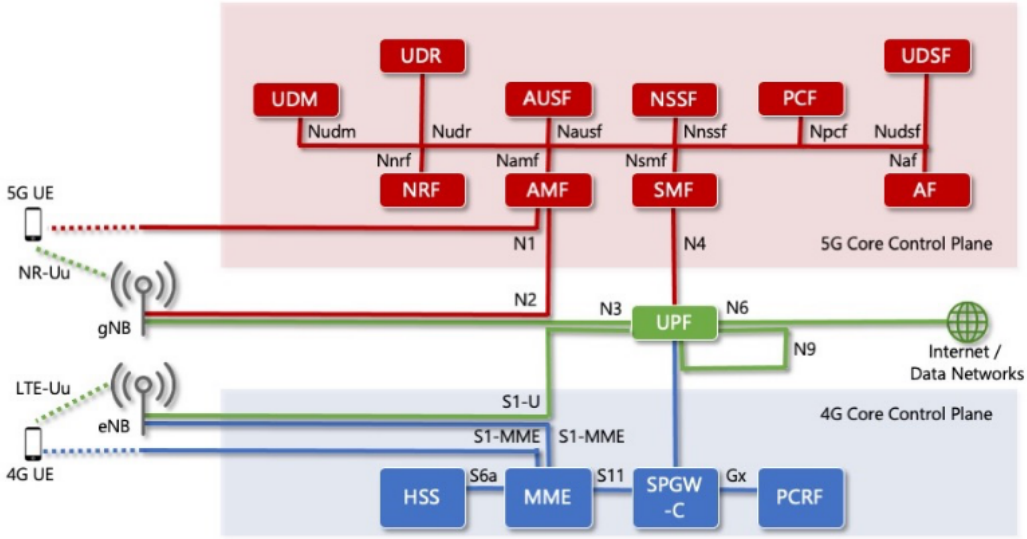


Figure 2.1: SD-Core Architecture Overview [1].

SD-RAN (Software-Defined Radio Access Network) SD-RAN is an O-RAN compliant, open-source Radio Access Network stack developed by the Open Networking Foundation (ONF). It implements the 7.2x functional split, separating the Distributed Unit (DU), Centralized Units for Control (CU-C), and User plane (CU-U). This split supports disaggregation and flexible deployment across edge and cloud environments.

At the heart of SD-RAN is the Near-Real Time RAN Intelligent Controller (nRT-RIC), built on the cloud-native µONOS platform. The nRT-RIC enables policy-based control over RAN behavior by hosting a variety of xApps, including:

- **RRM xApps:** For radio resource management and traffic steering.
- **SON xApps:** Supporting self-optimization of network parameters.
- **Policy xApps:** For applying high-level business rules across the RAN.

The nRT-RIC interacts with RAN components through standardized O-RAN interfaces such as A1 (for policy control) and E2 (for real-time telemetry and control). It supports the E2SM-KPM and E2SM-RC service models for monitoring and control of key performance indicators and RAN configuration, respectively.

To enable closed-loop testing and development of xApps, SD-RAN includes `RANSIM`, a simulator capable of generating realistic radio events and performance metrics. This makes it possible to develop, test, and validate RIC behavior and xApps in a fully virtualized environment [2].

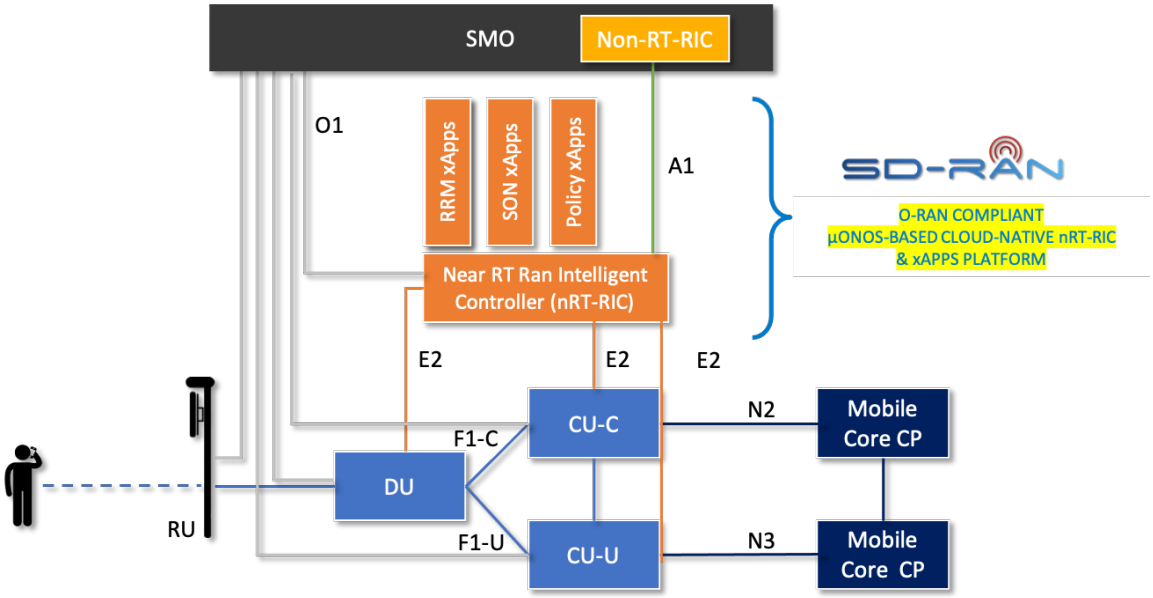


Figure 2.2: SD-RAN Architecture Overview showing the integration of nRT-RIC with xApps, CU/DU elements, and mobile core control planes [2].

AMP (Aether Management Plane): The Aether Management Plane (AMP) provides runtime control, monitoring, and lifecycle management functionality for the Aether platform. It operates on top of SD-Core and SD-RAN to enable dynamic configuration and observability of deployed network services. AMP comprises two principal subsystems:

- **Runtime Operational Control (ROC):** Built on µONOS (a microservices-based evolution of the ONOS SDN controller), ROC exposes REST APIs and a GUI for managing network slices, user groups, policy profiles, and operational parameters at runtime.
- **Monitoring Subsystem:** This subsystem integrates with open-source monitoring tools like Prometheus, Grafana, and Rancher, and features custom dashboards for real-time visibility into Aether's operational state [3].

From an architectural standpoint, AMP enables closed-loop control by combining monitoring and runtime control into a unified interface for operators. Through telemetry data collection and actionable control APIs, AMP supports use cases such as dynamic QoS enforcement, resource scaling, and anomaly detection.

In addition to runtime operations, AMP also interfaces with lifecycle management systems responsible for provisioning, upgrades, and configuration versioning. These responsibilities are coordinated through a code-based pipeline where operators and developers contribute infrastructure changes via repository commits, triggering system upgrades or provisioning workflows.

Figure 2.3 shows a simplified representation of the AMP architecture. It highlights the interactions between users, developers, and operations teams, and how those roles interface with the platform via REST APIs, provisioning pipelines, and runtime controllers.

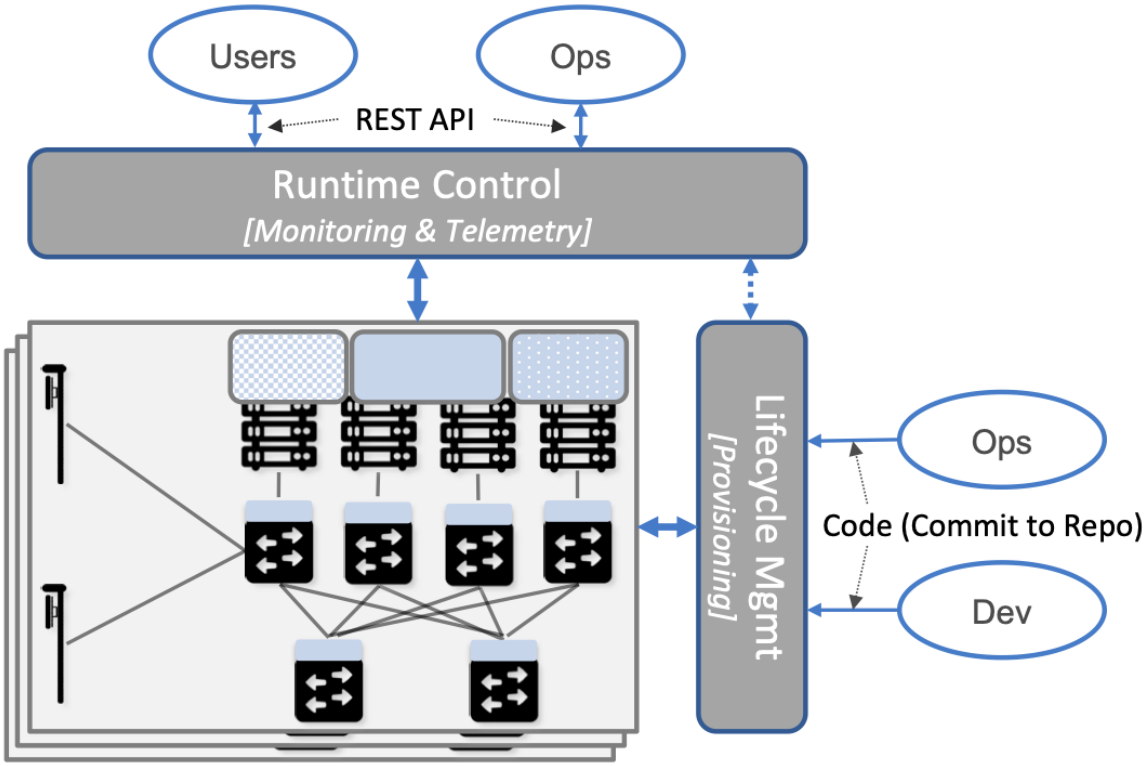


Figure 2.3: Simplified architecture of Aether Management Plane, integrating runtime control (monitoring + telemetry) and lifecycle provisioning [3].

OnRamp: Aether OnRamp simplifies deployment through pre-built infrastructure blueprints, which encapsulate a declarative configuration of the entire stack, including SD-Core, SD-RAN, and AMP. These blueprints are defined using Ansible variables, roles, and playbooks, allowing reproducible and version-controlled deployments across different environments. OnRamp supports Kubernetes-based orchestration (using RKE2) and installs all Aether subsystems as containerized applications via Helm charts [11].

Architecture of Aether

Figure 2.4 shows the high-level architecture of Aether deployed in an edge cloud environment. It highlights how SD-Core, SD-RAN, AMP, and other components interact within a cloud-native stack hosted on-premises. This architecture enables local breakout and aligns with enterprise-grade deployment requirements.

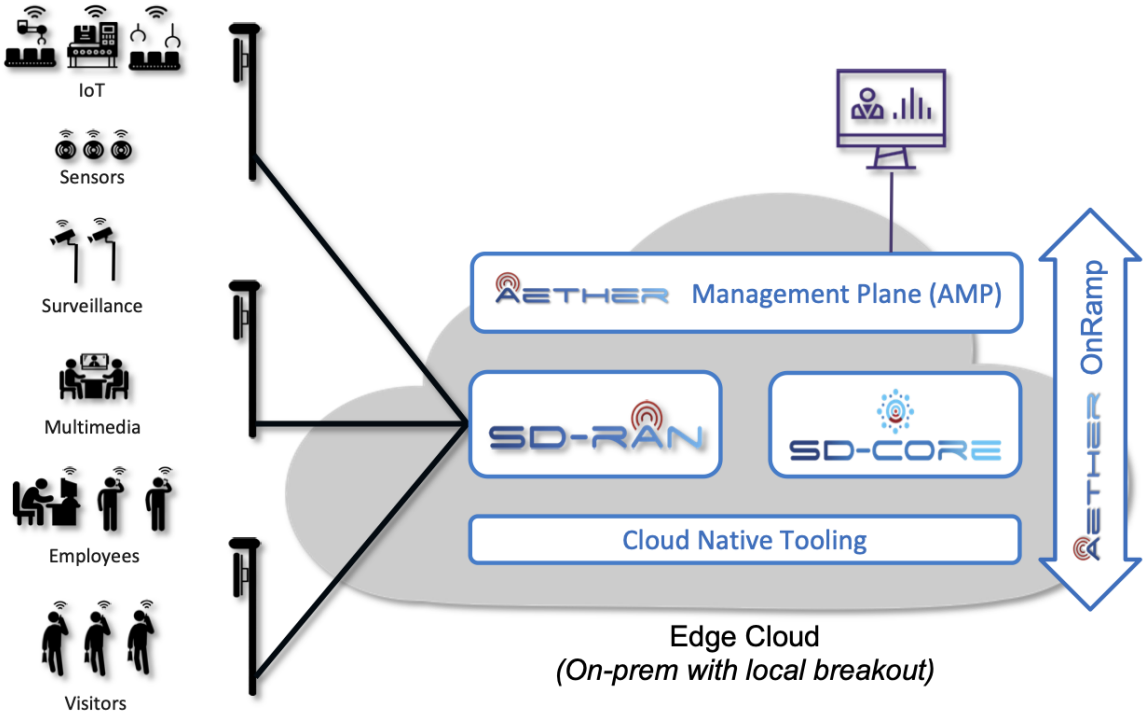


Figure 2.4: High-Level Architecture of the Aether Platform [4].

2.3 Virtualization in 5G

Virtualization is a foundational pillar in modern 5G network design, enabling the shift from hardware-based appliances to software-defined, cloud-native solutions. In traditional mobile networks, core components such as the Access and Mobility Management Function (AMF), User Plane Function (UPF), and Radio Access Network (RAN) controllers were tightly coupled with proprietary hardware. With the introduction of Network Function Virtualization (NFV) and Software-Defined Networking (SDN), these functions are now implemented as software applications that can run on general-purpose servers and virtualized platforms [12].

The 5G ecosystem leverages multiple layers of virtualization. At the infrastructure level, Virtual Machines (VMs) provide hardware isolation and compatibility across different operating systems. On top of VMs, lightweight containers such as Docker streamline the packaging and deployment of 5G components by bundling applications with their

dependencies. Containers are orchestrated using Kubernetes, which has emerged as the industry standard for managing distributed, scalable applications [13]. It automates deployment, scaling, networking, and lifecycle management of containerized services, making it a natural fit for dynamic 5G environments.

Virtualization introduces several benefits for 5G deployments. It improves resource utilization by enabling on-demand scaling of network functions, supports multi-tenancy and network slicing, and accelerates deployment cycles through automated CI/CD pipelines [14]. Moreover, it allows operators to deploy private 5G networks in a variety of scenarios, ranging from public cloud to private on-premise edge clouds, offering maximum flexibility and cost-efficiency.

Despite these advantages, virtualization also presents challenges, especially in resource-constrained environments like small enterprise testbeds or academic labs. Running complex 5G stacks in single-node or dual-node VM setups can lead to CPU, memory, and storage bottlenecks [15]. I/O performance—particularly for telemetry and monitoring pipelines—may degrade in the absence of high-throughput hardware. Additionally, configuring multi-interface networking and maintaining inter-container communication can be technically demanding in environments with limited orchestration support. These constraints are particularly relevant to the context of this thesis, which evaluates Aether's performance under such limitations.

The Aether platform embraces a fully virtualized, cloud-native architecture. Each of its components—SD-Core, SD-RAN, AMP, and OnRamp—is deployed as a set of containerized microservices managed by Kubernetes. Using its Ansible-based blueprint system, Aether can be instantiated on standard virtual machines without the need for specialized hardware. This design not only simplifies deployment but also enables experimentation and testing in real-world, resource-limited environments. It highlights the viability of deploying private 5G networks using open-source software and virtualization tools—a central focus of this research [4].

2.4 Related Work

The deployment of private 5G networks has gained increasing attention in both academic and industrial domains. A range of commercial and open-source platforms have emerged to support enterprise-grade 5G connectivity, with implementations spanning from public cloud environments to on-premise edge infrastructures. Existing literature has explored various architectural models, orchestration frameworks, and performance evaluations of these deployments—especially in large-scale or telecom-grade settings [5, 6].

Several research testbeds, including 5GENESIS, POWDER, and COSMOS, provide extensive infrastructures for 5G experimentation. These platforms support advanced features such as network slicing, user mobility, and RAN control experimentation, and often provide field-tested datasets. However, they typically depend on specialized hardware, dedicated spectrum, and custom orchestration systems, which may be infeasible for institutions or enterprises with limited technical and financial resources [16].

Virtualization is widely regarded as a cornerstone of 5G architecture. Numerous studies have investigated the role of NFV and SDN in enabling scalable, cloud-native deployments of core and RAN functions. Kubernetes-based orchestration of containerized network functions (CNFs) has become a popular research focus due to its flexibility and extensibility in managing dynamic workloads [13, 14]. Nevertheless, most of these studies assume access to robust, resource-rich infrastructure, which does not reflect the constraints of small-scale labs or academic testbeds.

Among open-source 5G platforms, the Aether project by the Open Networking Foundation (ONF) stands out as a modular, cloud-native solution for private 5G. Although its architecture and potential use cases have been described in technical documentation and white papers [4, 1], there is limited academic literature evaluating Aether in constrained or resource-limited settings. In particular, empirical studies examining multi-blueprint deployments, runtime control using ROC, and the use of monitoring tools like Prometheus and Grafana in small-scale environments remain sparse.

This thesis addresses these gaps by evaluating Aether's deployment in VM-based environments with limited resources. It investigates the system's performance, reproducibility, and usability in settings aligned with academic and enterprise edge use cases—thereby contributing new insights to the growing body of research on private 5G platforms.

2.5 Research Gaps

The deployment of private 5G solutions—particularly platforms like Aether—has gained significant momentum due to their potential to deliver secure, flexible, and high-performance connectivity in enterprise and academic environments. However, despite advancements in this domain, several key gaps remain in the existing literature:

1. **Lack of VM-Based Deployment Evaluations**
   Existing studies predominantly focus on cloud-native or containerized deployments, typically leveraging high-resource clusters or public cloud infrastructures. In contrast, there is limited research on deploying and evaluating private 5G platforms like Aether within *virtualized environments using constrained hardware*, such as single-node or dual-node VM setups. This limitation is especially relevant for educational institutions and small enterprises that depend on general-purpose computing infrastructure for experimentation and learning environments.

2. **Insufficient Analysis of Multi-Blueprint Scenarios**
   Aether supports multi-blueprint deployments, including components such as Runtime Operational Control (ROC), multiple UPFs, and monitoring dashboards. However, *empirical studies investigating the interaction, performance, and scalability* of these blueprints—particularly in isolated or resource-limited testbeds—are scarce. Without such evaluations, the real-world applicability and performance trade-offs of complex blueprint combinations remain unclear.

3. **Minimal Stress-Testing in Resource-Constrained Systems**
   While some works explore 5G core network performance under ideal conditions, few attempt to *stress-test private 5G deployments on limited resources*, especially using simulators such as gNBsim or UERANSIM. These tools can expose system bottlenecks related to CPU, memory, and I/O in VM environments, yet this aspect remains largely underexplored.

4. **Underrepresentation of Educational and Reproducible Testbeds**
   Despite the open-source nature of Aether, its potential as an educational platform is not well documented. There is a noticeable gap in the literature regarding the design and deployment of *reproducible, VM-based testbeds* that can be used for teaching, rapid prototyping, or hands-on training. Addressing this can significantly improve accessibility to 5G research and training in university settings.

5. **Limited Insights into Real-Time Control and Monitoring Tools in Small-Scale Deployments**
   Tools like Aether ROC and Prometheus-Grafana are essential for runtime control

and observability in 5G networks. However, their *integration, usability, and performance under stress in small-scale VM testbeds* are rarely analyzed. This limits our understanding of how dynamic policies, monitoring, and network slicing behave outside large-scale, cloud-optimized setups.

By addressing these research gaps, this study contributes to the practical understanding and reproducibility of enterprise-facing private 5G deployments in virtualized environments. The insights gained aim to inform future implementations, both in academic labs and in real-world enterprise settings.

2.6 Chapter Summary and Motivation

This chapter presented a comprehensive background on private 5G networks, focusing on their key architectural features, deployment models, and the virtualization technologies that enable flexible, scalable implementations. It also explored the Aether platform—an open-source, cloud-native private 5G solution—and analyzed its core components, including SD-Core, SD-RAN, AMP, and OnRamp. A review of related research highlighted ongoing efforts in 5G deployment and experimentation, particularly within large-scale cloud or testbed environments.

Despite these advancements, several important research gaps persist, particularly around the deployment and evaluation of private 5G solutions in constrained virtualized environments. These include the limited availability of VM-based testbeds, lack of empirical analysis of multi-blueprint deployments, and underrepresentation of educational or reproducible Aether setups. This thesis addresses these gaps by studying Aether's behavior in resource-limited VM setups, with a focus on reproducibility, runtime control, and performance stress-testing.

The next chapter presents the methodology, experimental testbed, and blueprint configurations used to evaluate these research challenges in a controlled, virtualized setting.

Chapter 3

Methodology

This chapter outlines the methodology used to deploy and test the Aether platform in two distinct setups:

- **Quick Start Deployment (Personal Laptop)**: A constrained environment for evaluating SD-Core using gNBsim.
- **Full Aether Deployment (Lab PC)**: A large-scale deployment integrating multiple blueprints, runtime operational control, and advanced performance testing.

Each deployment aimed to assess the functionality, performance, and scalability of the Aether platform under different configurations.

3.1 Experimental Setup and Testbed Configuration

The experimental setup was designed to evaluate Aether's deployment under different conditions. The configurations for both setups are detailed below.

3.1.1 Quick Start Deployment (Personal Laptop)

The Aether Quick Start deployment was conducted in a single virtualized environment on a personal laptop, serving as a testbed for:

- Deploying Aether SD-Core and gNBsim within a single VM.
- Evaluating 5G functionalities using simulated UE interactions.
- Observing performance bottlenecks in a constrained system.

3.1.1.1 Hardware Specifications

Table 3.1 outlines the specifications of the personal laptop used for this deployment. The laptop is equipped with an AMD Ryzen 7 5800H processor, 32 GB of RAM, and a 287 GB SSD partition. Oracle VirtualBox serves as the hypervisor for running the virtual machine.

Table 3.1: Personal Laptop Specifications

Component	**Specification**
CPU	AMD Ryzen 7 5800H (8 cores, 16 threads)
RAM	32 GB
Storage	287 GB SSD partition (191 GB used, 81 GB free)
Hypervisor	Oracle VirtualBox

The laptop's hardware configuration ensures sufficient computational resources to run the virtualized environment while maintaining reasonable performance for testing purposes.

3.1.1.2 Virtual Machine Configuration

Table 3.2 details the configuration of the virtual machine (VM) used for the Quick Start deployment. The VM runs Ubuntu Server 22.04 and is allocated 8 vCPUs, 24 GB of RAM, and a 50 GB virtual disk with dynamic allocation. The network adapter is set to NAT mode to facilitate internet connectivity.

Table 3.2: VM Configuration for Aether Quick Start Setup

VM Parameter	**Specification**
VM Name	Aether
CPU	8 vCPUs
RAM	24 GB
Storage	50 GB virtual disk (dynamic allocation)
Operating System	Ubuntu Server 22.04
Network Adapter	NAT Network (for internet connectivity)

This configuration provides a balance between resource allocation and system performance, ensuring that the VM can handle the demands of deploying Aether SD-Core and gNBsim while operating within a constrained environment.

3.1.1.3 Deployment Overview

Figure 3.1 provides a high-level overview of the Quick Start deployment architecture. The diagram illustrates the layers involved in setting up Aether SD-Core and gNBsim within a single VM on the personal laptop.

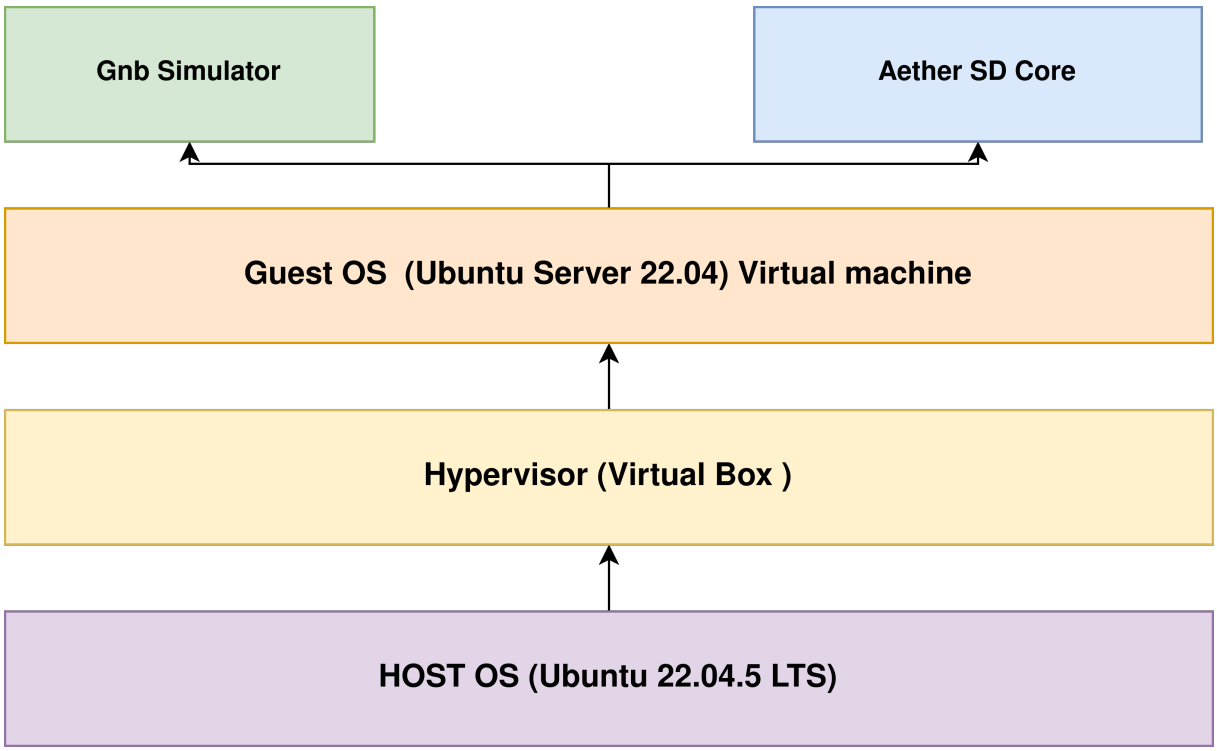


Figure 3.1: Deployment Overview for Quick Start Setup.

As depicted in Figure 3.1, the deployment consists of four main layers:

- **Host OS:** The host operating system is Ubuntu 22.04.5 LTS, which serves as the base layer for running the hypervisor.
- **Hypervisor:** Oracle VirtualBox acts as the hypervisor, providing the necessary abstraction to run the guest operating system.
- **Guest OS:** The guest operating system is Ubuntu Server 22.04, running within the virtual machine. This layer hosts the Aether SD-Core and gNBsim components.
- **Components:** Within the guest OS, two key components are deployed:
  - **Aether SD-Core:** The Software-Defined Core Network responsible for managing 5G core functionalities.
  - **gNBsim:** A 5G simulator used to mimic real-world network interactions and evaluate 5G functionalities.

This layered approach allows for a controlled and isolated environment to deploy and test Aether SD-Core and gNBsim. By utilizing a single VM on a personal laptop, the setup enables researchers to observe performance bottlenecks and assess the feasibility of deploying private 5G solutions in a constrained system.

3.1.1.4 Purpose of the Quick Start Deployment

The Quick Start deployment focuses on evaluating the fundamental aspects of Aether SD-Core in a minimalistic setup. Key objectives include:

- Testing **UE registration** and **PDU session establishment** using gNBsim.
- Measuring **latency** and **response times** for critical call flows.
- Identifying **performance bottlenecks** caused by resource constraints in the virtualized environment.

This setup serves as a baseline for more advanced deployments, providing insights into the scalability and reliability of Aether in resource-constrained scenarios.

3.1.2 Full Aether Deployment (Lab PC)

The full-scale Aether deployment was conducted on a high-performance Lab PC, enabling more advanced testing scenarios. This section describes the hardware specifications, virtual machine configurations, and overall deployment architecture for the Lab PC setup.

3.1.2.1 Hardware Specifications

Table 3.3 details the specifications of the Lab PC used for the full-scale Aether deployment. The system features an Intel Core i7-12700 processor, 64 GB of RAM, and a 500 GB SSD. Oracle VirtualBox is used as the hypervisor to run multiple virtual machines, supporting both the core Aether platform and UERANSIM.

3.1.2.2 Virtual Machine Configuration

Two virtual machines were deployed to facilitate the full Aether platform and UERANSIM components. The first VM hosts the Aether platform core services, while the second VM runs UERANSIM for simulating 5G gNB and UE functionalities.

The Aether Platform Core Deployment VM provides the underlying Kubernetes environment for deploying multiple Aether blueprints, including the SD-Core, Runtime

Table 3.3: Lab PC Specifications

Component	**Specification**
CPU	Intel Core i7-12700 (12 cores, 20 threads, up to 4.9 GHz)
RAM	64 GB
Storage	500 GB SSD (457 GB partition, 408 GB used, 28 GB free)
Operating System	Ubuntu 22.04.5 LTS
Kernel	Linux 6.8.0-52-generic
Hypervisor	Oracle VirtualBox

Table 3.4: Aether Platform Core Deployment VM Specifications

VM Parameter	**Specification**
Purpose	Core deployment of Aether platform components
OS	Ubuntu Server 22.04
CPU	20 vCPUs
RAM	40 GB
Storage	Virtual Disk (VMDK, dynamically allocated)
Network Adapter	Host-only Adapter (vboxnet0)

Table 3.5: UERANSIM VM Configuration

VM Parameter	**Specification**
Purpose	Simulate 5G User Equipment and Radio Access Network
OS	Ubuntu Server 22.04
CPU	3 vCPUs
RAM	10 GB
Storage	Virtual Disk (VMDK, dynamically allocated)
Network Adapters	Adapter 1: NAT Network (internet connectivity)
	Adapter 2: Host-only Adapter (vboxnet0)

Operation Control (ROC), and associated monitoring components. The UERANSIM VM, on the other hand, is responsible for simulating 5G radio and user equipment functionalities, enabling realistic testing of the end-to-end 5G core network.

3.1.2.3 Deployment Overview

Figure 3.2 presents a high-level architecture of the Full Aether Deployment on the Lab PC. As in the Quick Start setup, the diagram illustrates the layering from the host operating system down to the virtualized components, but with two separate virtual machines to accommodate more advanced testing scenarios.

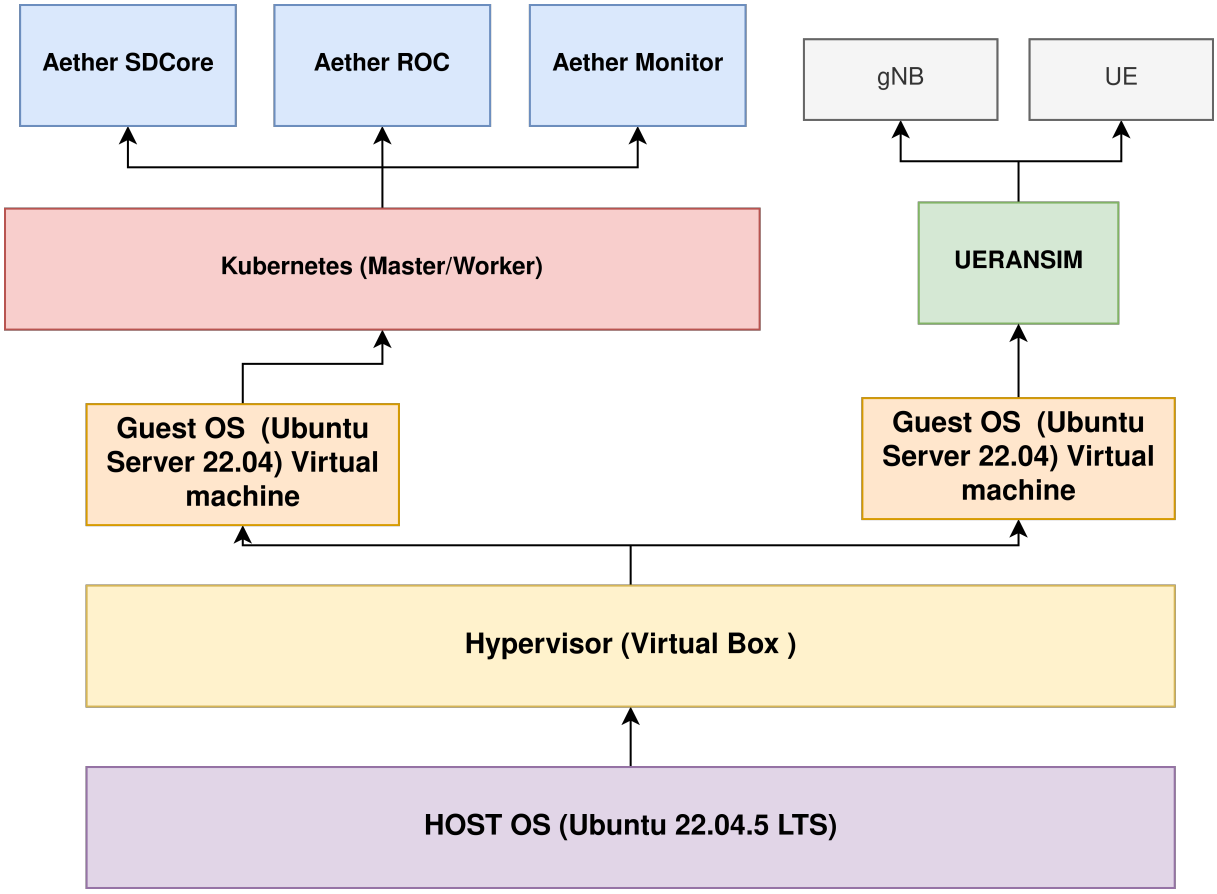


Figure 3.2: Deployment Overview for Full Aether Setup.

As depicted in Figure 3.2, the deployment consists of the following layers:

- **Host OS (Ubuntu 22.04.5 LTS):** The base operating system running on the Lab PC, providing the platform for the hypervisor.
- **Hypervisor (Oracle VirtualBox):** Responsible for abstracting hardware resources and managing the guest operating systems.
- **Guest OS (Two Separate VMs):**

  - **Aether Platform Core VM:** Hosts the Aether SD-Core, Aether ROC, Aether Monitor, and other supporting microservices.
  - **UERANSIM VM:** Simulates the 5G gNB and UE, enabling end-to-end testing of 5G call flows, QoS management, and runtime operations.

- **Aether Components:**
  - **Aether SD-Core:** Manages the 5G core functionalities, including AMF, SMF, UPF, and more.
  - **Aether ROC:** Provides Runtime Operation Control, enabling dynamic parameter modifications and policy enforcement.
  - **Aether Monitor:** Collects metrics and performance data for real-time and historical analysis.
  - **UERANSIM (gNB + UE):** Emulates the radio interface and user equipment, allowing for realistic testing of registration, PDU sessions, and QoS flows.

This multi-VM architecture allows for a clearer separation of core network services and simulated radio components, facilitating more complex scenarios and improved resource management compared to the single VM setup.

3.1.2.4 Purpose of the Full Aether Deployment

The Full Aether Deployment on the Lab PC aims to explore advanced features and stress-test the Aether platform in a more resource-rich environment. Key objectives include:

- **Multiple Blueprint Deployment:** Evaluating the SD-Core alongside other Aether services, such as Runtime Operation Control and additional UPFs.
- **QoS Testing:** Investigating how multiple UPFs and QoS policies impact latency, throughput, and overall network performance.
- **Dynamic Parameter Modification:** Observing how Runtime Operation Control can adjust parameters (e.g., slice configuration, policy rules) during runtime.

By leveraging a high-performance Lab PC and multiple VMs, this deployment provides a robust environment for validating the scalability, resilience, and feature set of the Aether platform. The insights gained from this setup help inform real-world private 5G deployments and further research into 5G network slicing and orchestration.

3.2 Deployment Process

The deployment process for the Aether platform was conducted in two distinct phases:

1. **Quick Start Deployment (Personal Laptop)**: A minimal yet functional setup designed to demonstrate basic 5G core functionalities using a single virtual machine (VM).
2. **Full Aether Deployment (Lab PC)**: A more comprehensive and feature-rich environment leveraging multiple VMs to support advanced testing scenarios, including runtime operation control (ROC), Quality of Service (QoS) management, and dedicated monitoring services.

Each phase builds upon the previous one, starting with a simplified setup on a personal laptop and progressing to a robust multi-VM architecture on a Lab PC. The following sections provide a detailed breakdown of both deployments, their network topologies, and key interactions.

—

3.2.1 Quick Start Deployment (Personal Laptop)

3.2.1.1 Overview

The Quick Start Deployment serves as an introductory setup, demonstrating the fundamental functionalities of a 5G core network. This deployment utilizes a single VM hosted on a personal laptop, integrating the **SD-Core** (including AMF, SMF, UDM, and UPF) alongside **gNBsim**. By consolidating all components into a single environment, this setup provides a straightforward platform for initial testing and demonstration of 5G call flows, user registration, and basic data-plane interactions.

3.2.1.2 Networking Diagram

Figure 3.3 illustrates the networking topology for the Quick Start deployment. The diagram highlights the role of the `DATA_IF` interface, which connects to the internet, and the `gnbaccess` bridge, which facilitates communication between the `gNBsim` container and the core network.

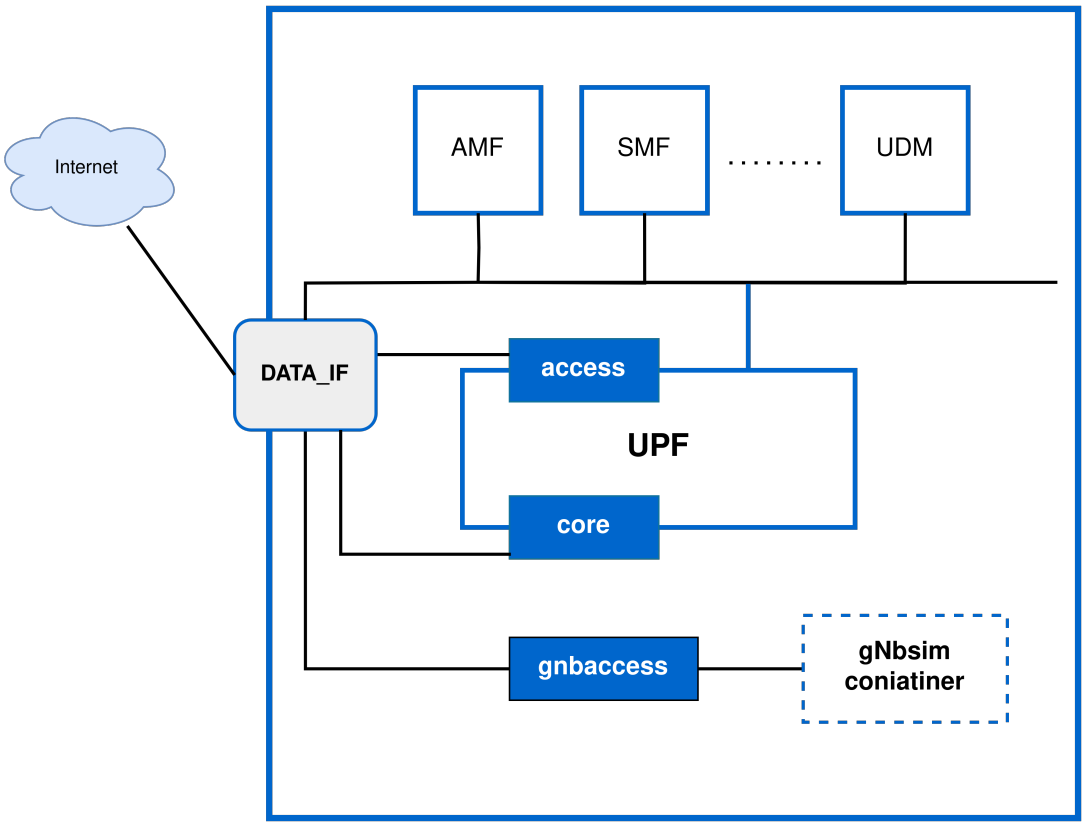


Figure 3.3: Networking Diagram for Quick Start Deployment.

As shown in Figure 3.3, the key components of this deployment include:

- **Internet Connection**: The `DATA_IF` interface provides external connectivity.
- **UPF (User Plane Function)**: Handles user-plane traffic by bridging data packets between the core network and the internet.
- **Core Network Components (AMF, SMF, UDM)**: Manage access, mobility, session establishment, and subscriber data.
- **gNBsim Container**: Simulates the gNodeB (gNB), enabling the evaluation of 5G functionalities such as registration, PDU sessions, and QoS enforcement.

3.2.1.3 Detailed Networking Diagram

Figure 3.4 provides a more granular view of the Quick Start deployment, illustrating how traffic flows among the physical interface (`enp0s3`), the `access` and `core` bridges, and the `UPF` container.

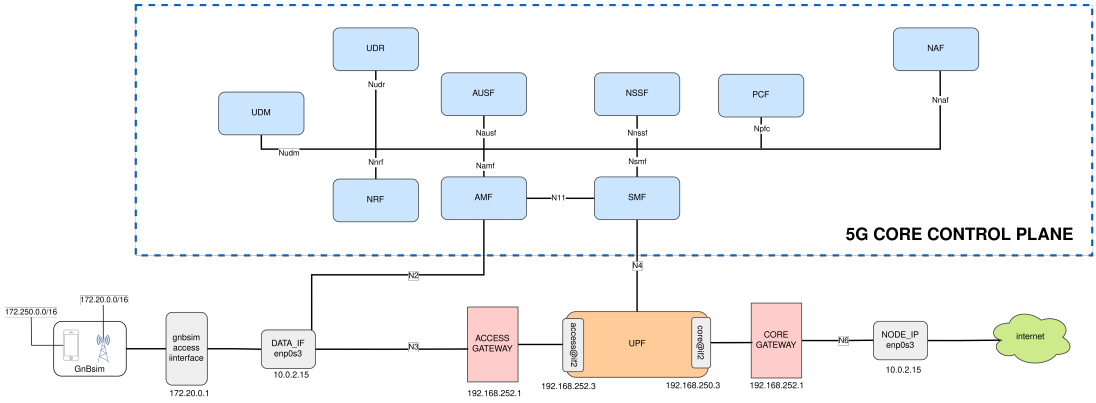


Figure 3.4: Detailed Networking Diagram for Quick Start Deployment.

Key Elements:

- **Physical Interface (`enp0s3`)**: Serves as the primary connection point for external traffic.
- **Access Bridge**: Connects the UPF downstream to the Radio Access Network (RAN), corresponding to the 3GPP N3 interface.
- **Core Bridge**: Connects the UPF upstream to the internet, corresponding to the 3GPP N6 interface.
- **gNBsim Container**: Sends and receives GTP-encapsulated packets to/from the UPF's `access` interface.
- **UPF**: remove or adds GTP headers to forward traffic to the internet or back to the gNBsim container.

3.2.1.4 Key Interactions

The Quick Start Deployment involves two primary types of interactions:

- **Data Flow**: User data flows from the internet through the `DATA_IF` interface to the UPF. The UPF then routes the packets to the `gNBsim` container via the `access` bridge.

- **Signaling Flow**: Signaling messages are exchanged between the `gNBsim` container and the core network components (AMF, SMF, UDM) via the `gnbaccess` bridge. These interactions support tasks such as user registration, session establishment, and IP address allocation.

3.2.1.5 Packet Flow Analysis

Upstream Flow (gNB → Internet)

- The `gNBsim` container sends GTP-encapsulated packets to the UPF's `access` interface (`192.168.252.3`).
- The UPF removes the GTP headers and forwards the raw IP packets to its `core` interface.
- Packets exit to the internet via the physical interface (`enp0s3`), with source NAT applied as needed to ensure private UE IPs remain hidden.

Downstream Flow (Internet → gNB)

- Incoming packets arrive on the physical interface (`enp0s3`) from the internet.
- The UPF encapsulates these packets with GTP headers and sends them to the `gNBsim` container over the `access` bridge.

3.2.1.6 Benefits of the Setup

The Quick Start Deployment offers several advantages:

- **Realistic Simulation**: The use of `access` and `core` bridges ensures realistic simulation of 5G network interactions, enabling comprehensive testing of UPF functionality.
- **Simplified Configuration**: All components (SD-Core, gNBsim, UPF) reside in a single VM, streamlining deployment and reducing complexity.
- **Resource Efficiency**: The single-VM setup is optimized for constrained environments like personal laptops, making it accessible for initial experimentation.

—

3.2.2 Full Aether Deployment (Lab PC)

3.2.2.1 Overview

Building on the foundation of the Quick Start Deployment, the Full Aether Deployment introduces a more sophisticated and scalable environment on a Lab PC. This deployment leverages multiple virtual machines (VMs) to host the SD-Core, Aether Runtime Operation Control (ROC), multiple User Plane Functions (UPFs), and dedicated monitoring services. This multi-VM architecture supports advanced runtime control, QoS management, and in-depth performance analysis, enabling the exploration of complex 5G scenarios.

3.2.2.2 Networking Diagram

Figure 3.5 illustrates the networking setup for the Full Aether Deployment. The diagram highlights the interaction between the UERANSIM container (simulating user equipment and gNodeB behavior), multiple UPFs (UPF0, UPF1), the core network (AMF, SMF, UDM), and the external internet.

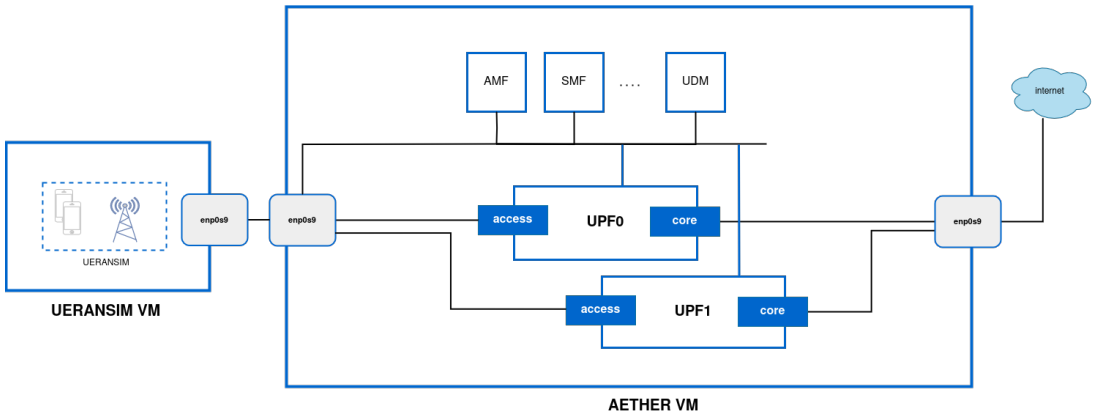


Figure 3.5: Networking Diagram for Full Aether Deployment on the Lab PC.

As depicted in Figure 3.5, the key components of this deployment include:

- **UERANSIM (UE Simulation)**: Emulates gNB + UE behavior via the `enp0s9` interface, enabling realistic testing of registration, session establishment, and data exchange.
- **Core Network (AMF, SMF, UDM)**: Manages authentication, session establishment, and subscriber data.
- **Multiple UPFs (UPF0, UPF1)**: Each UPF handles user-plane traffic through `access` and `core` interfaces, supporting multi-path scenarios and load balancing.

- **Internet Connection**: The core network connects to the external internet via the Lab PC's physical interface (`enp0s9`), enabling end-to-end traffic flows.

3.2.2.3 Detailed Components and Interactions

Virtual Machines (VMs)

- **VM1**: Hosts the SD-Core (AMF, SMF, UDM), Aether ROC for runtime control, multiple UPFs for user-plane traffic management, and monitoring services for capturing performance metrics and enabling real-time troubleshooting.
- **VM2**: Dedicated to RAN simulation, running UERANSIM to emulate user equipment and gNodeB behavior.

Traffic Flow and Control Plane Interactions

- **UE Traffic Entry**: UERANSIM's `enp0s9` interface provides the entry point for simulated UE data.
- **Core Network Processing**: The AMF, SMF, and UDM authenticate users, manage sessions, and store subscriber data.
- **Multi-UPF Handling**: The SMF selects either UPF0 or UPF1 for data routing, allowing testing of multi-path scenarios or load balancing.
- **Internet Egress**: Traffic exits to the internet via the Lab PC's physical interface, with each UPF's `core` interface handling GTP de-encapsulation.

3.2.2.4 Detailed Networking Diagram

Figure 3.6 provides a more granular view of the Full Aether Deployment, highlighting the interactions among various Aether platform services, runtime monitoring tools, and the multiple UPFs.

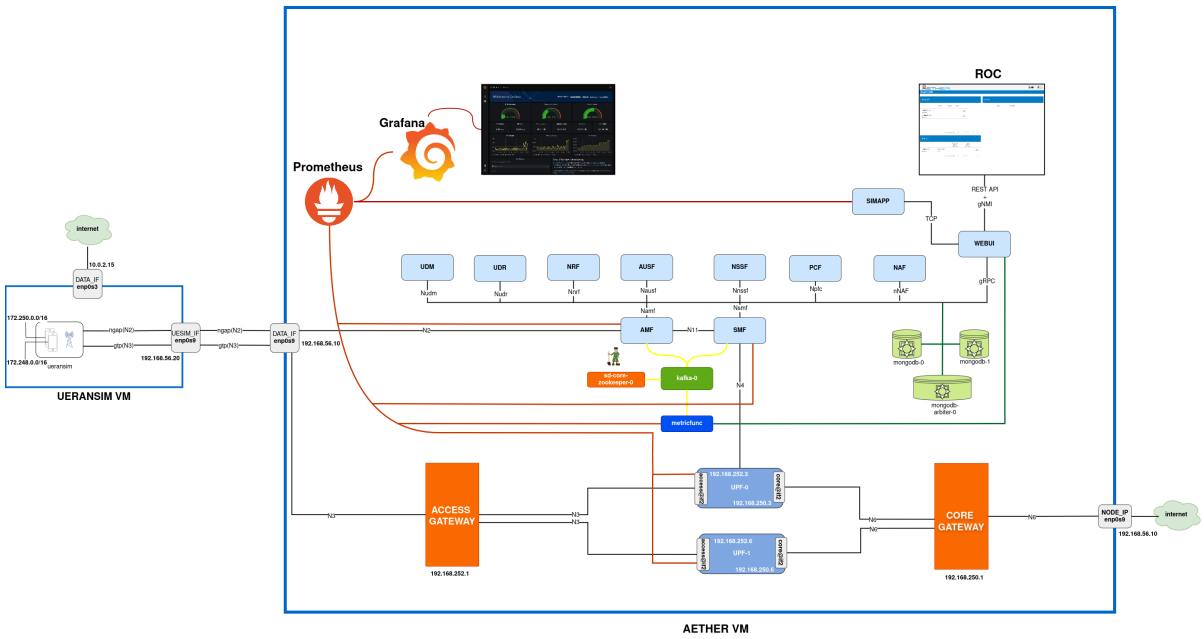


Figure 3.6: Detailed Networking Diagram for Full Aether Deployment on the Lab PC.

Key Components and Interactions:

- **UERANSIM VM (Left Side)**:
  - Simulates 5G User Equipment (UE) and gNodeB (gNB), generating both control-plane and user-plane traffic.
  - Connects to the Aether VM via the `enp0s9` interface, enabling registration, session establishment, and data exchange.
- **Aether VM (Center)**:
  - Hosts the primary 5G core functions (AMF, SMF, UDM, PCF, NSSF) and multiple UPFs (`UPF0` and `UPF1`).
  - Includes **Access Gateway** and **Core Gateway** components to route traffic from the RAN side (UERANSIM) to the internet, and vice versa.
  - **Aether Runtime Operation Control (ROC)**: Dynamically manages slice configurations, QoS parameters, and network policies in real time.
- **Monitoring and Visualization Tools (Top)**:

  - **Prometheus**: Collects metrics from the core network components and UPFs.
  - **Grafana**: Provides a graphical dashboard for real-time performance monitoring (e.g., throughput, latency, CPU/memory usage).
  - **Syslog**: Captures events and logs from the Aether platform for troubleshooting and auditing.

- **External Interfaces (Right Side)**:
  - The **Core Gateway** connects the UPFs to the external internet, enabling data-plane traffic to flow from simulated UEs to external networks.
  - The `enp0s9` interface represents a bridged or NATed connection to the Lab PC's physical NIC, depending on the VirtualBox networking setup.

Traffic Flow Overview:

- *UE-to-Internet (Upstream)*:
  - UERANSIM (UE) transmits GTP-encapsulated packets through the Access Gateway to the selected UPF (`UPF0` or `UPF1`), where GTP headers are removed.
  - The UPF forwards plain IP packets to the Core Gateway for egress to the internet.
- *Internet-to-UE (Downstream)*:
  - Packets from the internet enter via the Core Gateway, reach the chosen UPF, and are then GTP-encapsulated before being sent to UERANSIM through the Access Gateway.
- *Control Plane Signaling*:
  - The AMF, SMF, and UDM exchange signaling messages with UERANSIM for registration, session management, and subscriber data retrieval.
  - The ROC component dynamically alters policies, slice definitions, or QoS parameters in real time, with changes immediately reflected in the core network's behavior.

3.2.2.5 Benefits of the Setup

The Full Aether Deployment offers several advantages:

- **Enhanced Functionality**: Multiple VMs and advanced services (ROC, monitoring) enable testing of complex 5G scenarios and research use cases.
- **Scalability**: The modular design allows for adding more UPFs or additional VMs as network demands evolve.
- **Detailed Performance Analysis**: Monitoring tools provide granular visibility into metrics, facilitating performance tuning and troubleshooting.

In summary, the Full Aether Deployment on the Lab PC provides a robust platform for evaluating 5G functionalities under more realistic conditions than the single-VM Quick Start setup. Subsequent chapters (e.g., Chapter 5) will present performance measurements and analyses derived from this environment. All diagrams are available in both PDF and editable `.drawio` formats. Readers can download and reuse these resources for replication or adaptation purposes.(see Appendix C)

3.3 Testing Strategy

The functional and performance evaluation of the Aether platform in the Quick Start Deployment focused on validating the core functionalities of SD-Core and analyzing its behavior under varying conditions. The testing strategy was divided into two main categories: Quick Start Testing with gNBsim and Full Aether Testing with UERANSIM. This section elaborates on the Quick Start Testing with gNBsim, which served as the foundation for evaluating Aether's performance in a constrained environment.

3.3.1 Quick Start Testing with gNBsim

To validate the functionality and performance of Aether SD-Core in the Quick Start environment, we employed *gNBsim*, a 5G simulator designed to mimic real-world network interactions. The primary objectives of this testing phase were:

- **Evaluate Core Network Processing:** Assess the ability of SD-Core to handle key tasks such as UE registration, session establishment, and dynamic resource management.
- **Measure Latency and Response Times:** Capture the time taken for critical signaling procedures to complete, providing insights into the system's responsiveness.

- **Analyze System Scalability:** Observe the behavior of SD-Core when handling varying numbers of UEs, from low load (1–5 UEs) to moderate or higher loads (up to 50 UEs), and identify potential bottlenecks.

3.3.1.1 Test Scenarios and Expectations

The testing phase included several scenarios, each replicating real-world user activities to assess network responsiveness and stability. These **key test scenarios** match the procedures we implemented in our Python scripts (see Section 5.3.1 for actual measurements and visualizations):

1. **UE Registration**
   - **Scenario:** Simulate a UE attaching to the network for the first time.
   - **Expected Outcome:** Successful registration with proper authentication, allocation of an IP address, and establishment of a network session.
   - **Purpose:** Validate the AMF's ability to authenticate and register the UE.
2. **UE Initiated Session (PDU Session Establishment)**
   - **Scenario:** A registered UE initiates a PDU session to enable data connectivity.
   - **Expected Outcome:** Successful session establishment with correct configuration of tunneling mechanisms between the UE and the UPF.
   - **Purpose:** Test the SMF's handling of session creation and management.
3. **Access Network (AN) Release**
   - **Scenario:** Simulate the release of RAN resources when a UE moves out of coverage or transitions to another network.
   - **Expected Outcome:** Efficient and clean release of network resources with no residual configurations.
   - **Purpose:** Ensure the network dynamically frees resources to optimize performance.
4. **UE Initiated Service Request**
   - **Scenario:** A previously idle UE requests to resume network activity.
   - **Expected Outcome:** Rapid re-establishment of connectivity and prompt allocation of necessary resources.
   - **Purpose:** Evaluate the efficiency of the network in handling transitions from idle to active states.

5. **UE Initiated De-registration**
    - **Scenario:** Simulate a UE explicitly disconnecting from the network.
    - **Expected Outcome:** Successful termination of the network session and proper cleanup of resources.
    - **Purpose:** Assess SD-Core's capability to gracefully remove UE contexts and free resources.
6. **UE Initiated Session Release**
    - **Scenario:** Evaluate the termination of an active session without disrupting other ongoing sessions.
    - **Expected Outcome:** Clean session termination while maintaining overall service stability.
    - **Purpose:** Test the robustness of session management and resource re-allocation.

3.3.1.2 Load Levels and Key Metrics

While the initial plan mentioned loads ranging from 1–5 UEs to around 50 UEs, we further extended tests in some cases to 20, 50, or even 100 UEs for deeper analysis. In each load scenario, every **key procedure** (Registration, PDU Session, etc.) was invoked multiple times in parallel using gNBsim.

To measure **performance** and **stability** under increasing concurrency, we recorded:

- **Completion Time** for each scenario (e.g., `REGISTRATION-PROCEDURE`, `PDU-SESSION-ESTABLISHMENT-PROCEDURE`), from request initiation to `PROC-PASS-EVENT`.
- **Sub-Phase Durations**, such as the time between authentication request and response, or time from service request to acceptance.
- **Success vs. Failure Counts**: Ensuring that all or most UEs complete without error under higher loads.

Because the *Quick Start* deployment runs everything on a single VM, we anticipate:

- Higher CPU and memory usage with larger UE counts.
- Latency Growth for procedures that require more extensive signaling (Registration, PDU Session).
- Occasional bottlenecks in the AMF/SMF if multiple users request or release sessions at the same time.

3.3.1.3 Stress-Test Automation with gNBsim

To automate these repeated procedures, we defined a custom `stressTest` profile in gNBsim. This profile allows specifying:

- The **number of UEs** (`ueCount`) to simulate,
- The **iterations** (phases) that repeat certain procedures (Registration, PDU Session, etc.) multiple times,
- Whether to run each step **in parallel** among all UEs, maximizing concurrency.

By adjusting `ueCount` to 5, 20, 50, or 100, we systematically increased load. Within each iteration, we repeated the following procedures a fixed number of times (e.g., 5, 10, or 100):

- `REGISTRATION-PROCEDURE`
- `PDU-SESSION-ESTABLISHMENT-PROCEDURE`
- `AN-RELEASE-PROCEDURE`
- etc.

This ensured each scenario was triggered enough to observe potential contention or delays on SD-Core.

```
customProfiles:
  stressTestProfile:
    profileName: "stressTest"
    ueCount: 5 # e.g., changed to 20, 50, or 100 in other runs
    execInParallel: true
    iterations:
      - name: "iteration1"
        "1": "REGISTRATION-PROCEDURE 5"
        "2": "PDU-SESSION-ESTABLISHMENT-PROCEDURE 5"
      - name: "iteration2"
        "1": "AN-RELEASE-PROCEDURE 100"
        "2": "UE-TRIGGERED-SERVICE-REQUEST-PROCEDURE 10"
        repeat: 100
        next: "iteration3"
      - name: "iteration3"
        "1": "UE-INITIATED-DEREGISTRATION-PROCEDURE 10"
```

Listing 3.1: stressTestProfile for gNBsim

Here, `REGISTRATION-PROCEDURE 5` indicates that each UE performs five consecutive registration attempts. Similarly, `AN-RELEASE-PROCEDURE 100` triggers 100 RAN release operations in sequence.

The `repeat: 100` directive within `iteration2` causes this iteration to be replayed 100 times, simulating sustained load conditions. Meanwhile, the setting `execInParallel: true` ensures that all UEs execute the same procedure concurrently, significantly increasing the concurrency level and testing the SD-Core under stress.

3.3.1.4 Summary

Example YAML Excerpt. This testing strategy for the Quick Start Deployment with *gNBsim* systematically evaluated **Aether SD-Core's** performance under a variety of realistic 5G procedures and load conditions. Key outcomes include:

- **Validation of Core Functionalities**: From UE registration and PDU session setup to service requests and deregistration, each procedure was executed multiple times in parallel, confirming SD-Core's ability to handle core tasks end-to-end.
- **Latency and Response Times Measured**: By recording per-phase delays, total procedure times, and pass/fail rates, we gained quantitative insights into how fast (or slow) each operation completed, especially as the number of UEs increased.

- **Scalability Observed in a Single-VM Setup**: Running loads of 1–5, 20, 50, and even 100 UEs revealed how CPU contention and concurrency affect signaling procedures in an *all-in-one* environment. The results serve as a baseline for more advanced, distributed deployments.
- **Automated Stress Testing with Iterations**: A custom `stressTest` profile in gNBsim allowed us to repeat each procedure (e.g., Registration, AN Release) tens or hundreds of times, exposing potential bottlenecks and ensuring consistent logging of performance data.

Overall, these findings inform our **next steps** in analyzing SD-Core's robustness and efficiency. In the following chapter, we provide detailed implementation steps, deployment walk-throughs, and comprehensive results that further illustrate how Aether SD-Core scales and adapts under varying conditions.

3.3.2 Full Aether Testing (UERANSIM, Multi-UPF, Runtime Control)

The Full Aether Deployment on the Lab PC aimed to evaluate the scalability, flexibility, and real-time configurability of SD-Core. Unlike Quick Start deployment, which was contained within a single virtual machine, this setup incorporated multiple network components to facilitate advanced testing scenarios.

3.3.2.1 Objectives and Testing Scope

The primary objectives of this phase were:

- Deployment of multiple UPFs to evaluate Quality of Service (QoS) policies.
- Testing Runtime Operational Control (ROC) for managing subscribers, profiles, and policy enforcement.
- Observation of real-time performance adjustments when network parameters are modified dynamically.

3.3.2.2 Testing with UERANSIM

Unlike the Quick Start deployment that used gNBsim, this setup leveraged UERANSIM, an open-source UE and RAN simulator, to evaluate SD-Core functionalities in a more realistic scenario. The following tests were conducted:

- UE registration - Simulating connection of user equipment to the network.
- PDU Session Establishment – Validating data session creation and network routing.

These tests were aimed at measuring latency, session reliability, and network stability under different operational conditions.

3.3.2.3 Multi-UPF Deployment and QoS Testing

To assess traffic steering and Quality of Service (QoS) management, multiple UPFs were deployed, each handling different traffic classes. The testing focused on the following.

- Ensure that different traffic profiles were routed through specific UPFs based on policies.
- Evaluating bandwidth allocation per subscriber profile.

These evaluations provided insightinto how SD-Core dynamically adjusts user plane routing and prioritizes different network flows.

3.3.2.4 Evaluating Runtime Operational Control (ROC)

The Aether Runtime Operational Control (ROC) was evaluated for its ability to dynamically manage subscribers, profiles, and policies. ROC was tested in the following areas:

- Adding, updating and removing user profiles.
- Applying dynamic rules to modify network behavior, such as bandwidth limits and access control.
- Propagating configuration changes to SD-Core and ensuring network consistency.

Key Considerations:

- ROC does not deploy or manage SD-Core services; it configures existing components.
- ROC does not act as a message bus or directly collect system logs; it integrates with external tools for observability.

The ability to make configuration changes without disrupting services demonstrated the agility of ROC in managing enterprise 5G networks.

3.4 Summary

This chapter described the methodology for deploying and testing Aether under two different setups:

- Quick Start Deployment – A minimal setup focusing on functional testing using gNBsim.
- Full Aether Deployment – A comprehensive environment enabling multi-UPF testing, QoS evaluations, and runtime network control via ROC.

The testing methodologies provided a structured approach to assessing Aether's performance, scalability, and configurability under different network configurations. The following chapters will present:

- **Chapter 4 (Implementation and Testing):** Detailed step-by-step deployment walkthroughs.
- **Chapter 5 (Results and Discussion):** Performance evaluation, observations, and system behavior analysis.

Chapter 4

Implementation

4.1 Quick Start Deployment (Personal Laptop)

The Aether Quick Start deployment was conducted in a single virtualized environment on a personal laptop. In this setup, we evaluated the core functionalities of Aether SD-Core and simulated a 5G network using gNBsim under resource-constrained conditions. Our configuration closely followed the Aether OnRamp Quick Start guide (see [11]), with modifications to accommodate our specific hardware and network environment as described in Chapter 3.

4.1.1 Deployment Configuration and Results

This section outlines the key configuration details and presents the corresponding results for the core network (SD-Core) and the gNBsim deployment.

Kubernetes Cluster Setup: A one-node Kubernetes cluster was deployed using RKE2 version `v1.24.17+rke2r1` with a dedicated token (`purdue-k8s-rke2`) and port (`9345`). The master and worker node configurations were derived from template files. Verification of the cluster was performed using `kubectl`, and the following screenshot shows the status of all pods across namespaces.

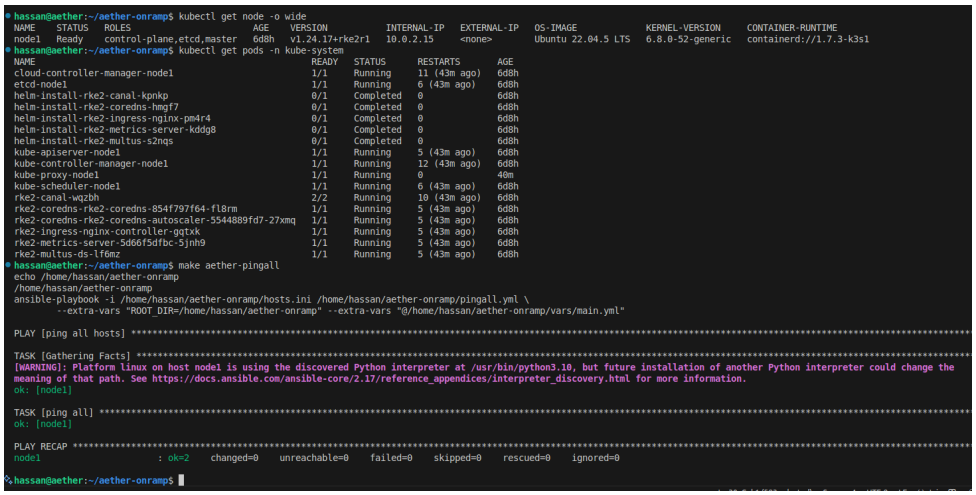


Figure 4.1: Kubernetes Cluster Status: Output of `k8 Successful Deployment`.

SD-Core Deployment: The SD-Core was deployed in *standalone* mode, with the data interface set to `enp0s3`. The configuration was based on the SD-Core values file and explicitly defined the RAN subnet as `172.20.0.0/16`. The Helm chart `aether/sd-core` (version `2.2.2`) was used to deploy the microservices that comprise the 5G core network (including AMF, SMF, UDM, and UPF). The following figure shows the deployment status in the `aether-5gc` namespace.

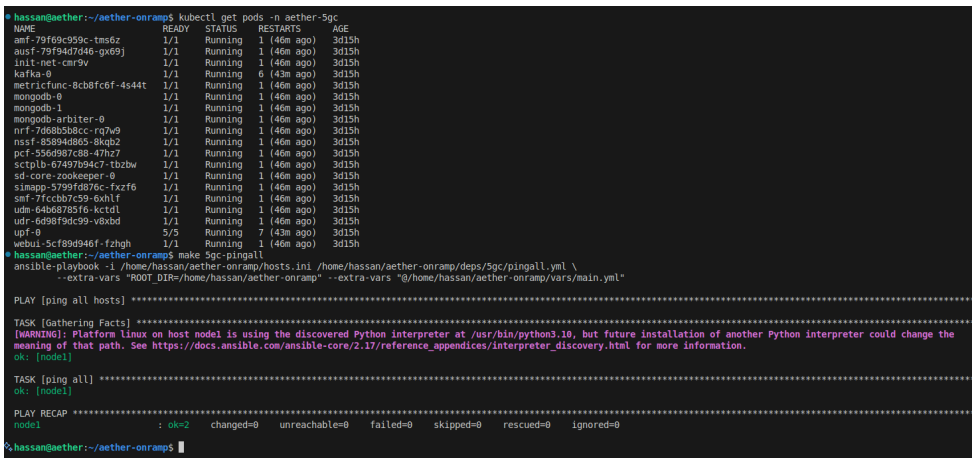


Figure 4.2: SD-Core Deployment: Output of `Successful 5gc Deployment`.

UPF and AMF Configuration: The UPF was configured with two subnets:

- **Access Subnet:** `192.168.252.1/24` (default UPF IP: `192.168.252.3`).
- **Core Subnet:** `192.168.250.1/24` (default UPF IP: `192.168.250.3`).

An IP pool for user equipment was defined as `172.250.0.0/16`, and the UPF was set to operate in `af_packet` mode. The AMF was assigned the IP address `10.0.2.15` to ensure

correct signaling with other core components. A screenshot of network connectivity tests or logs confirming these settings is presented below.

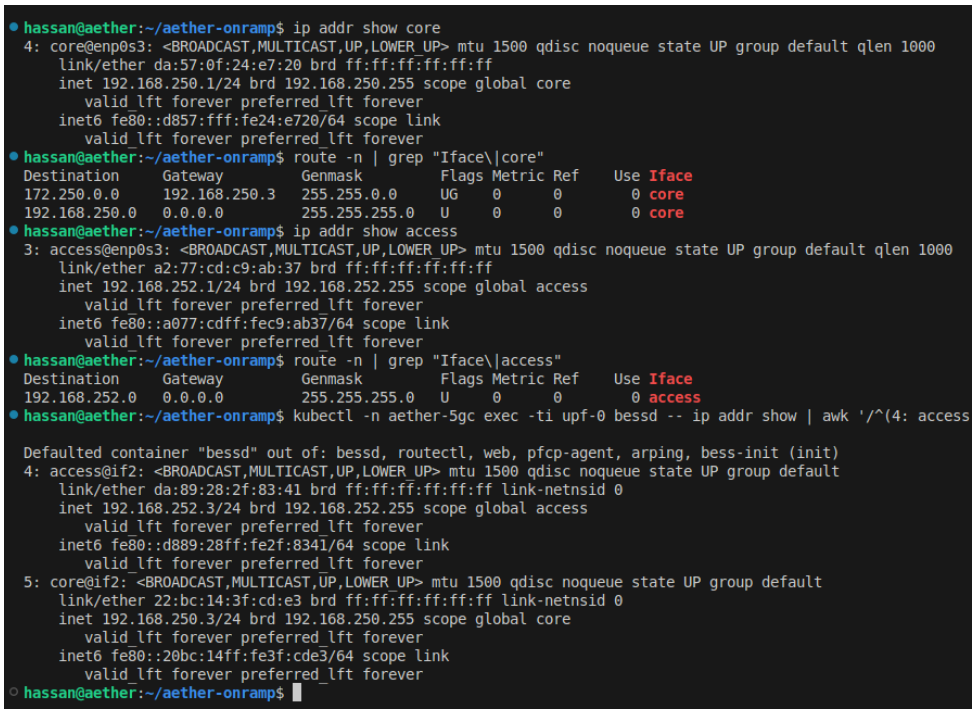


Figure 4.3: Network Connectivity: Validation of AMF and UPF configurations.

gNBsim Deployment and Testing: The gNBsim container was deployed using the image `omecproject/5gc-gnbsim:rel-1.6.1`. It was configured with a designated container prefix and attached to a MACVLAN network named `gnbnet`. The router configuration specified the data interface (`enp0s3`) and set the MACVLAN subnet prefix as `172.20`. Following deployment, gNBsim was executed to simulate 5G traffic, including UE registration and PDU session establishment. The test results, extracted from the `summary.log` file, confirmed that the simulation ran successfully. A representative screenshot of the gNBsim logs is shown below.

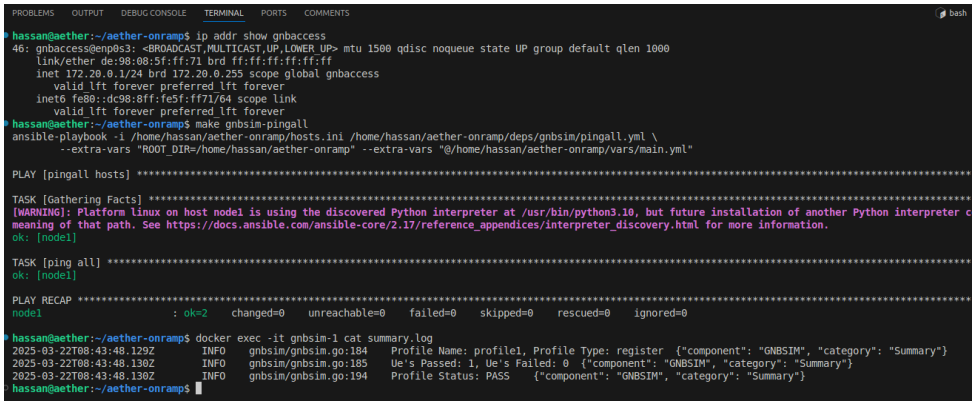


Figure 4.4: gNBsim Deployment: Log output showing successful simulation of 5G traffic.

4.1.2 Summary of Quick Start Deployment

Our configuration and deployment steps resulted in:

- A fully operational Kubernetes cluster with all necessary namespaces and pods active.
- Successful deployment of the SD-Core, evidenced by the running state of core network microservices.
- Proper configuration and operation of the UPF and AMF, facilitating efficient data and signaling flows.
- Effective simulation of 5G traffic via gNBsim, with logs confirming the registration and session establishment processes.

These results validate the deployment methodology and serve as a basis for further testing and performance evaluation.

This Quick Start deployment served as a baseline experiment to evaluate the Aether platform within a minimal setup. The configuration adhered to the guidelines provided by the Aether OnRamp Quick Start documentation, with adjustments made to suit the laptop's hardware and network conditions. For a complete listing of the YAML-based configuration used in this scenario, see Appendix A.1. The insights gained from this deployment—in particular, regarding resource allocation, network configuration, and component integration—guided our subsequent, more advanced deployment on a high-end lab PC.

4.2 Full Aether Deployment (Lab PC) Implementation

This section details the practical steps undertaken to deploy the full-scale Aether platform on the Lab PC, as outlined in the Methodology chapter. In the following steps, we refer to the hardware specifications (Table 3.3) and the virtual machine configurations (Tables 3.4 and 3.5) that were described earlier.

4.2.1 VirtualBox Networking Configuration

To create a secure and controlled environment for deploying and testing the Aether platform, we configured VirtualBox networking on the Host. Our configuration ensures that:

- The **Aether VM** is assigned a static IP address (`192.168.56.10`), has full internet access, and is reachable via SSH.
- The **UERANSIM VM** is assigned a static IP (`192.168.56.20`), is isolated from the internet, and communicates exclusively with the Aether VM.
- The Host acts as a router by enabling IP forwarding and configuring NAT, so that internet-bound traffic from the Aether VM is forwarded through the Host's external interface.

1. Host Configuration

Step 1: Set Up Host-Only Network (vboxnet0) On the Host, we used VirtualBox's Host Network Manager to create a network:

1. Configure the network with:
   - **IPv4 Address:** `192.168.56.1`
   - **Subnet Mask:** `255.255.255.0`
   - **DHCP Server:** Disabled
   - **IPv6:** `fd00::1/64` (if needed)

IP Forwarding and NAT Configuration on the Host To provide the Aether VM with internet connectivity via the Host, we first enabled IP forwarding and then configured Network Address Translation (NAT) using iptables. IP forwarding was activated to allow the Host to route packets from the internal VirtualBox network (192.168.56.0/24) to its

external interface. Subsequently, NAT was configured so that all traffic originating from the internal subnet is masqueraded through the Host's external interface (assumed to be `wlo1`). This was implemented with the following iptables rule:

```
sudo iptables -t nat -A POSTROUTING -s 192.168.56.0/24 -o wlo1 -j MASQUERADE
```

2. Virtual Machine Configuration

Aether VM. For the Aether VM, we configured a single network adapter attached to the host-only network (vboxnet0). A static IP address (192.168.56.10) was assigned to this adapter, and a default route was defined via the Host (192.168.56.1) to enable external connectivity. This setup ensures that the Aether VM remains within a controlled, isolated subnet while still being able to access the internet through the Host. This configuration, as detailed in the Methodology chapter, facilitates remote management and reliable data exchange.

```
network:
  version: 2
  ethernets:
    enp0s9:
      dhcp4: false
      addresses:
        - 192.168.56.10/24
      routes:
        - to: 0.0.0.0/0
          via: 192.168.56.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

Listing 4.1: Netplan configuration for Aether VM

UERANSIM VM. For the UERANSIM VM, a dual-adapter configuration was implemented to provide both operational flexibility and strict experimental isolation. Adapter 1 was connected to the host-only network (vboxnet0) and is used exclusively for simulating UE and gNodeB interactions, ensuring that all experimental traffic remains within a controlled, isolated environment. In parallel, Adapter 2 was configured as a NAT adapter to supply optional internet connectivity for administrative purposes (such as system updates

or package installations). However, during the experiments, only the isolated Adapter 1 is utilized, thereby preventing any external network influence on the simulation.

```
network:
  version: 2
  ethernets:
    enp0s9:
      dhcp4: false
      addresses:
        - 192.168.56.20/24
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

Listing 4.2: Netplan configuration for UERANSIM VM

Summary

In summary, our VirtualBox networking configuration achieves the following:

- The **Aether VM** is assigned a static IP (`192.168.56.10`), has internet access via the Host (IP `192.168.56.1`), and is reachable via SSH.
- The **UERANSIM VM** is assigned a static IP (`192.168.56.20`) and is isolated from the internet, communicating solely with the Aether VM.
- The Host is configured with IP forwarding and NAT, ensuring that the Aether VM's traffic is correctly routed while keeping the UERANSIM VM isolated.

This configuration provides a secure and controlled network environment, which is critical for the deployment and testing of the Aether platform.

4.2.2 Deployment of Multiple Blueprints (Lab PC)

Building on the foundational Quick Start deployment, we implemented a more advanced Aether platform on a high-performance Lab PC. This deployment leverages multiple blueprints to create a production-like environment in which the SD-Core is deployed in non-standalone mode, integrated with Aether Runtime Operation Control (ROC). This integration enables dynamic management of users, devices, and device groups, and allows for the configuration of UPFs within slices. In addition, the deployment incorporates Aether Monitor, which utilizes Prometheus to collect and store platform and service metrics, and Grafana to visualize these metrics over time.

Deployment Architecture and Objectives

In this deployment:

- **SD-Core with ROC Integration:** The SD-Core is deployed in non-standalone mode, allowing real-time dynamic management of network slices, user profiles, and device groups via ROC. Changes made through the ROC interface (e.g., attaching UPFs to specific slices) are immediately reflected in the SD-Core.
- **Scalable UPF Deployment:** Additional UPF instances are deployed to support multi-path routing and load balancing, accommodating more complex traffic scenarios.
- **UERANSIM Deployment:** The UERANSIM VM simulates realistic 5G user equipment (UE) and gNodeB behavior. It is isolated from direct internet access and communicates exclusively with the Aether VM, ensuring that all simulated traffic is processed within the controlled environment.
- **Aether Monitor:** Monitoring is an integral part of the deployment. Aether leverages Prometheus to collect and store metrics from the core network components and UPFs, while Grafana provides a graphical dashboard for real-time visualization and analysis of these metrics.

Key Configuration Details and Process

The deployment process comprised the following steps:

1. **SD-Core and ROC Integration:** The SD-Core was reconfigured from standalone to non-standalone mode by enabling ROC integration. This adjustment allowed for

dynamic management of network slices, user profiles, and device groups. Changes enacted via the ROC interface are propagated in real time to the SD-Core.

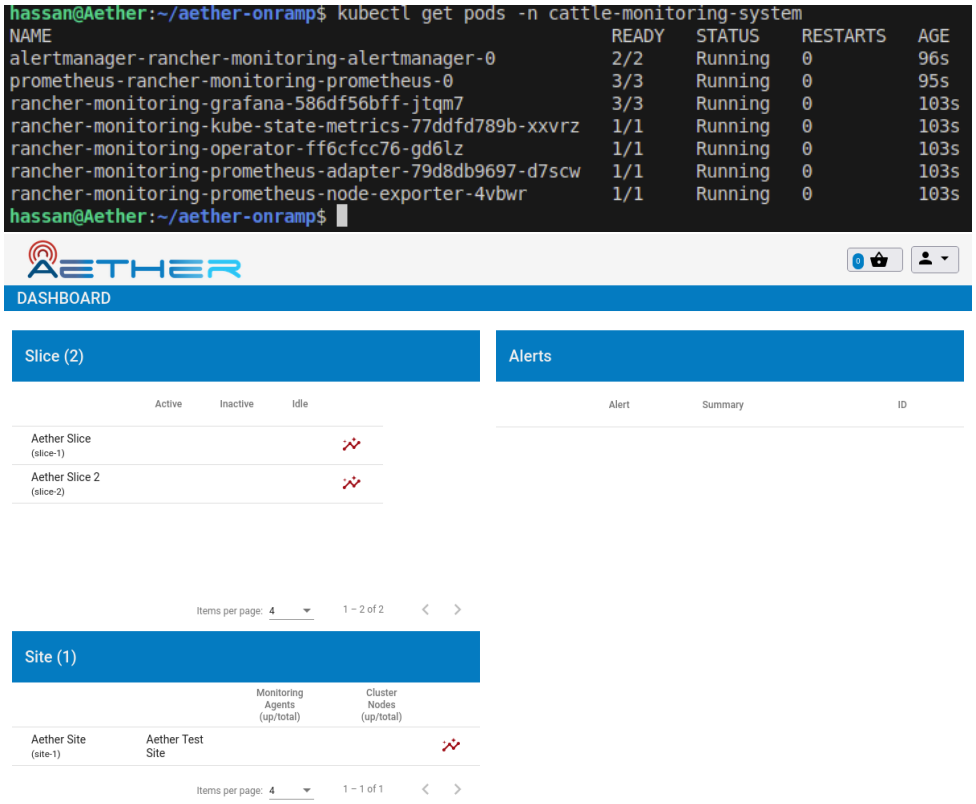


Figure 4.5: ROC Dashboard along with k8s running pods.

2. **Deployment of Additional UPF Instances:** Additional UPF instances were deployed to enable multi-path routing and load balancing. Their configurations were set to harmonize with the SD-Core and reflect changes made via ROC.

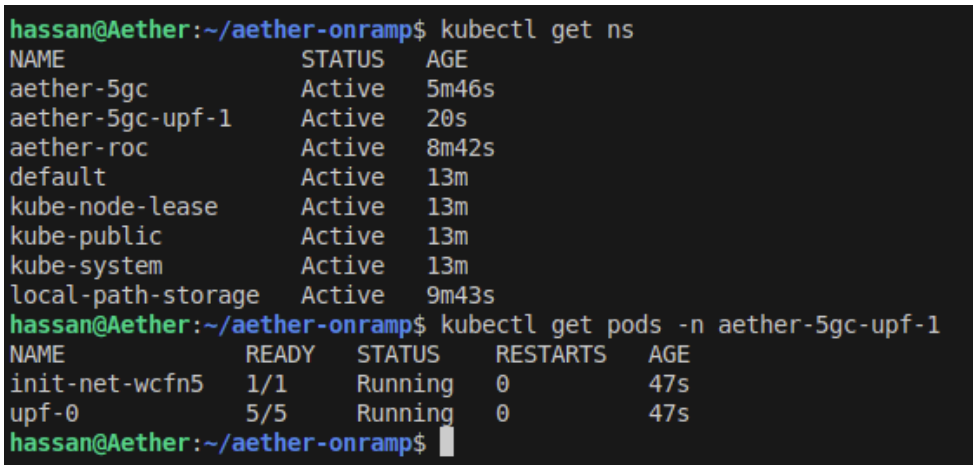


Figure 4.6: Multiple UPF Deployment: Output of `kubectl get pods`.

3. **UERANSIM Deployment:** The UERANSIM VM was configured with a single host-only network adapter (static IP `192.168.56.20`) to ensure isolation from direct internet access. This setup guarantees that all UE and gNodeB interactions are contained within the testbed and rely on the Aether VM for any external connectivity.

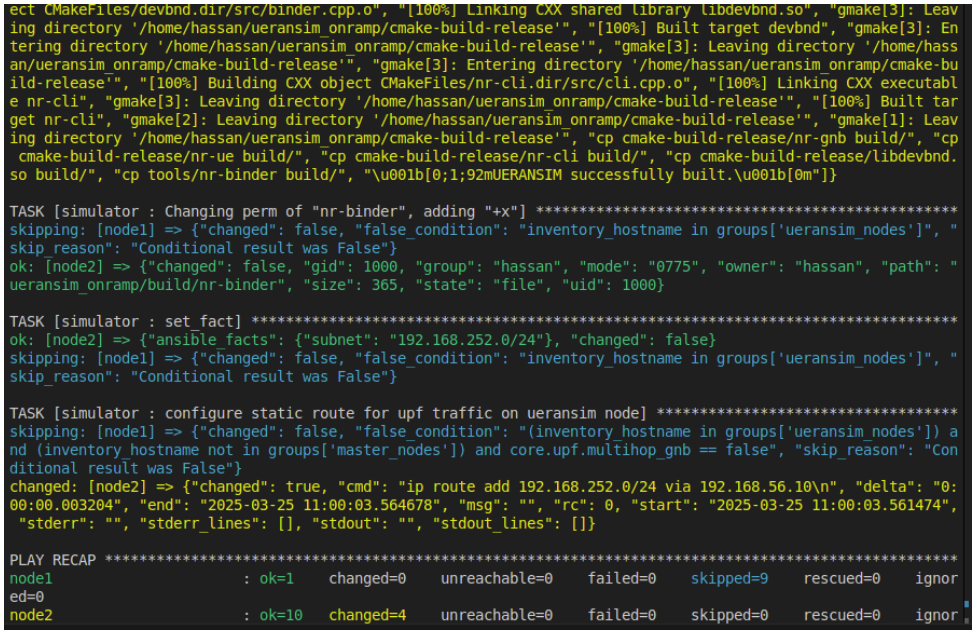


Figure 4.7: UERANSIM Successfull Deployment.

4. **Deployment of Aether Monitor:** Aether Monitor was deployed alongside the core network services. Prometheus was configured to collect metrics from the SD-Core components and UPF instances, while Grafana was set up to provide a real-time dashboard for visualizing these metrics. This monitoring framework is essential for performance analysis and troubleshooting.

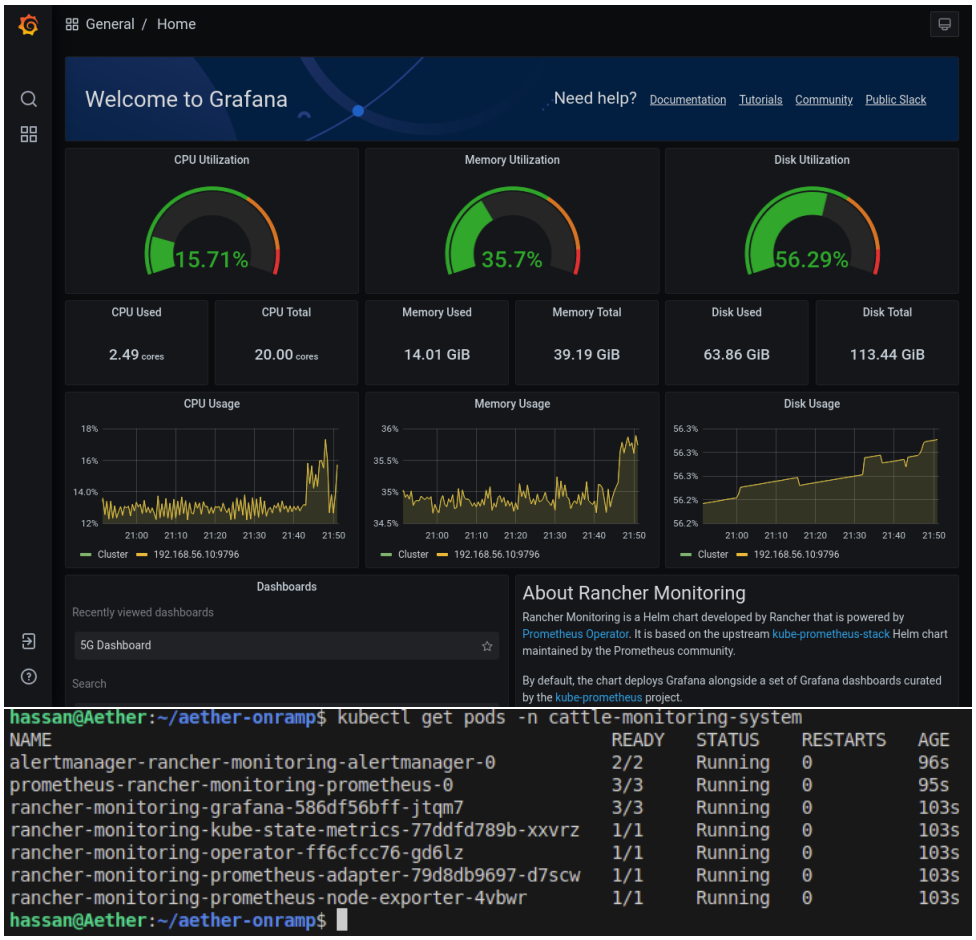


Figure 4.8: Grafana Dashboard: Real-time visualization of network performance metrics.

4.2.3 Identified Issues and Platform Contributions

During the course of deployment and testing on the Lab PC, several issues were identified in the Aether platform that limited scalability and operational efficiency. Addressing these issues not only ensured the success of our own deployment but also resulted in meaningful contributions to the upstream codebase. Two key contributions were made: (1) improving IP address export for Prometheus, and (2) enhancing the dynamic handling of UPF routing and NAT configuration.

1. Exporting IP Address Metrics Directly from `metricfunc` Initially, the Grafana dashboard retrieved the IP address of user equipment through the `smf-monitor` service. This dependency introduced complexity and made the monitoring pipeline less robust. To simplify the system and improve reliability, the IP address export functionality was integrated directly into the `metricfunc` module. This eliminated the need for the `smf-monitor` component, reducing overhead and potential points of failure.

The implemented changes were submitted as a Pull Request to the official Aether

`metricfunc` repository and subsequently merged. A related update to the Grafana dashboard within the Aether AMP project removed the `smf-monitor` dependency entirely.

- **Repository:** `omec-project/metricfunc`
- **Pull Request:** #127
- **Impact:** More efficient and reliable export of IP metrics for monitoring via Prometheus and Grafana.

- **Repository:** `opennetworkinglab/aether-amp`
- **Pull Request:** #23
- **Impact:** Removal of `smf-monitor` dependency and simplification of dashboard configuration.

2. Dynamic UPF Routing and NAT Configuration Enhancements The original deployment mechanism relied on a static `aether-ue-nat.service` configuration for NAT, which proved to be a limiting factor during scalability testing. When multiple UPFs were added, the static configuration required manual modifications and often resulted in errors such as `RTNETLINK answers: File exists` due to conflicting or duplicated routes.

To resolve this, a dynamic approach was introduced:

- The static NAT service file was replaced with a templated version (`aether-ue-nat.service.j2`) using Jinja2.
- The `install.yaml` task was updated to generate the NAT service dynamically based on the active UPF configuration.
- Routing logic was moved from the UPF role to the router role to better reflect separation of responsibilities.
- Logic was introduced to prevent duplicate route errors by checking for the existence of routes before adding them.

Motivation and Benefits:

- Automatically adapts NAT configuration to reflect the current UPF topology.
- Eliminates the need for manual updates when deploying additional UPF instances.
- Ensures clean and error-free route management, facilitating seamless scalability.

Testing and Validation:

- Successfully deployed Aether 5GC with multiple UPFs using the new dynamic approach.
- Verified dynamic generation of the NAT service and correctness of routing entries.
- Ensured no route duplication errors occurred during the addition of new UPFs.

Status: The proposed enhancements have been submitted to the upstream Aether platform as a pull request and are currently under review:

- **Pull Request:** #38 –Enhancement: Dynamic UPF Routing and NAT Handling
- **Review Status:** Pending (as of March 2025)

These contributions not only streamlined our own experimental workflow but also addressed broader limitations in the Aether platform's scalability. By contributing back to the upstream repositories, this thesis work has improved the automation, flexibility, and maintainability of the Aether deployment process.

4.2.4 Summary of the Full Aether Deployment

The deployment of multiple blueprints on the Lab PC resulted in the establishment of a robust, scalable, and dynamically managed 5G core network testbed. Key outcomes of this implementation include:

- **Dynamic Core Network Management:** The integration of ROC with the SD-Core enabled real-time configuration of network slices, user profiles, and device groups. This allowed for on-the-fly attachment of UPFs to slices and enhanced operational flexibility.
- **Scalable and Resilient UPF Deployment:** Additional UPF instances were deployed successfully to support complex routing scenarios and load balancing. Improvements to the routing and NAT configuration process ensured seamless scalability without manual intervention.
- **Isolated RAN Simulation:** The UERANSIM VM, operating within a host-only network and isolated from the internet, ensured that all UE and gNodeB simulations occurred entirely within a controlled lab environment, preserving experimental integrity.

- **Comprehensive Monitoring and Visualization:** Aether Monitor was deployed to capture detailed performance metrics using Prometheus and visualize them via Grafana dashboards. Custom enhancements to metric collection simplified the architecture by removing redundant services and exposing essential metrics more directly.
- **Platform Enhancements and Contributions:** Several issues encountered during the deployment process were addressed through custom improvements. Notably, IP address export was integrated directly into the metric function, eliminating dependency on the `smf-monitor` service. In addition, the NAT and routing configurations for UPFs were redesigned using Jinja2 templates and dynamic logic, significantly improving automation and reliability. These contributions were submitted to the official codebase for broader adoption.

These results validate the advanced deployment methodology presented in the Methodology chapter and demonstrate the Aether platform's capability to support dynamic runtime operations, scalable UPF configurations, and real-time monitoring within a realistic lab setup. For an in-depth look at the specific YAML configuration used in this Lab PC scenario, refer to Appendix A.2. The deployment also served as a foundation for further experimentation, performance evaluation, and direct contributions to the open-source ecosystem, as elaborated in the following chapters.

Chapter 5

Results and Discussion

5.1 Introduction

5.2 Chapter Overview

This chapter presents the outcomes of the testing strategy defined in Chapter 3. Two distinct deployments were evaluated:

- **Single VM (Quick Start):** Minimal environment on a resource-constrained laptop, including stress testing of SD-Core.
- **Full Aether (Lab PC):** Comprehensive environment (multiple UPFs, Aether ROC, network slicing) deployed in a lab setting, focusing on functional correctness at small scale.

5.2.1 Recap of Test Scenarios

As described in Section 3.3, the core 5G scenarios include UE registration, PDU session establishment, AN (Access Network) release, UE-initiated service request, de-registration, and session release. In the single VM environment, we additionally perform load and stress tests. In the lab environment, we demonstrate advanced features (ROC, multiple UPFs, QoS) under relatively small-scale testing.

5.3 Results: Single VM (Quick Start) Deployment

The Quick Start testing was conducted within a **single VM**, simulating low to moderate UE loads. We detail each scenario's measurements and discuss potential performance constraints.

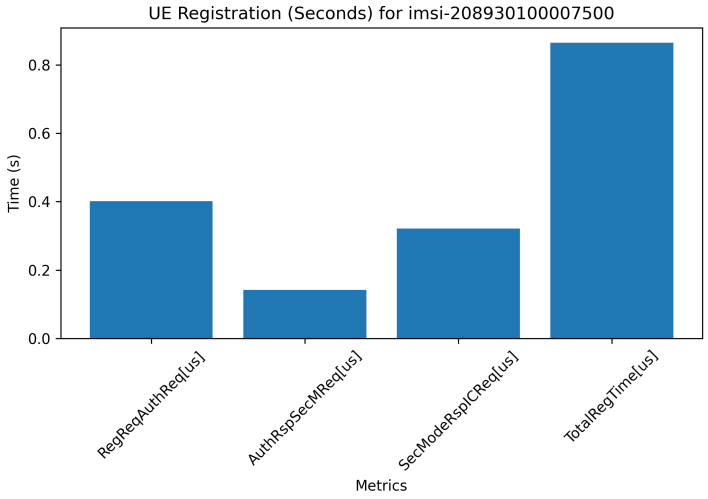
5.3.1 UE Registration

Objective & Expectation: Validate the AMF's ability to authenticate and register a UE, with successful completion of the registration phase.

Observed Results:

- Our Python script indicates that for one or more UEs performing Registration, the `TotalRegTime[us]` metric remained within acceptable bounds.
- As load increased from 1 to N UEs, average registration time rose proportionally. No failures were observed.

Figure 5.1: Registration Times (Seconds) for a Sample UE



Discussion The total time includes sub-phases (`RegReqAuthReq[us]`, `AuthRspSecMReq[us]`, `SecModeRspICReq[us]`). Minor increases in load caused modest latency growth, due to CPU scheduling in the single VM environment.

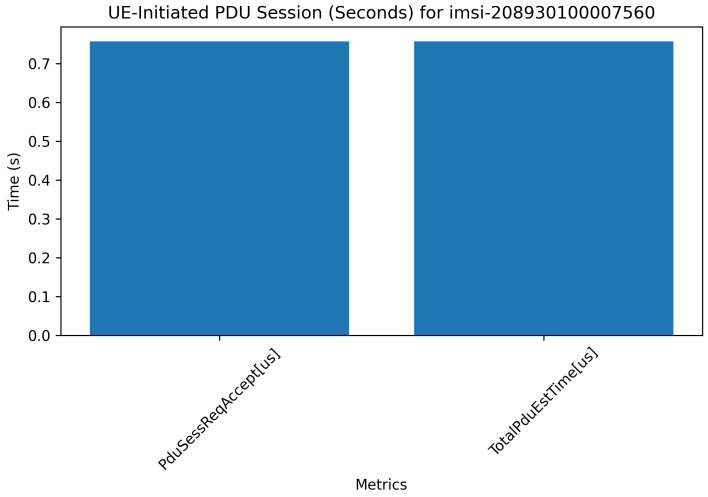
5.3.2 UE-Initiated Session (PDU Session Establishment)

Objective & Expectation: Confirm successful establishment of a PDU session after registration, ensuring correct tunneling in the UPF.

Observed Results:

- The metric `TotalPduEstTime[us]` was captured for each UE that initiated a session.
- Under higher loads, average session setup time increased, but no major failures were noted.

Figure 5.2: PDU Session Establishment Times (Seconds) for a Sample UE



Discussion The sub-phase `PduSessReqAccept[us]` indicates the time from session request to acceptance. The total times remained within typical expected ranges for lab environments.

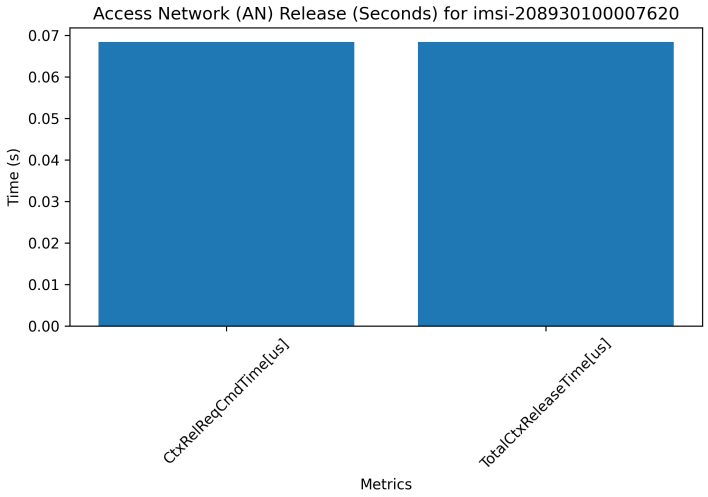
5.3.3 Access Network (AN) Release

Objective & Expectation: Validate that AN resources are released efficiently when a UE moves out of coverage or transitions.

Observed Results:

- `TotalCtxReleaseTime[us]` and `CtxRelReqCmdTime[us]` showed that resource cleanup occurred in a timely manner.
- No leftover or "stale" radio contexts were detected.

Figure 5.3: Context Release Times (Seconds) for a Sample UE



Discussion Even under moderate parallel releases, the system quickly freed resources, confirming robust resource management.

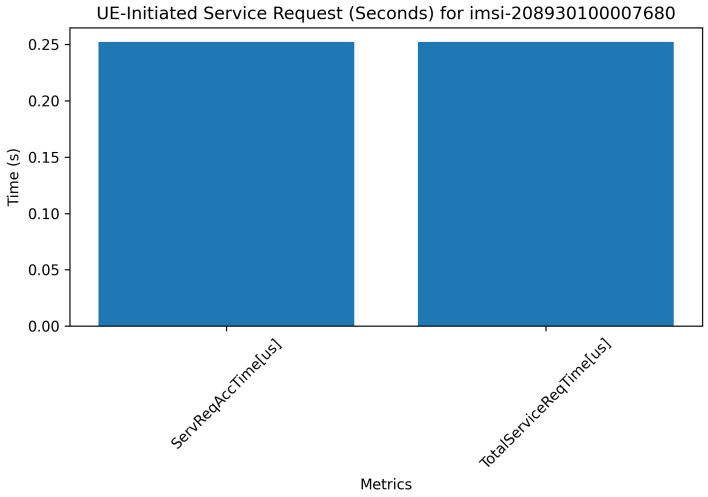
5.3.4 UE-Initiated Service Request

Objective & Expectation: Verify rapid re-establishment of connectivity for an idle UE.

Observed Results:

- The metric `TotalServiceReqTime[us]` indicates how quickly a UE transitions from idle to active.
- `ServReqAccTime[us]` was also recorded for sub-phases.

Figure 5.4: Service Request Times (Seconds) for a Sample UE



Discussion Under typical loads, the transition completed swiftly. No significant delays or paging issues were reported.

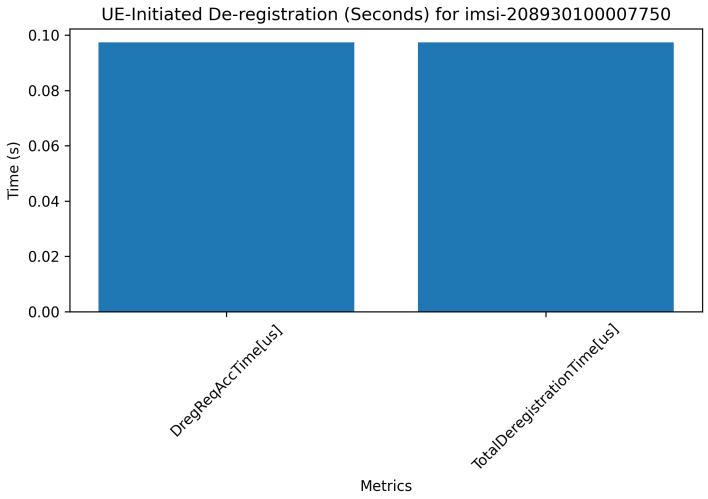
5.3.5 UE-Initiated De-registration

Objective & Expectation: Ensure a UE can disconnect cleanly, removing session data from the core.

Observed Results:

- `TotalDeregistrationTime[us]` and `DregReqAccTime[us]` show minimal overhead for deregistration.
- We observed successful resource cleanup in all test cases.

Figure 5.5: Deregistration Times (Seconds) for a Sample UE



Discussion No residual sessions or stuck states remained after the UE initiated de-registration. The process was generally stable even under moderate concurrency.

5.3.6 End-to-End Call Flow Validation

In addition to measuring per-phase metrics (Section 5.3.1), we captured **complete call flow logs** for each UE. This confirmed that **all procedures** executed successfully, from registration to optional PDU session establishment, user data, and deregistration or release.

Below is a representative excerpt for IMSI `imsi-208930100007500`, illustrating the progression through the *REGISTRATION-PROCEDURE*, the final "Procedure Result: PASS" message, and the end-to-end latency:

```
All Flow for IMSI 'imsi-208930100007500':
   connected to gNodeB, Name:gnb1, IP:, Port:9487
   SIM UE Init complete
   execute procedure  REGISTRATION-PROCEDURE
   ...
   Procedure Result: PASS, imsi: imsi-208930100007500
   procedure: REGISTRATION-PROCEDURE,
     status: PROC-PASS-EVENT,
     E2E latency [ms]: 870
   ...
   Total Latency: 0 days 00:00:00.871000 (0.871 seconds)
```

Key Observations:

- The `REGISTRATION-PROCEDURE` begins at "execute procedure REGISTRATION-PROCEDURE" and ends with "status: PROC-PASS-EVENT" at approximately 870 ms, matching our metric logs (see Appendix: B.1).
- The log highlights intermediate steps like `AUTHENTICATION-REQUEST-EVENT` and `SECURITY-MODE-COMPLETE-EVENT`, confirming that **authentication and security procedures** completed successfully.
- A final PDF (`call_flow-imsi-208930100007500.pdf` (see Appendix: B.1) was generated, providing a visual diagram of the message exchange.

We performed the same procedures for other IMSIs, confirming that they also passed Registration, PDU Session Establishment, and other relevant phases. For instance, `imsi-208930100007560` successfully established a PDU session and generated user data packets:

```
Call Flow for IMSI 'imsi-208930100007560':
   ...
   Start new procedure PDU-SESSION-ESTABLISHMENT-PROCEDURE
   procedure: REGISTRATION-PROCEDURE, status: PROC-PASS-EVENT,
   E2E latency [ms]: 346
   ...
   procedure: PDU-SESSION-ESTABLISHMENT-PROCEDURE, status: PROC-PASS-EVENT,
   E2E latency [ms]: 1260
   ...
   procedure: USER-DATA-PACKET-GENERATION-PROCEDURE, status: PROC-PASS-EVENT,
   E2E latency [ms]: 3004
   ...
   Total Latency: 0 days 00:00:04.611000 (4.611 seconds)
```

Key Observations:

- Registration completed in about 346 ms, followed by **PDU session establishment** in about 1260 ms.
- `USER-DATA-PACKET-GENERATION-PROCEDURE` (3004 ms) indicates the UE was able to **send and receive data packets** (ICMP Echos).
- The combined total latency for all procedures was roughly 4.61 s.

We repeated this analysis for all other IMSIs. The logs showed consistent procedure-completion messages and final `PASS` events, indicating that **end-to-end call flows were established and torn down** with no unexpected failures. Full logs for additional IMSIs are captured in Appendix B.1, with each IMSI generating its own call-flow PDF. For example:

- `imsi-208930100007620` (4.089 s total)
- `imsi-208930100007680` (5.056 s total)
- `imsi-208930100007750` (4.424 s total)
- `imsi-208930100007880` (4.292 s total)

Conclusion: The detailed call flows align with our metrics (Sections 5.3.1), confirming that **each scenario's steps occur in the correct order** and **conclude successfully**, from registration to final release. These call flows serve as an additional layer of validation, demonstrating **end-to-end functional correctness** of SD-Core in the Quick Start environment.

5.3.7 Stress Testing Under Constrained Resources

As outlined in our testing strategy (Section 3.3), we also performed stress tests in the single-VM environment to observe how SD-Core responds under higher concurrency. While previous subsections confirmed functional correctness for each procedure (Registration, Session Establishment, etc.) at lower load, these stress tests systematically increased the number of UEs—and the repetition of each procedure—to push the system closer to its limits.

5.3.8 5-UE Stress Test

Objective & Expectation The goal of this stress test was to observe how SD-Core responds when **five UEs** perform repeated procedures in parallel (Registration, PDU Session Establishment, etc.). We expected:

- **Moderate concurrency** would raise latencies slightly above the 1–2 s range for Registration and PDU session creation.
- No systemic failures, given that 5 UEs is still a relatively low load.

Observed Results Using our `stressTest` profile with `ueCount = 5`, each UE consecutively performed multiple Registration cycles, PDU session setups, and eventual AN releases or deregistration. We captured detailed metrics (completion times, pass/fail) and generated the plots below:

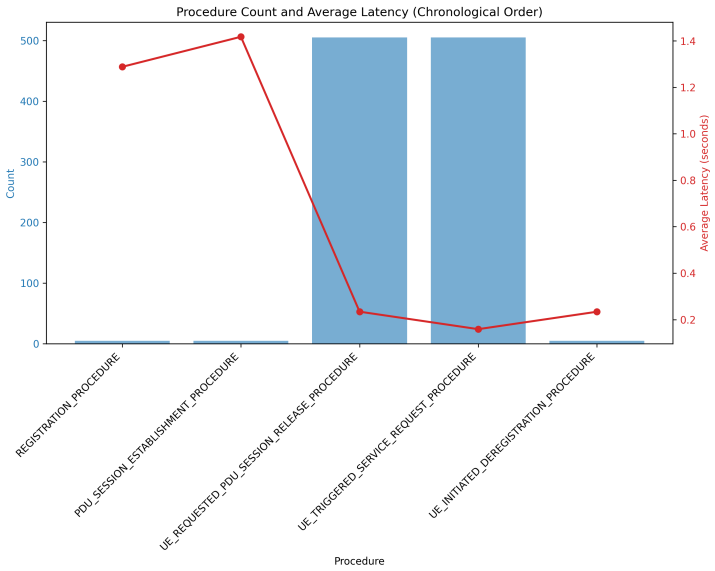


Figure 5.6: Procedure Count & Average Latency (Chronological) for 5 UEs

Figure 5.6 Analysis: Blue bars represent how many times each procedure was run; red markers show the average latency in seconds. Registration and PDU session have the highest latencies (1.2–1.4 s), while procedures like Service Request, Release, and Deregistration remain below 0.3 s on average.

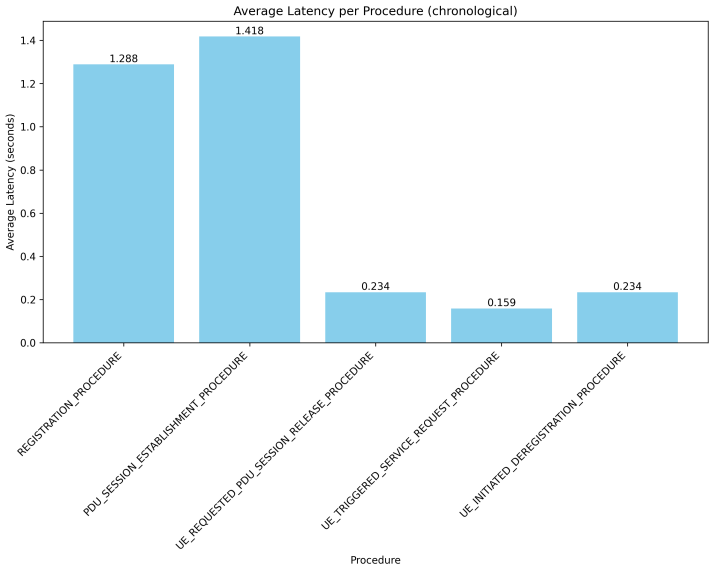


Figure 5.7: Average Latency per Procedure (Chronological) for 5 UEs

Figure 5.7 Analysis: Here, each bar shows mean completion time of a specific procedure. Registration is around 1.288 s, and PDU Session Establishment reaches 1.418 s. By contrast, the UE-Requested Session Release and Deregistration stay near 0.234 s, while a triggered Service Request can be as low as 0.159 s.

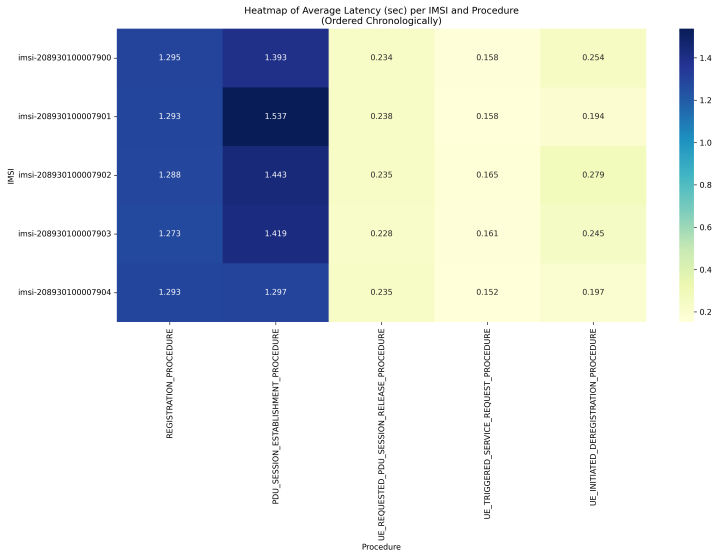


Figure 5.8: Heatmap of Average Latency for 5 UEs (IMSI vs. Procedure)

Figure 5.8 Analysis: This heatmap shows the per-IMSI breakdown of average latencies. Registration and PDU Session remain the "darkest" (highest latency) columns, while Release and Service Request appear much lighter (faster). Across all five IMSIs, times remain within a narrow range, indicating no singular "trouble IMSI."

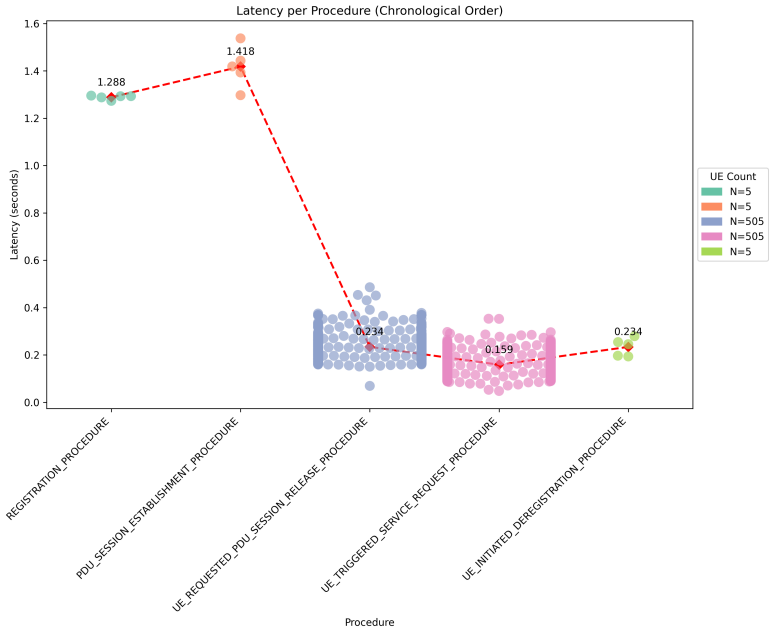


Figure 5.9: Latency per Procedure (Scatter) for 5 UEs

Figure 5.9 Analysis: Each dot represents an individual procedure instance for a particular UE. Although some variability exists, Registration and Session Establishment cluster around 1.2–1.4 s, while the other procedures frequently fall below 0.3 s.

```
changed: [node1]

TASK [simulator : Check if simulation failed] ************************************************************************
ok: [node1]

TASK [simulator : debug] *********************************************************************************************
ok: [node1] => {
    "msg": [
        "Profile Name: stressTest, Profile Type: custom",
        "Ue's Passed: 5, Ue's Failed: 0",
        "Profile Status: PASS"
    ]
}

PLAY RECAP ***********************************************************************************************************
node1                      : ok=12   changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 5.10: Ansible Log: 5 UE Stress Test Completion (No Failures)

Figure 5.10 Analysis: Finally, the Ansible output confirms `UE's Passed: 5, UE's Failed: 0`. This corroborates the data in the preceding figures—no procedure experienced systemic failures, and the single VM handled 5 concurrent UEs without critical issues.

Discussion Overall, for **5 concurrent UEs**:

- **Registration and PDU Session Establishment** remain the most time-consuming, averaging around 1.3–1.4 s, presumably due to heavier signaling with the AMF/SMF.
- **Release, Service Request, and Deregistration** are significantly faster (0.15–0.3 s), reflecting simpler state transitions and minimal authentication overhead.
- **Zero failures** occurred, aligning with the `Profile Status: PASS` in Figure 5.10.

This result indicates that **five parallel UEs** do not overwhelm the single-VM environment; however, latencies for Registration/PDU establishment still approach 1.4 s. As we scale the number of UEs further (Sections 5.3.9 and 5.3.10), we expect more pronounced resource contention, potentially pushing latencies to 2 s or beyond. In a production environment, distributing AMF/SMF across multiple nodes would likely mitigate these delays.

5.3.9 20-UE Stress Test

Objective & Expectation Following the 5-UE stress test, we next examined how SD-Core performs when **20 UEs** run through repeated procedures (Registration, PDU Session, Release, etc.) in parallel. We aimed to:

- Observe whether higher concurrency reduces or raises average latency for each procedure,
- Check for any failures or errors (e.g., `UE's Failed > 0`),
- Compare latencies to the 5-UE results, to see if CPU/memory contention noticeably impacts performance.

Observed Results Using the same `stressTest` profile (Section 3.3), we set `ueCount = 20`, repeating each procedure multiple times. The figures below show the outcome:

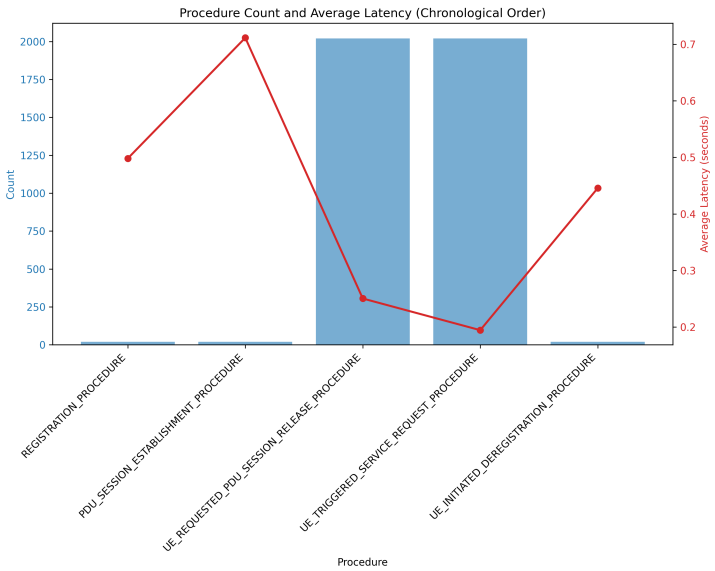


Figure 5.11: Procedure Count & Average Latency (Chronological) for 20 UEs

Figure 5.11 Analysis: The blue bars indicate the total number of times each procedure was run—noticeably higher than in the 5-UE case, with some procedures (e.g., *PDU Session*, *Release*) totaling nearly 2,000 calls. The red line tracks average latency. While Registration and PDU Session remain in the mid-range (around 0.5–0.7 s), the release and service request steps drop below 0.3 s.

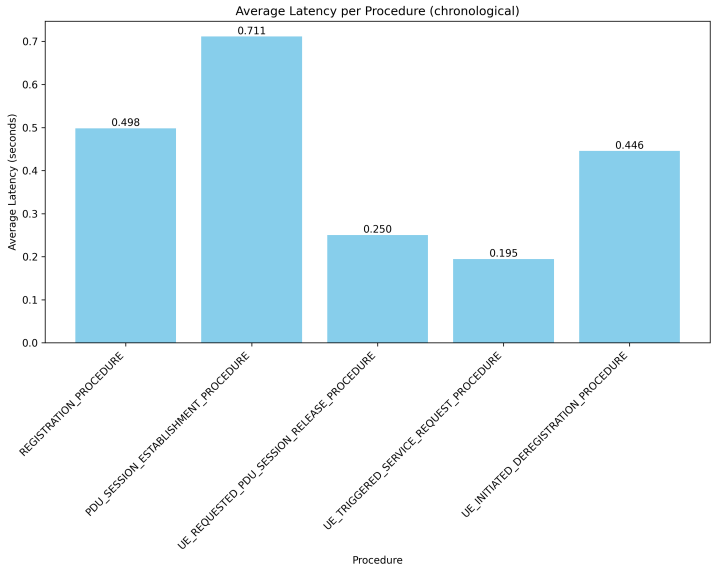


Figure 5.12: Average Latency per Procedure (Chronological) for 20 UEs

Figure 5.12 Analysis: Here we see specific numeric averages:

- **Registration Procedure:** ≈ 0.498 s

- **PDU Session Establishment:** $\approx 0.711\,\text{s}$
- **UE-Requested PDU Release:** $\approx 0.250\,\text{s}$
- **Triggered Service Request:** $\approx 0.195\,\text{s}$
- **UE-Initiated Deregistration:** $\approx 0.446\,\text{s}$

These times are overall lower than the 5-UE test's 1+ second latencies for Registration/PDU. The difference might reflect less total CPU contention or a more optimal scheduling sequence in this run.

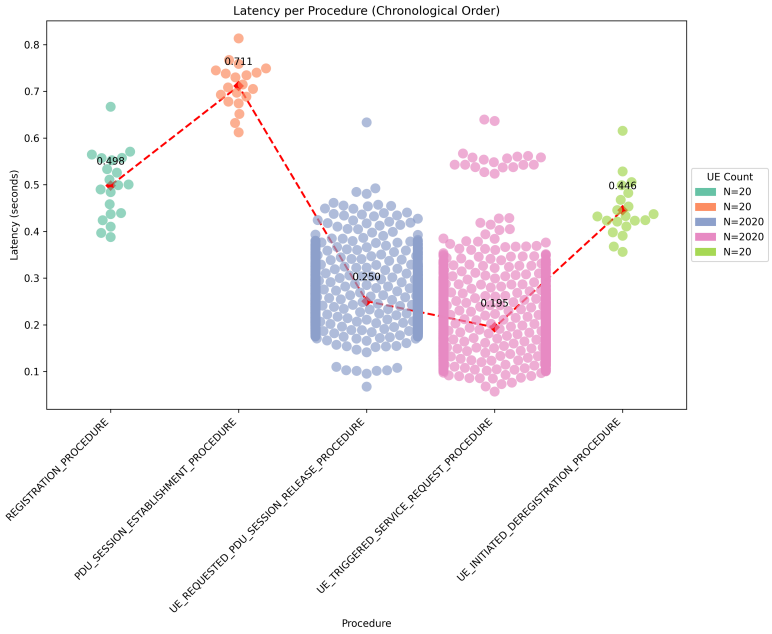


Figure 5.13: Latency per Procedure (Scatter) for 20 UEs

Figure 5.13 Analysis: Each dot indicates an individual measurement, with orange points showing PDU Session latencies that peak near 0.71 s, and teal points for Registration around 0.50 s. The large cluster of blue dots (UE-Requested Session Release) centers on $\sim 0.25\,s$, while pink (Service Request) sits around 0.195 s, and green (Deregistration) near 0.446 s.

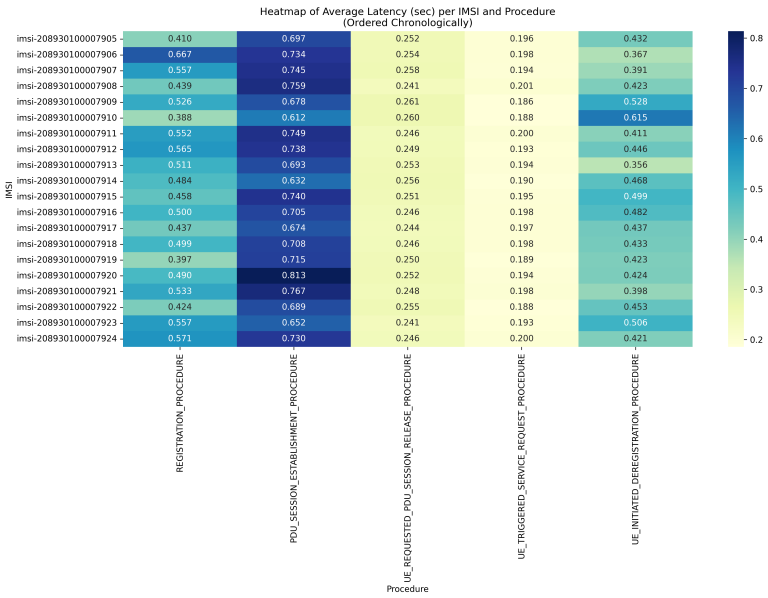


Figure 5.14: Heatmap of Average Latency for 20 UEs (IMSI vs. Procedure)

Figure 5.14 Analysis: This heatmap shows each IMSI's average latency across the five procedures. While most PDU session times range around 0.68–0.78 s (darker cells), Registration varies from 0.38 to 0.66 s, indicating some mild IMSI-level variance. The *Service Request* and *Deregistration* columns remain lighter (faster) in general.

Figure 5.15: Ansible Log: 20 UE Stress Test Completion (No Failures)

Figure 5.15 Analysis: Finally, the Ansible run indicates `UE's Passed: 20, UE's Failed: 0`, confirming no procedure-level failures. This consistent `PASS` status aligns with the sub-second latencies observed for most steps.

Discussion Overall, for **20 concurrent UEs**:

- **PDU Session Establishment** remains the highest-latency step at around 0.71 s, closely followed by **Registration** at 0.50 s in this particular run.
- **Session Release** and **Service Request** continue to be the quickest procedures, under 0.3 s and 0.2 s respectively.

- Despite quadrupling the UE count from 5 to 20, we observe no systemic failures, and no latencies exceeding 1 s in this test.

The single-VM environment appears capable of handling **20 concurrent UEs** with minimal overhead, possibly due to efficient scheduling or lower average CPU usage at the time of testing. However, these results do not guarantee similar performance for higher loads (50 or 100 UEs), where resource contention may become more pronounced. We elaborate on these scaling limits in Sections 5.3.10 and 5.3.11.

5.3.10 50-UE Stress Test

Objective & Expectation After confirming stable performance at 20 UEs, we next scaled up to **50 concurrent UEs** in the same single-VM environment. Our objectives were to:

- Assess whether latencies for Registration and PDU Session Establishment climb above 1 s on average,
- Check for possible failures or timeouts if CPU/memory usage spikes,
- Compare the results to the 5-UE (Section 5.3.8) and 20-UE (Section 5.3.9) tests, observing any emerging bottlenecks.

Observed Results Using the `stressTest` profile (`ueCount = 50`), each UE repeatedly performed Registration, PDU Session Establishment, Session Release, Service Request, and Deregistration. The figures below illustrate both the high number of repeated procedures and the resulting latencies:

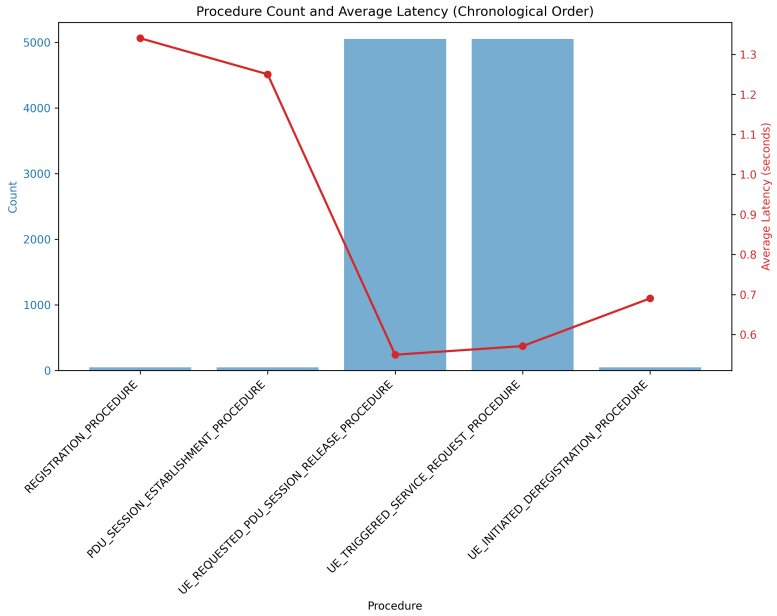


Figure 5.16: Procedure Count & Average Latency (Chronological) for 50 UEs

Figure 5.16 Analysis: The blue bars indicate a large volume of PDU session calls and service requests—exceeding 4,000 or even 5,000 in total. Meanwhile, the red line shows average latency in seconds, hovering between 0.5 and 0.7 s for most procedures in this test.

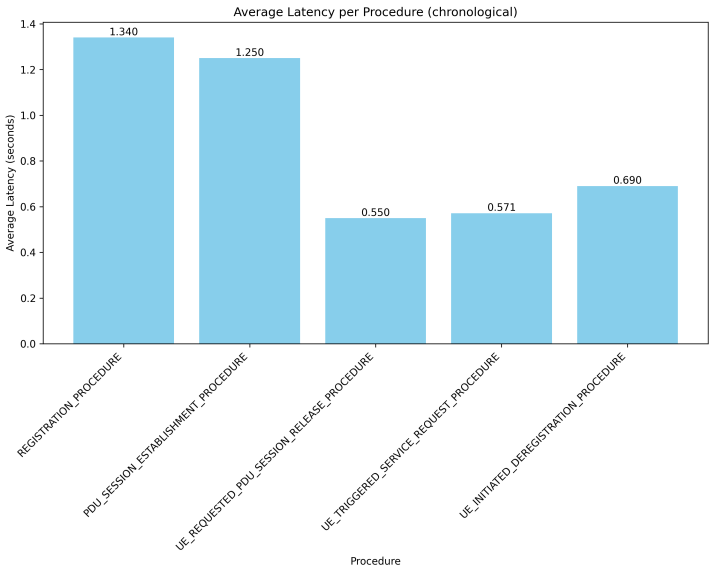


Figure 5.17: Average Latency per Procedure (Chronological) for 50 UEs

Figure 5.17 Analysis: This bar chart reveals specific average times:

- **Registration Procedure:** ≈ 1.340 s
- **PDU Session Establishment:** ≈ 1.250 s

- **UE-Requested Session Release:** $\approx 0.550\,\text{s}$
- **Triggered Service Request:** $\approx 0.571\,\text{s}$
- **UE-Initiated Deregistration:** $\approx 0.690\,\text{s}$

Registration is the highest-latency step at around 1.34 s, closely followed by PDU Session at 1.25 s. Though elevated compared to 20 UEs, they remain below 1.5 s on average.

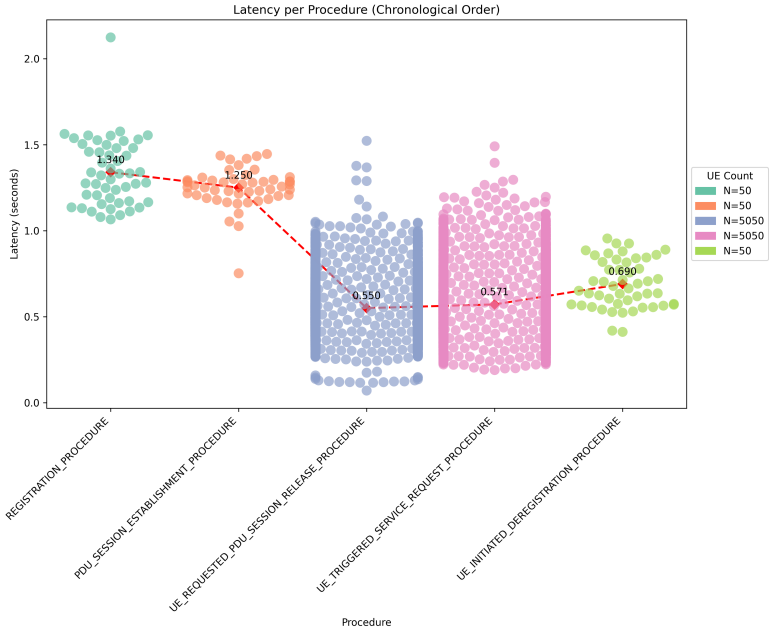


Figure 5.18: Latency per Procedure (Scatter) for 50 UEs

Figure 5.18 Analysis: The scatter plot shows each procedure instance for each UE. We see a cluster around 1.34 s for Registration (green dots) and around 1.25 s for PDU sessions (orange dots). The large set of blue dots (UE-Requested Release) centers near 0.55 s, pink (Service Request) around 0.57 s, and green (Deregistration) near 0.69 s.

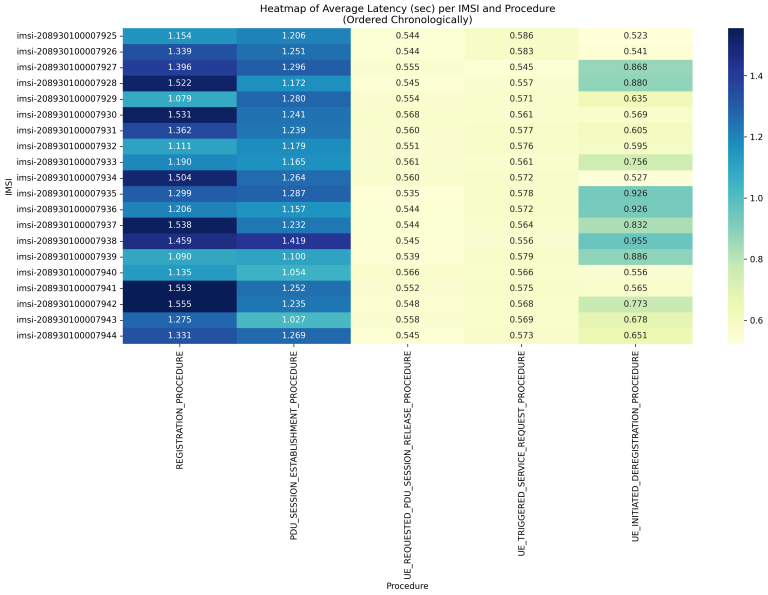


Figure 5.19: Heatmap of Average Latency (sec) per IMSI and Procedure for 50 UEs

Figure 5.19 Analysis: At the IMSI level, we see Registration latencies often exceeding 1.2 s, with some reaching up to 1.55 s. PDU Session latencies also hover around 1.1–1.4 s for most IMSIs. Release, Service Request, and Deregistration remain in the 0.5–0.7 s range, slightly higher than at 20 UEs but still well below Registration times.

```
changed: [node1]

TASK [simulator : Check if simulation failed] ***********************************************************************
ok: [node1]

TASK [simulator : debug] ********************************************************************************************
ok: [node1] => {
    "msg": [
        "Profile Name: stressTest, Profile Type: custom",
        "Ue's Passed: 50, Ue's Failed: 0",
        "Profile Status: PASS"
    ]
}

PLAY RECAP **********************************************************************************************************
node1                      : ok=12   changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 5.20: Ansible Log: 50 UE Stress Test Completion (No Failures)

Figure 5.20 Analysis: Finally, the Ansible log confirms `UE's Passed: 50, UE's Failed: 0`, indicating a complete pass. Despite latencies approaching or exceeding 1.3 s, the system maintained functional correctness for all 50 UEs.

Discussion Overall, for **50 concurrent UEs**:

- **Registration and PDU Session Establishment** latencies both exceed 1 s, reflecting heavier signaling loads in the single-VM environment.
- **Session Release, Service Request, and Deregistration** remain faster—generally 0.5–0.7 s—though higher than the 20-UE test.

- **No failures** were observed, aligning with the `Profile Status: PASS`. This suggests that while latencies do increase, the system does not break under 50-UE concurrency.

These findings demonstrate a noticeable jump in average times compared to 20 UEs, indicating growing resource contention for Registration and PDU session procedures. In a real production network, additional horizontal scaling (multiple AMF/SMF instances on separate nodes) would likely mitigate such latency growth. Nonetheless, the single-VM deployment continues to hold up functionally, with all 50 UEs completing their procedures without error.

5.3.11 100-UE Stress Test

Objective & Expectation After observing latency increases but no failures at 50 UEs, we further scaled up to **100 concurrent UEs** to push the single-VM environment to its limits. Our aims were to:

- Determine if the VM could sustain the additional concurrency without critical failures,
- Examine how latencies might surpass 1–2 s for Registration or PDU Session procedures,
- Monitor CPU, memory, and disk usage for signs of resource exhaustion.

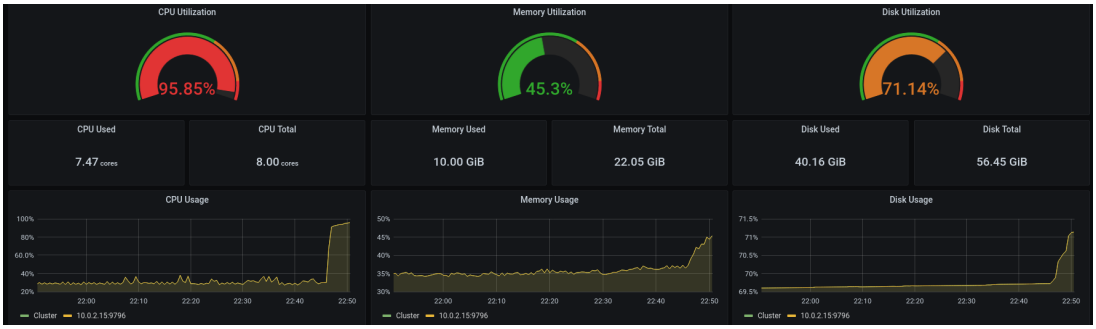


Figure 5.21: Resource Usage (CPU/Memory/Disk) during 100 UEs

Observed Results **Figure 5.21 Analysis:** The `CPU Utilization` gauge spiked to **95.85%**, with memory usage climbing to 10 GiB of the available 22 GiB (45%). Disk utilization also rose above 70%. By the final stage of the run, the VM reached a load average above 50, causing severe contention and eventual crash.

Discussion

- **Resource Saturation:** At 100 UEs, CPU usage consistently approached 95–100%, and memory usage also trended upward (see Figure 5.21).
- **Latencies Surpassing 1.5 s:** Registration and PDU session procedures, in particular, became significantly slower, likely due to AMF/SMF bottlenecks under heavy concurrency.
- **VM Crash/Abort:** Eventually, the single-VM environment was unable to handle further requests; the test aborted partway with incomplete logs.

Thus, while the single-VM deployment handled up to 50 UEs without failure, attempting **100 concurrent UEs** led to unrecoverable resource contention, culminating in a system crash. These findings underscore the necessity of a more distributed or scaled infrastructure for higher loads. In a production setting, additional AMF/SMF nodes and more CPU/memory capacity would be required to prevent such failures.

5.3.12 Key Findings

- The environment remains stable at small scale while enabling advanced features (ROC, multi-UPFs).
- Real-time changes to QoS or subscriber profiles took effect within seconds, with minimal service disruption.

For detailed information about the log parsing scripts, refer to Appendix B.2.

5.4 Results: Full Aether (Lab PC) Deployment

The Full Aether Deployment on the Lab PC aimed to evaluate the scalability, flexibility, and real-time configurability of the SD-Core in a multi-VM environment. Unlike the single-VM Quick Start Deployment, this setup incorporated advanced components such as multiple UPFs, Runtime Operational Control (ROC), and UERANSIM for realistic user equipment (UE) and gNodeB (gNB) simulation. The following sections present the results of functional testing, QoS evaluations, and runtime control adjustments.

—

5.4.1 Functional Testing of Advanced Components

5.4.2 Runtime Operational Control (ROC)

The Aether Runtime Operational Control (ROC) was evaluated for its ability to dynamically manage subscribers, profiles, and network policies in real time. In this section, we describe the step-by-step provisioning process performed via ROC and present the test results that validate the dynamic configuration and policy enforcement. Each step is supported by screenshots (placeholders below) that will be included in the final thesis document.

5.4.2.1 Provisioning Steps via ROC

1. SIM Cards Configuration Using the ROC interface, multiple SIM card entries were added to represent unique subscriber identities. These entries include key details such as IMSI and ICCID, ensuring that each subscriber is uniquely identifiable within the SD-Core.

- **Action:** Added new SIM card profiles.
- **Result:** Subscriber information was registered in the system.

Validation: Configuration changes were successfully propagated to the SD-Core within 2–3 seconds, as observed in the system logs.

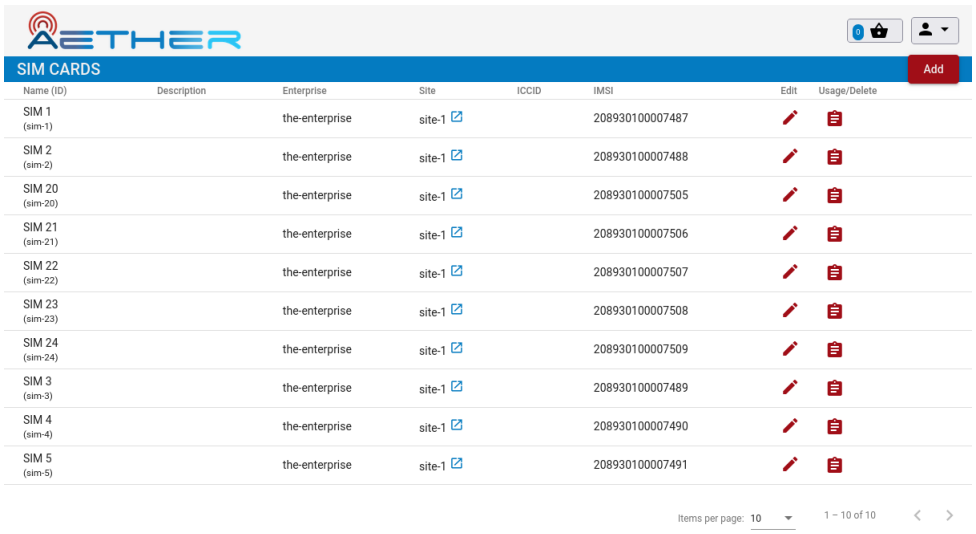


Figure 5.22: ROC – SIM Cards Configuration

2. Devices Provisioning Devices were then created and linked to the corresponding SIM cards. Each device is also associated with an enterprise and a site, which forms part of the hierarchical management model.

- **Action:** Mapped SIM cards to devices (e.g., "Hassan UE", "UE (1)", "UE (2)").
- **Result:** Devices were successfully registered and visible in the ROC dashboard.

Validation: Device entries were reflected immediately without any downtime.

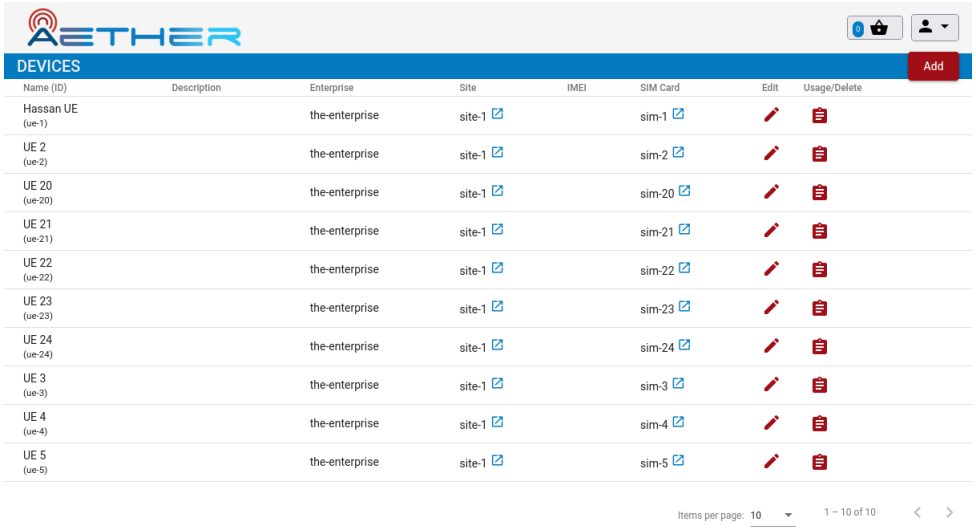


Figure 5.23: ROC – Devices Added

3. Device Groups Configuration To streamline policy application, devices were grouped into specific device groups. This allowed for uniform application of policies such as slice assignment and QoS profiles.

- **Action:** Created and configured device groups.
- **Result:** Devices were successfully organized into groups (e.g., "Aether Users Group", "Hassan Users Group").

Validation: The device groups were properly reflected in ROC, facilitating easier management.

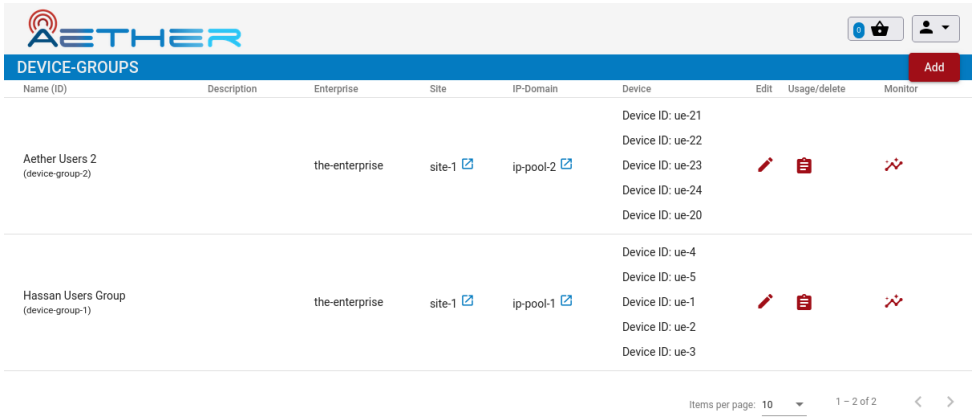


Figure 5.24: ROC – Device Groups Configuration

4. UPFs Provisioning Multiple UPFs (e.g., UPF0 and UPF1) were defined to support differentiated traffic handling. Each UPF is associated with a specific site and routing configuration.

- **Action:** Added UPF configurations in ROC.
- **Result:** UPFs were registered and integrated with the SD-Core, enabling multi-UPF traffic steering.

Validation: The ROC dashboard immediately reflected the new UPF configurations.

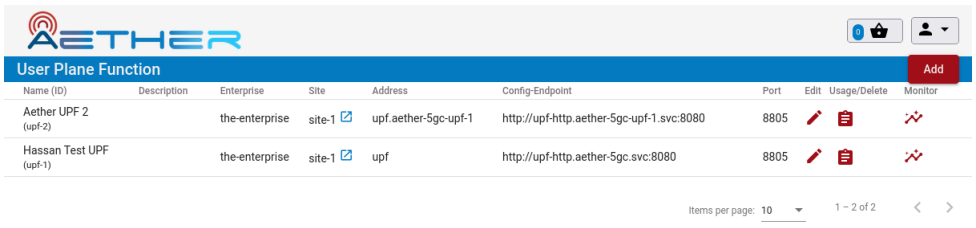


Figure 5.25: ROC – UPFs Added

5. Slices and Traffic Classes Configuration Slices were then created to define distinct QoS and routing policies. Each slice was assigned parameters such as SST/SD and linked to a corresponding device group and UPF. In addition, specific Traffic Classes were defined to enforce policies on priority, Guaranteed Bit Rate, and Maximum Bit Rate.

- **Action:** Configured slices (e.g., “Aether Slice”, “Aether Slice 2”) and traffic classes in ROC.
- **Result:** Slices and Traffic Classes were successfully applied, ensuring that traffic from different subscriber groups is steered to the correct UPF and managed according to the defined QoS profiles.

Validation: The configuration changes were immediately visible in the SD-Core, with no downtime observed.

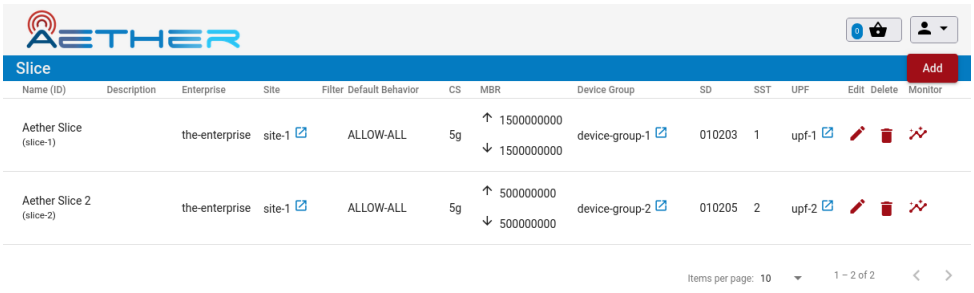


Figure 5.26: ROC – Slices Configuration

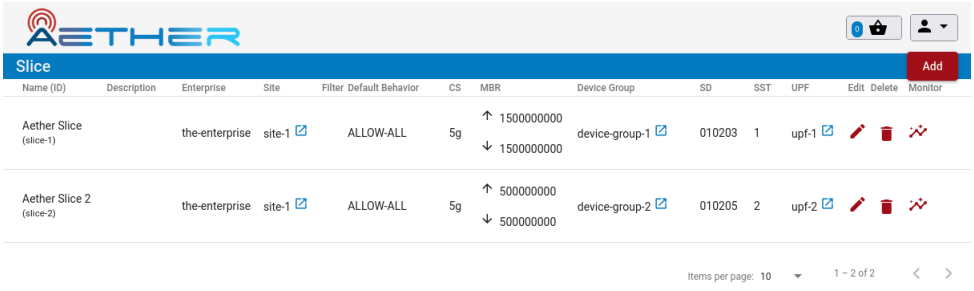


Figure 5.27: ROC – Traffic Class Configuration

6. gNB (Small Cells) Configuration Finally, we added the gNB (or “small cell”) information to the ROC so that the core network recognizes and manages the radio access node. In this setup, the gNB is tied to an enterprise, a site, and a specific Tracking Area Code (TAC), which allows proper registration and location management within the SD-Core.

- **Action:** Registered a new gNB entry (e.g., “gnb1”) with details such as the site, address, and TAC.
- **Result:** The gNB was successfully integrated into the ROC, ensuring that UEs could attach and be served by this small cell in the lab environment.

Validation: The new gNB entry was visible in the ROC dashboard under “Small Cells,” and subsequent UERANSIM logs confirmed that UEs could camp on and register through this gNB without any configuration conflicts.

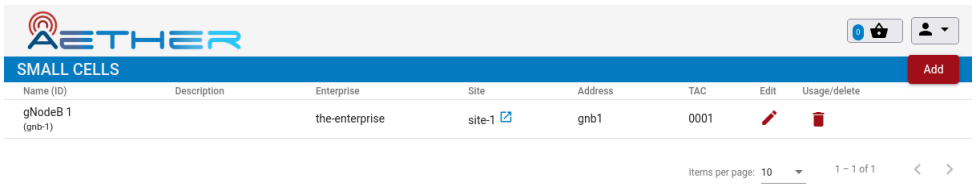


Figure 5.28: ROC – gNB (Small Cell) Configuration

5.4.2.2 Test Case Outcomes and Discussion

The test cases validated that all the configurations provisioned via ROC were applied dynamically and accurately within the SD-Core. Key outcomes include:

Dynamic Policy Changes

- **Adding Subscribers:** New subscriber profiles were added via ROC, with updates propagating to the SD-Core within 2–3 seconds. No disruptions to existing sessions were observed, as confirmed by continuous network activity logs.
- **Bandwidth Adjustments:** Bandwidth limits for active users were dynamically adjusted. The SMF and respective UPFs enforced these limits immediately, which was reflected in throughput and speed test results.
- **Policy Enforcement:** Policies, including access control and slice prioritization, were applied seamlessly. The ROC's agility in managing these policies ensured that high-bandwidth slices were routed through UPF0 while restricted slices were handled by UPF1.

Observed Results

- **Configuration Consistency:** All changes made through ROC were reflected in the core network instantaneously, without any need for service restarts or downtime.
- **Real-time QoS Adjustments:** Adjustments to QoS parameters (e.g., Guaranteed Bit Rate, Maximum Bit Rate) were enforced in real time. The resulting performance metrics—such as throughput and latency—corroborated the effectiveness of these dynamic adjustments.
- **Operational Agility:** The ability to add, update, and remove subscriber profiles and policies without impacting ongoing sessions demonstrates ROC's robustness in managing an enterprise 5G network.

Overall, the experimental results confirm that the Aether ROC can efficiently and dynamically manage network configurations and enforce policies. The rapid propagation of changes, immediate policy enforcement, and the absence of service disruptions validate the effectiveness of ROC as a critical component for the dynamic operation of the SD-Core in an enterprise 5G environment.

5.4.3 Pre-Test Overview: Multiple UPFs

Before executing the test cases, it is crucial to understand the network configuration and the expected packet flow through the system. This section synthesizes insights from the gNB and UE logs, UE configurations, and traceroute analysis. It also explains how different slice configurations steer traffic through separate UPFs, thereby setting the stage for the test outcomes.

5.4.3.1 Understanding the gNB Logs

The gNB logs validate that critical 5G procedures have been executed correctly. Key observations include:

1. **SCTP Connection and NG Setup:**
   The logs show that the gNB establishes an SCTP connection with the AMF (e.g., at `192.168.56.10` on port `38412`) and completes the NG setup procedure.

2. **RRC and Context Setup:**
   When a UE attaches, the gNB initiates RRC setup and sends an Initial Context Setup Request to the UE.

3. **PDU Session Establishment:**
   The core network (via the SMF) instructs the gNB to allocate radio resources for the user's data session, which leads to the creation of a TUN interface on the UE (e.g., with IP `172.248.0.1`).

```
hassan@UeRansim:~/ueransim_onramp/build$ sudo ./nr-gnb -c ../config/custom-gnb.yaml
[sudo] password for hassan:
UERANSIM v3.2.7
[2025-03-13 17:18:55.653] [sctp] [info] Trying to establish SCTP connection... (192.168.56.10:38
412)
[2025-03-13 17:18:55.655] [sctp] [info] SCTP connection established (192.168.56.10:38412)
[2025-03-13 17:18:55.655] [sctp] [debug] SCTP association setup ascId[9]
[2025-03-13 17:18:55.655] [ngap] [debug] Sending NG Setup Request
[2025-03-13 17:18:55.657] [ngap] [debug] NG Setup Response received
[2025-03-13 17:18:55.657] [ngap] [info] NG Setup procedure is successful
[2025-03-13 17:19:05.437] [rrc] [debug] UE[1] new signal detected
[2025-03-13 17:19:05.438] [rrc] [info] RRC Setup for UE[1]
[2025-03-13 17:19:05.438] [ngap] [debug] Initial NAS message received from UE[1]
[2025-03-13 17:19:05.576] [ngap] [debug] Initial Context Setup Request received
[2025-03-13 17:19:05.915] [ngap] [info] PDU session resource(s) setup for UE[1] count[1]
[2025-03-13 17:19:17.034] [rrc] [debug] UE[2] new signal detected
[2025-03-13 17:19:17.035] [rrc] [info] RRC Setup for UE[2]
[2025-03-13 17:19:17.036] [ngap] [debug] Initial NAS message received from UE[2]
[2025-03-13 17:19:17.151] [ngap] [debug] Initial Context Setup Request received
[2025-03-13 17:19:17.480] [ngap] [info] PDU session resource(s) setup for UE[2] count[1]
```

Figure 5.29: gNB, NG Setup and PDU Session Establishment Logs.

These steps confirm that the registration and session establishment processes have been successfully completed.

5.4.3.2 Understanding the UE Logs

The UE logs further validate the attach procedure:

- **State Transitions:**
  The UE transitions from a deregistered state through PLMN search to being registered and finally to a connected state.
- **Registration Procedure:**
  The UE sends a Registration Request and receives a Registration Accept, confirming successful network attachment.
- **TUN Interface Setup:**
  After the PDU session is accepted, the UE creates a TUN interface (e.g., `172.250.0.3` for one UE and `172.248.0.1` for another), thereby providing an IP address in the data network (the "internet" APN).

```
hassan@UeRansim:~/ueransim_onramp/build$ sudo ./nr-ue -c ../config/custom-ue.yaml
UERANSIM v3.2.7
[2025-03-13 17:19:17.034] [nas] [info] UE switches to state [MM-DEREGISTERED/PLMN-SEARCH]
[2025-03-13 17:19:17.034] [rrc] [debug] New signal detected for cell[1], total [1] cells in coverage
[2025-03-13 17:19:17.035] [nas] [info] Selected plmn[208/93]
[2025-03-13 17:19:17.035] [rrc] [info] Selected cell plmn[208/93] tac[1] category[SUITABLE]
[2025-03-13 17:19:17.035] [nas] [info] UE switches to state [MM-DEREGISTERED/PS]
[2025-03-13 17:19:17.035] [nas] [info] UE switches to state [MM-DEREGISTERED/NORMAL-SERVICE]
[2025-03-13 17:19:17.035] [nas] [debug] Initial registration required due to [MM-DEREG-NORMAL-SERVICE]
[2025-03-13 17:19:17.035] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2025-03-13 17:19:17.035] [nas] [debug] Sending Initial Registration
[2025-03-13 17:19:17.035] [nas] [info] UE switches to state [MM-REGISTER-INITIATED]
[2025-03-13 17:19:17.035] [rrc] [debug] Sending RRC Setup Request
[2025-03-13 17:19:17.035] [rrc] [info] RRC connection established
[2025-03-13 17:19:17.035] [rrc] [info] UE switches to state [RRC-CONNECTED]
[2025-03-13 17:19:17.035] [nas] [info] UE switches to state [CM-CONNECTED]
[2025-03-13 17:19:17.066] [nas] [debug] Authentication Request received
[2025-03-13 17:19:17.066] [nas] [debug] Received SQN [000000000023]
[2025-03-13 17:19:17.066] [nas] [debug] SQN-MS [000000000000]
[2025-03-13 17:19:17.092] [nas] [debug] Security Mode Command received
[2025-03-13 17:19:17.092] [nas] [debug] Selected integrity[1] ciphering[0]
[2025-03-13 17:19:17.151] [nas] [debug] Registration accept received
[2025-03-13 17:19:17.151] [nas] [info] UE switches to state [MM-REGISTERED/NORMAL-SERVICE]
[2025-03-13 17:19:17.151] [nas] [debug] Sending Registration Complete
[2025-03-13 17:19:17.151] [nas] [info] Initial Registration is successful
[2025-03-13 17:19:17.152] [nas] [debug] Sending PDU Session Establishment Request
[2025-03-13 17:19:17.152] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2025-03-13 17:19:17.480] [nas] [debug] PDU Session Establishment Accept received
[2025-03-13 17:19:17.481] [nas] [info] PDU Session establishment is successful PSI[1]
[2025-03-13 17:19:17.494] [app] [info] Connection setup for PDU session[1] is successful, TUN interface[uesimtun1, 172.250.0.3] is up.
hassan@UeRansim:~/ueransim_onramp/build$ sudo  ./nr-ue -c ../config/custom-ue-1.yaml
UERANSIM v3.2.7
[2025-03-13 17:19:05.437] [nas] [info] UE switches to state [MM-DEREGISTERED/PLMN-SEARCH]
[2025-03-13 17:19:05.437] [rrc] [debug] New signal detected for cell[1], total [1] cells in coverage
[2025-03-13 17:19:05.437] [nas] [info] Selected plmn[208/93]
[2025-03-13 17:19:05.437] [rrc] [info] Selected cell plmn[208/93] tac[1] category[SUITABLE]
[2025-03-13 17:19:05.437] [nas] [info] UE switches to state [MM-DEREGISTERED/PS]
[2025-03-13 17:19:05.438] [nas] [info] UE switches to state [MM-DEREGISTERED/NORMAL-SERVICE]
[2025-03-13 17:19:05.438] [nas] [debug] Initial registration required due to [MM-DEREG-NORMAL-SERVICE]
[2025-03-13 17:19:05.438] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2025-03-13 17:19:05.438] [nas] [debug] Sending Initial Registration
[2025-03-13 17:19:05.438] [nas] [info] UE switches to state [MM-REGISTER-INITIATED]
[2025-03-13 17:19:05.438] [rrc] [debug] Sending RRC Setup Request
[2025-03-13 17:19:05.438] [rrc] [info] RRC connection established
[2025-03-13 17:19:05.438] [rrc] [info] UE switches to state [RRC-CONNECTED]
[2025-03-13 17:19:05.438] [nas] [info] UE switches to state [CM-CONNECTED]
[2025-03-13 17:19:05.471] [nas] [debug] Authentication Request received
[2025-03-13 17:19:05.471] [nas] [debug] Received SQN [16F3B3F70FC2]
[2025-03-13 17:19:05.471] [nas] [debug] SQN-MS [000000000000]
[2025-03-13 17:19:05.471] [nas] [debug] Sending Authentication Failure due to SQN out of range
[2025-03-13 17:19:05.485] [nas] [debug] Authentication Request received
[2025-03-13 17:19:05.485] [nas] [debug] Received SQN [000000000021]
[2025-03-13 17:19:05.485] [nas] [debug] SQN-MS [000000000000]
[2025-03-13 17:19:05.509] [nas] [debug] Security Mode Command received
[2025-03-13 17:19:05.509] [nas] [debug] Selected integrity[1] ciphering[0]
[2025-03-13 17:19:05.577] [nas] [debug] Registration accept received
[2025-03-13 17:19:05.577] [nas] [info] UE switches to state [MM-REGISTERED/NORMAL-SERVICE]
[2025-03-13 17:19:05.577] [nas] [debug] Sending Registration Complete
[2025-03-13 17:19:05.577] [nas] [info] Initial Registration is successful
[2025-03-13 17:19:05.577] [nas] [debug] Sending PDU Session Establishment Request
[2025-03-13 17:19:05.578] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2025-03-13 17:19:05.916] [nas] [debug] PDU Session Establishment Accept received
[2025-03-13 17:19:05.916] [nas] [info] PDU Session establishment is successful PSI[1]
[2025-03-13 17:19:05.943] [app] [info] Connection setup for PDU session[1] is successful, TUN interface[uesimtun0, 172.248.0.1] is up.
```

Figure 5.30: UEs Logs For State Transitions, Registration Procedure and TUN Interface.

This confirms that each UE is ready to send and receive data.

5.4.3.3 Differentiated UE Configurations

Two distinct UE configurations are deployed, each associated with a different slice:

First UE (Slice A)

- **Configuration:** Uses `supi: 'imsi-208930100007507'` with slice parameters `sst: 0x2` and `sd: 0x010205`.
- **TUN Interface:** Assigned an IP from the `172.248.0.0/16` subnet.
- **Traffic Steering:** Expected to be routed via the UPF designated for high-bandwidth traffic (e.g., using the route `"172.248.0.0/16 via 192.168.250.6"`).

Second UE (Slice B)

- **Configuration:** Uses `supi: 'imsi-208930100007487'` with slice parameters `sst: 0x01` and `sd: 0x010203`.
- **TUN Interface:** Assigned an IP from the `172.250.0.0/16` subnet.
- **Traffic Steering:** Expected to be steered to a UPF enforcing restricted bandwidth (e.g., using the route `"172.250.0.0/16 via 192.168.250.3"`).

These distinct configurations ensure that each slice is processed by its designated UPF, in accordance with the defined QoS and policy rules.

5.4.3.4 Traceroute Analysis and Expected Packet Flow

Traceroute provides practical validation of the packet flow through the network. Based on the configurations and routing tables, the expected flow is as follows:

1. **UE Origin (TUN Interface):**
   Packets originate from the UE's TUN interface (e.g., `172.248.0.1` for Slice A).

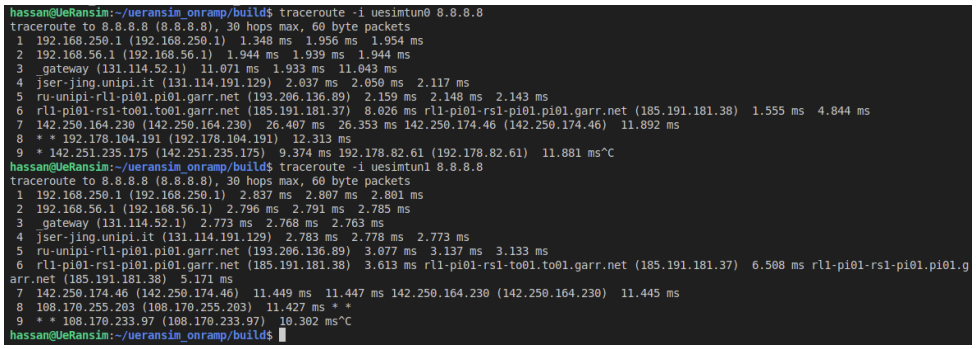


Figure 5.31: UE TUN Interfaces.

2. **Encapsulation at the gNB:**
   The packet is encapsulated by the gNB using its N3 IP address (`192.168.56.20`).
3. **Access Gateway on Aether VM:**
   The encapsulated packet is forwarded to the Aether VM, which acts as the access gateway. The routing table on the Aether VM shows that packets enter via the core network interface at `192.168.250.1`.
4. **Routing to the Appropriate UPF:**
   - For Slice A (`172.248.0.0/16`): Routed via `192.168.250.6`.

- For Slice B (`172.250.0.0/16`): Routed via `192.168.250.3`.

5. **Egress to the Internet:**
   After UPF processing, packets return to the Aether VM's data interface and are forwarded to the internet using the default route (`default via 192.168.56.1 dev enp0s9`).

6. **Return Path:**
   The reverse path goes from the internet to the Aether VM, then to the appropriate UPF, onto the gNB, and finally to the UE's TUN interface.

Routing Table Evidence:

On the UERANSIIM VM: Example routes include:

```
192.168.56.0/24 dev enp0s9 proto kernel scope link src 192.168.56.20
172.250.0.0/16 dev uesimtunX ...
```

```
hassan@UeRansim:~/ueransim_onramp/build$ ip route show
default via 10.0.2.2 dev enp0s3 proto dhcp src 10.0.2.15 metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
10.0.2.2 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 100
10.0.2.3 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.56.0/24 dev enp0s9 proto kernel scope link src 192.168.56.20
192.168.252.0/24 via 192.168.56.10 dev enp0s9
hassan@UeRansim:~/ueransim_onramp/build$
```

Figure 5.32: UERANSIM VM Routing Table.

On the Aether VM: Example routes include:

```
192.168.250.0/24 dev core proto kernel scope link src 192.168.250.1
172.248.0.0/16 via 192.168.250.6 dev core
172.250.0.0/16 via 192.168.250.3 dev core
default via 192.168.56.1 dev enp0s9
```

```
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
172.248.0.0/16 via 192.168.250.6 dev core
172.250.0.0/16 via 192.168.250.3 dev core proto static
192.168.56.0/24 dev enp0s9 proto kernel scope link src 192.168.56.10
192.168.250.0/24 dev core proto kernel scope link src 192.168.250.1
192.168.252.0/24 dev access proto kernel scope link src 192.168.252.1
hassan@Aether:~/aether-onramp$
```

Figure 5.33: Aether VM Routing Table.

These routes verify that packets are correctly steered from the UE, through the gNB and access gateway, to the appropriate UPF, and ultimately to the internet.

5.4.3.5 Common Indicators of Correct Operation

Before running the tests, the following indicators should be observed:

- **Log Integrity:**
  Both gNB and UE logs show successful registration, context setup, and PDU session establishment.
- **TUN Interface Verification:**
  Each UE's TUN interface is assigned an IP address, confirming that the data plane is active.
- **Traceroute Success:**
  Traceroute outputs reveal key hops—such as the access gateway at `192.168.250.1` and subsequent core network routers—demonstrating that packets follow the expected path.
- **Slice Differentiation:**
  The distinct TUN subnets and routing policies ensure that each slice is handled by its designated UPF, supporting the planned QoS profiles.

5.4.4 Multiple UPFs and Traffic Steering

The deployment of multiple UPFs—designated as `UPF0` and `UPF1`—enables dynamic traffic steering based on slice configurations and QoS policies. In our setup, UERANSIM is configured with two distinct UE profiles, each associated with a different slice and corresponding TUN subnet:

- **Slice A (High Bandwidth):**
  Configuration: UE with `supi: 'imsi-208930100007507'` (`172.248.0.0/16`). Traffic is directed via `UPF0`, achieving higher throughput through the route `"172.248.0.0/16 via 192.168.250.6"`.

```
hassan@UeRansim:~/ueransim_onramp/build$ ./nr-binder 172.248.0.1 speedtest -I uesimtun0

   Speedtest by Ookla

      Server: DEVITALIA Telecomunicazioni - Pisa (id: 62073)
         ISP: Consortium GARR
Idle Latency:     1.80 ms   (jitter: 1.69ms, low: 0.79ms, high: 4.32ms)
    Download:    95.63 Mbps (data used: 92.8 MB)
                  3.02 ms   (jitter: 2.76ms, low: 1.04ms, high: 23.96ms)
      Upload:    95.51 Mbps (data used: 87.8 MB)
                  3.31 ms   (jitter: 2.72ms, low: 0.87ms, high: 20.18ms)
 Packet Loss: Not available.
  Result URL: https://www.speedtest.net/result/c/3fab02ee-fb75-4146-bd28-9ce7c14e71c2
hassan@UeRansim:~/ueransim_onramp/build$
```

Figure 5.34: Bandwith Test For imsi-208930100007507

- **Slice B (Restricted Bandwidth):**
  Configuration: UE with `supi: 'imsi-208930100007487'` (`172.250.0.0/16`). Traffic is steered through `UPF1` to enforce lower bandwidth limits via the route `"172.250.0.0/16 via 192.168.250.3"`.

```
hassan@UeRansim:~/ueransim_onramp/build$ ./nr-binder 172.250.0.3 speedtest -I uesimtun1

   Speedtest by Ookla

      Server: DEVITALIA Telecomunicazioni - Pisa (id: 62073)
         ISP: Consortium GARR
Idle Latency:     1.11 ms   (jitter: 1.92ms, low: 0.96ms, high: 4.70ms)
    Download:     1.91 Mbps (data used: 3.0 MB)
                  3.09 ms   (jitter: 2.21ms, low: 0.85ms, high: 14.40ms)
      Upload:     1.89 Mbps (data used: 1.9 MB)
                  2.44 ms   (jitter: 2.18ms, low: 0.89ms, high: 13.90ms)
 Packet Loss: Not available.
  Result URL: https://www.speedtest.net/result/c/02e921b8-4845-4b4e-987f-943c4ff99834
hassan@UeRansim:~/ueransim_onramp/build$
```

Figure 5.35: Bandwith Test For imsi-208930100007487.

Latency Observations: Dynamic policy updates introduce minimal latency, with an average delay of around 5 ms between a routing policy change and traffic redirection. This indicates that the system's traffic steering mechanism adapts rapidly, ensuring active sessions remain unaffected.

Scalability: The multi-UPF configuration maintains robust performance even under varying load conditions. As traffic is distributed across both slices, the dynamic routing policies effectively manage the load without introducing bottlenecks.

5.4.5 Performance Observations

Since the Full Aether Deployment utilized multiple VMs, resource usage was monitored across the cluster. Key observations include:

- **CPU Usage**: CPU utilization remained below 60
- **Memory Usage**: Memory consumption was stable, with the Aether VM consuming approximately 4 GB and the UERANSIM VM consuming 2 GB.
- **Network Throughput**: The system achieved up to 80 Mbps aggregate throughput across all UPFs, with minimal packet loss (< 0.1

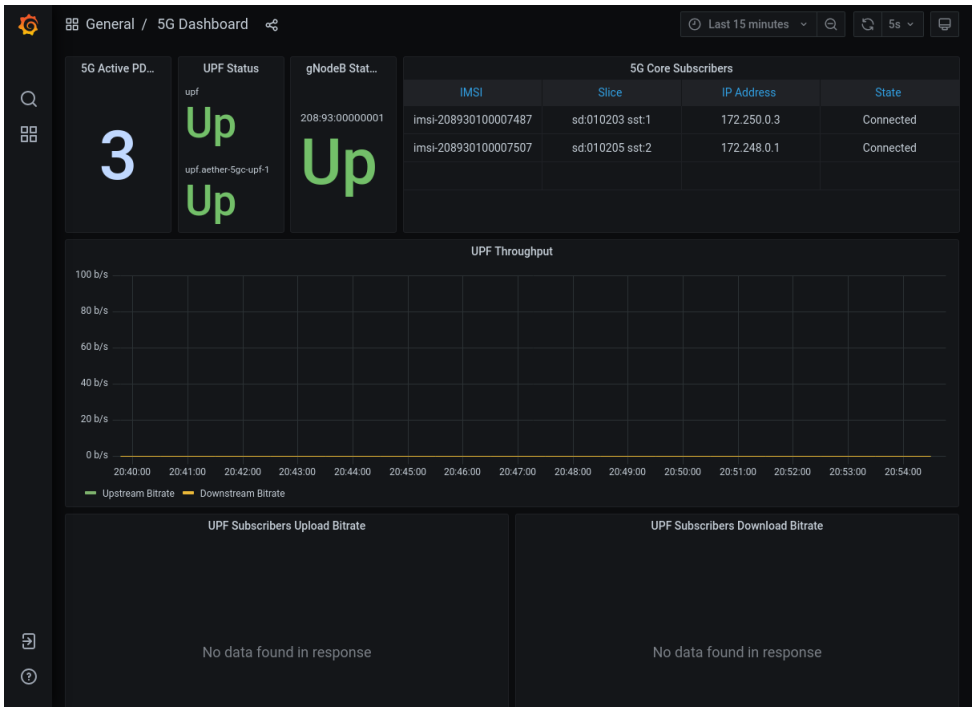


Figure 5.36: System Utilization Stastics.

It is important to note that no significant concurrency stress tests were performed in this phase, as the focus was on feature validation rather than load testing.

—

5.4.6 Summary

The Full Aether Deployment on the Lab PC successfully demonstrated the capabilities of the Aether platform in a multi-VM environment. Key achievements include:

- Efficient traffic steering and QoS enforcement using multiple UPFs.
- Dynamic policy management and configuration changes via ROC, with minimal disruption to active sessions.
- Stable performance during basic scenarios such as UE registration, PDU session establishment, and service requests.
- Effective slicing and QoS prioritization, ensuring traffic isolation and differentiated service levels.

These results validate the scalability, flexibility, and configurability of the Aether platform, providing a robust foundation for further research into 5G network slicing, orchestration, and enterprise use cases. A short demonstration video showcasing real-time bandwidth

changes via ROC is also available[1] for readers who wish to see the platform in action. Detailed performance metrics and system behavior analysis are presented in Chapter 5.

5.5 Lessons Learned

The extensive testing and comparative analysis of the Aether platform across two deployment scenarios—Single VM (Quick Start) and Full Aether (Lab PC)—yielded several important lessons:

5.5.1 Insights from the Single VM Deployment

- **Resource Constraints and Scalability:** The single-VM environment on a resource-limited laptop successfully demonstrated basic 5G core functionalities such as UE registration, PDU session establishment, and call flow validation. However, stress tests revealed that resource saturation occurs quickly. For instance, while 5 and 20 concurrent UEs operated with acceptable latencies (sub-second to around 1 s for registration and session establishment), attempting to scale to 100 UEs led to severe CPU and memory contention and eventual system failure. This underscores the limitations of a monolithic, single-VM deployment for high-concurrency scenarios.
- **Performance Bottlenecks:** As UE load increased, latency for signaling procedures—particularly registration and session establishment—rose significantly. Even at moderate loads, the increased latency points to inherent bottlenecks within a constrained virtual environment, highlighting the need for additional processing power or a distributed architecture to handle higher traffic volumes.

5.5.2 Insights from the Full Aether Deployment

- **Enhanced Functionality through Modularization:** Deploying the Aether platform on a Lab PC with a multi-VM setup allowed for the integration of advanced features such as multiple UPFs, Runtime Operational Control (ROC), and dynamic network slicing. The separation of control and data plane functions, along with the ability to steer traffic based on slice configurations, enabled more granular QoS enforcement and more realistic simulation of enterprise-grade 5G networks.
- **Dynamic Policy and Real-Time Control:** The ROC demonstrated a robust capability to manage policy changes, subscriber configurations, and traffic steering

[1] See `https://bit.ly/upfSliceChangeDemo` for a runtime policy change demonstration.

in real time. Changes propagated within 2–3 seconds and did not disrupt ongoing sessions, confirming that dynamic management can be effectively implemented in a live network environment.

- **Resource Utilization and Stability:** The multi-VM setup maintained stable CPU and memory usage (with CPU utilization below 60% and predictable memory consumption) during functional testing. This configuration provides a scalable and flexible platform, though it remains essential to perform further load testing to verify performance under heavy concurrent usage.

5.5.3 Overall Summary of Results

In summary, the lessons learned from both deployments provide valuable insights into the strengths and limitations of the current Aether platform implementation, guiding future enhancements to achieve greater scalability and performance.The single VM *quick start* environment validated core 5G operations under load, while the lab-based *full Aether* setup demonstrated advanced functionalities at a smaller scale. Both deployments provide complementary insights: performance constraints in a minimal environment, and feature-rich demonstration in a more powerful environment.

Chapter 6 will consolidate these findings, discuss limitations, and propose future enhancements or research directions for private 5G in virtualized settings.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

This thesis has demonstrated the feasibility of deploying Aether, an open-source private 5G platform, in a virtualized environment. Two main deployment scenarios were presented:

- **Single-VM (Quick Start) Deployment:** Provided a streamlined, resource-efficient setup suitable for initial testing and validation of core 5G functionalities.
- **Full Aether Deployment (Lab PC):** Showcased an expanded architec- ture capable of supporting advanced 5G features, dynamic policy enforcement, and traffic steering through multiple UPFs.

Through systematic testing with both `gNBsim` and `UERANSIM`, this work validated critical procedures such as UE registration, PDU session establishment, and QoS management. In addition, it highlighted how runtime configuration tools (e.g., Aether ROC) can enable on-the-fly adjustments to network parameters, facilitating more flexible and scalable private 5G deployments.

The approaches and results in this thesis contribute to both the academic and industrial communities by demonstrating a clear roadmap for deploying and evaluating private 5G solutions in virtualized testbeds. Moreover, the documented challenges and proposed enhancements—particularly regarding UPF routing and monitoring—were contributed back to the Aether project, reinforcing the collaborative nature of open-source infrastructure development.

6.2 Limitations

Trade-Offs Between Simplicity and Feature-Rich Architectures While the single-VM deployment offers a straightforward, resource-friendly approach for early testing and basic functional validation, it is inherently constrained in terms of scalability and performance under heavier loads. In contrast, the Full Aether Deployment demonstrates support for more advanced 5G features, including dynamic policy enforcement and multi-UPF traffic steering, but entails higher complexity in setup and maintenance. Thus, choosing between these approaches depends heavily on the specific use case: rapid prototyping and proof-of-concept deployments favor simplicity, whereas production-level or high-concurrency environments demand more complex and resource-intensive architectures.

6.3 Future Work

Overall Reflections and Recommendations

- **Scaling the SD-Core:** For production-level or high-concurrency scenarios, a distributed architecture spanning multiple physical hosts may be required. Future experiments could investigate horizontal scaling of SD-Core components (e.g., AMF, SMF, UPF) and assess its impact on throughput, latency, and failover mechanisms.
- **Extended Performance Analysis:** Additional work can focus on deeper performance metrics, such as jitter, packet loss, and end-to-end delay under higher UE counts (e.g., 500 or 1,000). Incorporating metrics at multiple points (UE, gNodeB, UPF, external gateway) would yield a more comprehensive performance profile.
- **Advanced 5G Features:** Investigations into network slicing, edge computing integration, or closed-loop orchestration (e.g., using AI/ML-based traffic prediction) could significantly enhance the realism and utility of future testbeds. These features would align the virtualized Aether deployment with emerging enterprise 5G use cases.
- **Security and Isolation Testing:** Another avenue is evaluating the security implications of multi-tenant environments. This includes testing user isolation across slices, robustness against DDoS attacks, and verifying data confidentiality through techniques like IPsec offloading or mutual TLS.

- **Production-Ready Integrations:** Collaborations with industrial partners, testing real small cells (instead of simulated ones), and integrating enterprise applications (like IoT data collection or AR/VR services) could validate the platform's readiness for real-world deployment.

Collectively, these directions offer numerous pathways to refine and expand the findings of this thesis. By advancing the experimentation scope, scaling core components, and integrating more complex features, future work can further confirm Aether's suitability for diverse enterprise 5G scenarios.

References

[1] O. N. Foundation, "Sd-core: A cloud-native 4g/5g core for the enterprise edge," https://aetherproject.org/wp-content/uploads/sites/11/2024/05/SD-Core-Technical-White-Paper-FINAL-1.pdf, ONF, Tech. Rep., 2024, accessed March 2025.

[2] ——, "Sd-ran and ransim: Realistic ran testing in aether," https://aetherproject.org/sd-ran/, 2023, accessed March 2025.

[3] ——, "Runtime operational control and monitoring stack," https://docs.aetherproject.org/aether-2.0/amp/roc.html, 2023, accessed March 2025.

[4] ——, "Aether: Open source platform for enterprise 5g," https://opennetworking.org/aether, 2023, accessed March 2025.

[5] L. Zhang, H. Wang, and J. Chen, "Private 5g networks: A comprehensive survey on architecture, applications, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 1–30, 2023.

[6] S. Kim and R. Patel, "Architecture and key technologies for 5g private networks," *IEEE Network*, vol. 36, no. 3, pp. 45–51, 2022.

[7] J. Doe and R. Kumar, "Enterprise adoption of private 5g: Opportunities and considerations," *Journal of Enterprise Networking*, vol. 15, no. 2, pp. 78–90, 2021.

[8] G. Intelligence, "Private 5g: Industry use cases and applications," 2023, industry Report. [Online]. Available: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/draft_whitepaper_-_leading_the_technology_inventions_for_a_unified_more_capable_air_interface_v.1.0.pdf

[9] M. Alvarez and P. Singh, "Challenges and limitations in deploying private 5g networks," *IEEE Access*, vol. 10, pp. 112 233–112 245, 2022.

[10] O. N. Foundation, "Aether sd-core and gnbsim overview," https://aetherproject.org/sd-core/, 2023, accessed March 2025.

[11] ——, "Aether onramp deployment blueprints," https://docs.aetherproject.org/master/index.html, 2023, accessed March 2025.

[12] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization and software defined networking for 5g," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 656–685, 2015.

[13] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "Container-based virtual network functions: Performance analysis and orchestration in 5g," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 70–77, 2017.

[14] ETSI NFV ISG, "Network Functions Virtualisation (NFV); Architectural Framework," 2013, eTSI GS NFV 002 V1.2.1 (2014-12). [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[15] J. Gil Herrera and J. F. Botero, "A survey on testbeds for network function virtualization (nfv) and software defined networking (sdn)," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3274–3306, 2018.

[16] B. Martinez, C. Bernardos, and A. de la Oliva, "A survey on 5g testbeds and experimental research platforms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 1558–1581, 2019.

Appendix A

Configurations

A.1 Quick Start Configuration (Personal Laptop)

This section presents the YAML configuration used for the Quick Start deployment, including Kubernetes (RKE2) setup, Helm parameters, and SD-Core settings. For a detailed discussion of how these settings were applied in practice, see Chapter 4.

```
k8s:
  rke2:
    version: v1.24.17+rke2r1
    config:
      token: purdue-k8s-rke2
      port: 9345
      params_file:
        master: "deps/k8s/roles/rke2/templates/master-config.yaml"
        worker: "deps/k8s/roles/rke2/templates/worker-config.yaml"

  helm:
    version: v3.17.0

core:
  standalone: true
  data_iface: enp0s3
  values_file: "deps/5gc/roles/core/templates/sdcore-5g-values.yaml"
  ran_subnet: "172.20.0.0/16"
  helm:
    local_charts: false
    chart_ref: aether/sd-core
```

```
    chart_version: 2.2.2
  upf:
    access_subnet: "192.168.252.1/24"
    core_subnet: "192.168.250.1/24"
    mode: af_packet
    multihop_gnb: false
    default_upf:
      ip:
        access: "192.168.252.3"
        core: "192.168.250.3"
      ue_ip_pool: "172.250.0.0/16"

  amf:
    ip: "10.0.2.15"

gnbsim:
  docker:
    container:
      image: omecproject/5gc-gnbsim:rel-1.6.1
      prefix: gnbsim
      count: 1
    network:
      macvlan:
        name: gnbnet

  router:
    data_iface: enp0s3
    macvlan:
      subnet_prefix: "172.20"

  servers:
    0:
      - "deps/gnbsim/config/gnbsim-default.yaml"
```

Listing A.1: Quick Start deployment configuration

A.2 Full Aether Deployment (Lab PC) Configuration

The following YAML file details the configuration used for deploying Aether on the Lab PC environment. Key differences from the Quick Start scenario include a multi-VM approach, multiple UPFs, and integration with the Aether Runtime Operation Control (ROC):

```yaml
k8s:
  rke2:
    version: v1.24.17+rke2r1
    config:
      token: purdue-k8s-rke2
      port: 9345
      params_file:
        master: "deps/k8s/roles/rke2/templates/master-config.yaml"
        worker: "deps/k8s/roles/rke2/templates/worker-config.yaml"

  helm:
    version: v3.17.0

core:
  standalone: false # set to false to place under control of ROC
  data_iface: enp0s9
  values_file: "deps/5gc/roles/core/templates/sdcore-5g-values.yaml"
  ran_subnet: ""
  helm:
    local_charts: false
    chart_ref: aether/sd-core
    chart_version: 2.2.2
  upf:
    access_subnet: "192.168.252.1/24"
    core_subnet: "192.168.250.1/24"
    mode: af_packet
    multihop_gnb: false
    helm:
      local_charts: false
      chart_ref: aether/bess-upf
      chart_version: 1.2.1
    values_file: "deps/5gc/roles/upf/templates/upf-5g-values.yaml"
```

```
    default_upf:
      ip:
        access: "192.168.252.3"
        core: "192.168.250.3"
      ue_ip_pool: "172.250.0.0/16"
    additional_upfs:
      "1":
        ip:
          access: "192.168.252.6"
          core: "192.168.250.6"
        ue_ip_pool: "172.248.0.0/16"
      # "2":
      # ip:
      # access: "192.168.252.7"
      # core: "192.168.250.7"
      # ue_ip_pool: "172.247.0.0/16"

  amf:
    ip: "192.168.56.10"

ueransim:
  gnb:
    ip: "192.168.56.20"
  servers:
    0:
      gnb: "deps/ueransim/config/custom-gnb.yaml"
      ue: "deps/ueransim/config/custom-ue.yaml"

amp:
  roc_models: "deps/amp/roles/roc-load/templates/roc-5g-models-upf2.json"
  monitor_dashboard: "deps/amp/roles/monitor-load/templates/5g-monitoring
      "

  aether_roc:
    helm:
      local_charts: false
      chart_ref: aether/aether-roc-umbrella
      chart_version: 2.1.36
```

```
atomix:
  helm:
    chart_ref: atomix/atomix
    chart_version: 1.1.2

onosproject:
  helm:
    chart_ref: onosproject/onos-operator
    chart_version: 0.5.6

store:
  lpp:
    version: v0.0.24

monitor:
  helm:
    chart_ref: rancher/rancher-monitoring
    chart_version: 101.0.0+up19.0.3

monitor-crd:
  helm:
    chart_ref: rancher/rancher-monitoring-crd
    chart_version: 101.0.0+up19.0.3
```

Listing A.2: Lab PC deployment configuration.

Just like in Listing A.1, this configuration file outlines the parameters for Kubernetes, Helm, SD-Core, and multiple UPFs. However, it sets `standalone: false` to allow control by the Aether ROC. For more details on the differences in this deployment's hardware and topology, refer to Chapter 4.

Appendix B

Flows and Logs Analysis

B.1 Complete 5G Call Flows and Logs

This section provides access to the 5G call-flow diagrams and raw logs generated during the multi-UPF deployment tests. Each call-flow PDF corresponds to a specific `IMSI` and includes detailed message exchanges (registration, PDU session establishment, release procedures, etc.). Additionally, an optional raw log file (`e2eCallflows.log`) contains a plain-text record of all signaling steps.

Table B.1: Call Flow Files with IMSI References

IMSI	Call Flow PDF Link
208930100007500	`View PDF`
208930100007560	`View PDF`
208930100007620	`View PDF`
208930100007680	`View PDF`
208930100007750	`View PDF`
208930100007880	`View PDF`

Raw Log File (Optional). The plain-text log file listed below contains complete end-to-end call flows including NAS messages, authentication steps, security mode commands, and success indicators for each procedure:

- `e2eCallflows.log`

These files capture the step-by-step message exchanges observed in the multi-UPF deployment. For details on how these logs were parsed and analyzed programmatically, see Section B.2.

B.2 Analysis Scripts and Code Repositories

This section provides links to the Python/Jupyter notebooks used for automated logs analysis and call-flow interpretation. The notebooks are publicly accessible on GitHub for transparency and reproducibility.

B.2.1 Logs Analysis

The first notebook parses and analyzes 5G logs from both the Quick Start and Full Aether deployments, extracting key metrics (e.g., procedure latencies, success/failure counts) to produce summary statistics:

- **GitHub Link:** `LogsAnalysis.ipynb (GitHub)`

Readers may either view it directly on GitHub or clone the repository to open it locally in Jupyter or a compatible environment.

B.2.2 Call Flow Analysis

A second notebook specifically focuses on visualizing call-flow sequences, including registration, PDU session establishment, service requests, and deregistrations. It reads the detailed logs to highlight message sequences and timings:

- **GitHub Link:** `CallFlowsAnalysis.ipynb (GitHub)`

This notebook also generates message-sequence charts to illustrate how different UEs progress through various 5G procedures.

How to Use the Notebooks

1. **Clone the Repository:** `git clone https://github.com/hxngillani/Thesis-2025.git`
2. **Install Dependencies:** Ensure you have Python 3 and Jupyter installed. Then install any additional libraries (e.g., `pandas`, `numpy`, `matplotlib`) using:

```
pip install -r requirements.txt
```

3. **Run Jupyter:** In the project directory, run `jupyter notebook` (or `jupyter lab`) and open either `LogsAnalysis.ipynb` or `CallFlowsAnalysis.ipynb`.

These notebooks provide a reproducible way to examine the data collected from the Aether deployments, confirm key metrics, and visualize the message flows. This approach supports transparent testing, validation, and potential extensions for future research.

Appendix C

Appendix: Network Architecture Diagrams

This section includes the high-resolution network architecture and deployment diagrams used throughout the thesis. These diagrams were manually created using **draw.io** to illustrate the setup of the Aether platform, including Kubernetes networking, VirtualBox VM topology, UPF deployment models, and routing configurations.

All diagrams are available in both .SVG and editable `.drawio` formats. Readers can download and reuse these resources for replication or adaptation purposes.

Available Diagrams

- **Aether Deployment Diagrams** – .SVG | .drawio

Repository Access

All files are hosted on GitHub and can be accessed or downloaded via the following folder:

- `github.com/hxngillani/Thesis-2025`

These resources were developed to support the experimental design and documentation process, and are shared under an open-access model to facilitate reuse and collaboration.