D5.5: Final report on AI-data driven networking QoS management, and onboard networking capabilities

Revision: v.1.1

Work package	WP 5
Task	Task 5.2, 5.4, 5.5
Due date	31/03/2026
Submission date	30/03/2026
Deliverable lead	Marius Corici (FHG)
Version	1.1F
Authors	Daniele Tarchi, David Naseh, Swapnil Sadashiv Shinde (CNIT), Benjamin Barth, Roshith Sebastian, Gayathri Guruvayoorappan (DLR), Amel Tibhirt (Orange), Marius Corici, Hauke Buhr, Hemant Zope (FHG)
Reviewers	Oriol Font Bach (SRS)
Abstract	This deliverable described the developments for AI driven NTN network, the QoS Management and the onboarding network capabilities
Keywords	AI/ML, 5G NTN, Network Function Split, Network Function Placement

Co-funded by the European Union 6GSNS
Grant Agreement No.: 101096573
Call: HORIZON-JU-SNS-2022
Topic: HORIZON-JU-SNS-2022-STREAM-A-01-02
Type of action: HORIZON-JU-RIA

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	29/01/2024	1st edit, draft ToC	Marius Corici (FHG)
V0.2	12/09/2025	Major revision of the ToC	Daniele Tarchi (CNIT)
V1.0	07/10/2025	First Version	All partners
V1.01	20/03/2026	Draft version fo QA review	Daniele Tarchi (CNIT)
V1.02	24/03/2026	QA review	Oriol Font Bach (SRS)
V1.03	30/03/2026	Revised version according to the QA comments	Marius Corici (FHG), Daniele Tarchi (CNIT)
V1.1F	31/03/2026	Approved version for EC portal upload	Tomaso de Cola (DLR)

DISCLAIMER

Co-funded by the European Union 6GSNS

5G-STARDUST (*Satellite and Terrestrial Access for Distributed, Ubiquitous, and Smart Telecommunications*) project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101096573.

This work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**to specify R, DEM, DEC, DATA, DMP, ETHICS, SECURITY, OTHER***	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	

Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

This deliverable, “Final report on artificial intelligence (AI)-data driven networking, quality of service (QoS) management, and onboard networking capabilities”, presents the concluding results of the 5G-STARDUST project’s activities in Work Package 5. The focus is set on enabling efficient and flexible integration of terrestrial and non-terrestrial systems through AI-driven network management, advanced QoS mechanisms, and novel spaceborne networking capabilities.

The first part of the report advances the concept of AI data-driven networking by embedding artificial intelligence into the control and optimization of heterogeneous 5G and beyond-5G infrastructures. Building on the open radio access network (O-RAN) architecture, the work demonstrates how machine learning (ML) and reinforcement learning (RL) can dynamically optimize function placement, bearer selection, and beam hopping in multi-tier terrestrial network (TN)–non terrestrial network (NTN) deployments. By coupling AI monitoring frameworks with flexible orchestration, the approach reduces signaling overhead while maintaining accurate data for adaptive decision-making.

The second part introduces an end-to-end QoS management framework tailored to mega-constellations. This includes a cross-layer design that interconnects the 5G service layer with the constellation-level routing layer, enabling continuous QoS assurance despite satellite mobility and variable link conditions. The solution leverages conditional handovers, predictive routing, and adaptive resource allocation to guarantee robust QoS, even under high-latency and dynamic topologies typical of NTNs.

The third part of the deliverable addresses onboard networking capabilities. Different architectural options for deploying core network functions in orbit are analyzed, ranging from minimal regenerative payloads to full spaceborne cores. The study shows that co-locating tightly coupled functions into “micro-cores” distributed across multiple satellites reduces inter-satellite signaling and improves robustness. Further, the concept of user plane function (UPF) shortcuts is introduced, including a functional split of the UPF to balance contained latency with reduced feeder-link dependency. These mechanisms provide resilient and efficient user-plane operations while minimizing the complexity of satellite payloads.

Overall, the deliverable underlines three main advancements: (1) the adoption of AI techniques for real-time orchestration of TN–NTN resources, (2) the design of robust QoS assurance tailored to satellite mega-constellations, and (3) the introduction of practical models for onboard core network, UPF optimization, multi-access edge computing (MEC) and TN-NTN interoperability.

Together, through the algorithms, implementations, and validations towards these directions, the feasibility of integrating satellite and terrestrial 5G, and future 6G, are creating a credibility basis into new technologies. The deliverable shows how latency, mobility, and resource constraints can be effectively addressed, while also identifying a set of promising technologies that are ready to be further matured beyond the project in research, standardization, and industrial adoption.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AF	Application Function
AI	Artificial Intelligence
AnLF	Analytics Logical Function
ATSS	Access Traffic Steering, Switching, and Splitting
BDQ	Branch Dueling Q-network
CN	Core Network
CPU	Central Processing Unit
DBN	Dynamic Bayesian Network
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
E2E	End-to-End
eMBB	enhanced Mobile Broadband
IP	Internet Protocol
KAN	Kolmogorov-Arnold Networks
KPIs	Key Performance Indicators
LP	Linear Programming
LSTM	Long Short-Term Memory
MEC	Multi-Access Edge Computing
MILP	Mixed-Integer Linear Programming
ML	Machine Learning
mMTC	massive Machine Type Communications
MTLF	Model Training logical function
NF	Network Function
NTN	Non-Terrestrial Network
NWDAF	Network Data Analytics Function
O-RAN	Open Radio Access Network
PDU	Protocol Data Unit

QoA	Quality of Assurance
QoS	Quality of Service
RAN	Radio Access Network
RIC	RAN Intelligent Controller
RL	Reinforcement Learning
SFC	Service Function Chaining
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TN	Terrestrial Network
UE	User Equipment
UPF	User Plane Function
URLLC	Ultra Reliable Low Latency Communication
VLEO	Very Low Earth orbit
VNF	Virtual Network Function

1 INTRODUCTION

The deliverable titled *"Final report on AI-data driven networking, quality of service (QoS) management, and onboard networking capabilities"* is centered on the progress and final innovations achieved in Work Package 5 (WP5) of the 5G-STARDUST project. WP5 focuses on integrating artificial intelligence (AI)-driven networking solutions, comprehensive QoS management, and the analysis of on-board networking capabilities in order to enhance the performance and reliability of 5G and beyond-5G networks in the context of Non-Terrestrial Networks (NTNs).

This deliverable builds on the architectural developments from D3.2 [42] and extends the preliminary activities reported in D5.3. It specifically addresses the advancements in three major tasks that are pivotal in achieving the project's goals:

- **Task 5.2: AI Data Driven Network Management and Slicing.** This task has developed optimizations for network management using AI insights. The predictability and flexibility of NTNs, when combined with static Terrestrial Networks (TNs), allow superior service delivery through optimized network configurations. The framework reported here enables multiple AI decisions to run in parallel, balancing data acquisition, decision-making, and dissemination under strict resource constraints. The work demonstrates how machine learning (ML) and reinforcement learning (RL) algorithms can optimize orchestration, network slicing, and function placement across TN–NTN infrastructures.

- **Task 5.4: End-to-End QoS Management.** This task has proposed mechanisms to extend 5G QoS profiles to accommodate the unique properties of NTNs and enforce these profiles consistently across both terrestrial and non-terrestrial domains. Particular emphasis has been placed on protecting feeder links, rerouting at constellation level, and introducing cross-layer coordination between the 5G service layer and the routing layer. These mechanisms have been validated to sustain robust QoS despite the variability of non-terrestrial topologies and link conditions.

- **Task 5.5: On-Board Networking Capabilities.** This task has introduced an analysis of deploying 5G core network functions and Multi-Access Edge Computing (MEC) capabilities on-board satellite payloads. Several placement models were studied, ranging from regenerative payloads with minimal on-board functionality to fully integrated spaceborne cores. The work highlights the advantages of co-locating tightly coupled functions into distributed micro-cores, the need for active state synchronization between satellites, and the role of user plane function (UPF) shortcuts in reducing feeder-link dependency and containing latency. These investigations provide a foundation for understanding how NTN can evolve towards more autonomous and resilient architectures in 6G.

This deliverable provides the final presentation of the activities carried out in WP5. It demonstrates the feasibility of key features for AI-driven orchestration, end-to-end QoS assurance, and on-board core and edge deployments, while underlining a set of promising technologies that can be matured beyond the project in the evolution of NTN–TN integration.

2 AI DATA-DRIVEN NETWORKING

2.1 MOTIVATIONS

2.1.1 Overview of O-RAN and VNFs

The Open Radio Access Network (O-RAN) introduces an open and disaggregated approach to the design and deployment of Radio Access Networks (RAN). It defines standards and specifications that enable interoperability across vendors and promotes an architecture that is more intelligent, virtualized, and modular. By separating hardware and software components and relying on open interfaces, O-RAN reduces vendor lock-in and supports the integration of best-in-class subcomponents from multiple suppliers, fostering competition and innovation [1].

The emergence of O-RAN is a relatively recent phenomenon, with the O-RAN Alliance, a key driving force behind this movement, being founded in 2018 by several leading mobile operators. This development has been largely propelled by the escalating demands placed on modern mobile networks, particularly with the advent of 5G and the proliferation of Internet of Things (IoT) devices. These demands necessitate a level of flexibility, scalability, and cost-efficiency that traditional RAN architecture often struggles to provide. O-RAN builds upon the established principles of virtualization (vRAN) and cloudification, extending these concepts with a significant emphasis on the crucial aspect of open interfaces. This evolution signifies a concerted effort within the telecommunications industry to move towards more adaptable and efficient network solutions.

The adoption of O-RAN architectures in satellite and TNs offers a convincing pathway to address the challenges and unlock the full potential of these next-generation networks. NTNs face difficulties, such as high latencies, the complexity of integrating diverse network elements, and the need for flexible resource management across vast and often remote areas. O-RAN's foundational principles of openness, disaggregation, and intelligence provide inherent advantages. Its modular design and open interfaces facilitate multi-vendor interoperability, reducing vendor lock-in and fostering a more competitive and innovative ecosystem. This flexibility is crucial for NTNs, allowing for the integration of specialized components from different suppliers, optimizing for the harsh and resource-constrained environments of space or high altitudes. Furthermore, O-RAN's support for AI-driven RAN Intelligent Controllers (RICs) enables dynamic optimization of network resources and intelligent management of complex network topologies, which is vital for adapting to the dynamic nature of NTN links, including satellite movements and varying atmospheric conditions. Ultimately, O-RAN can enhance the cost-efficiency, adaptability, and scalability of Beyond 5G NTNs, accelerating the delivery of ubiquitous and resilient connectivity.

Virtual Network Functions (VNFs) represent a fundamental shift in how network services are designed, deployed, and managed. In essence, VNFs are software-based applications that perform specific network functions which were traditionally carried out by dedicated, proprietary hardware appliances [12]. These functions can include a wide array of services such as routing, firewalls, load balancing, Wide Area Network (WAN) optimization, and Network Address Translation (NAT).

The core role of VNFs lies in decoupling network functions from physical hardware. Instead of requiring a separate physical box for each network service, VNFs allow these services to run as software applications on standard IT infrastructure, typically on virtual machines (VMs) or, more recently, containers. This is a key principle of Network Functions Virtualization (NFV), an

architectural framework that leverages IT virtualization technologies to virtualize entire classes of network node functions.

Within the NFV architecture, VNFs are the actual software components that deliver network services. They operate on top of an NFV Infrastructure (NFVI), which includes the necessary physical compute, storage, and networking resources, as well as a virtualization layer (e.g., hypervisors like KVM[1] or VMware[2]). A management and orchestration (MANO) framework is responsible for managing the NFVI and orchestrating the lifecycle of VNFs, including their deployment, configuration, scaling, and termination.

VNFs can be "chained" together in a specific order to create more complex network services, a process known as service chaining. This allows for a modular and flexible approach to building and delivering end-to-end network services.

O-RAN principles encourage open interfaces and multi-vendor environments. VNFs align with this by promoting the use of software from various vendors on common hardware platforms. This reduces vendor lock-in and allows organizations to choose competitive solutions for different network functions.

The benefits of VNFs become even more noticeable in Integrated Satellite and Terrestrial Networks. These converged networks aim to provide ubiquitous, resilient, and high-performance connectivity by seamlessly blending the wide coverage of satellites with the high capacity of terrestrial systems.

2.1.2 Monitoring for AI-Driven Control in Integrated TN-NTN

An effective AI-driven network relies on a comprehensive and efficient monitoring system capable of collecting the necessary data to train and execute ML models. In the hybrid TN-NTN architecture of 5G-STARDUST, this presents a unique challenge: balancing the need for granular, real-time data with the constraints of the satellite backhaul, which represents a limited and valuable resource.

The monitoring framework for 5G-STARDUST has therefore addressed a critical trade-off:

- **Data Granularity vs. Resource Overhead:** Collecting highly detailed performance indicators (e.g., per-user equipment, UE, signal-to-interference-plus-noise ratio ,SINR, individual physical resource block, PRB, usage) from RAN nodes provides the most accurate input for AI algorithms, especially for real-time control loops in the Near-Real-Time (RT) RIC. However, transmitting this large volume of telemetry data, particularly from an onboard gNB over a feeder link, consumes significant bandwidth and power.

- **Centralized vs. Distributed Data Collection:** A centralized approach, where all data is sent to a ground-based Non-RT RIC for analysis, provides a global network view but incurs high transport costs and latency. A distributed approach, where data is pre-processed onboard the satellite or at the gNB, reduces the data volume but requires additional computational resources on the satellite payload.

[1] Kernel-based Virtual Machine (KVM) https://linux-kvm.org

[2] VMWare Virtual machines https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion

To address this, the 5G-STARDUST architecture has leveraged the O-RAN framework to create an adaptive monitoring system:

1. **Dynamic Data Collection:** Non-RT RIC is orchestrating the monitoring process, defining the data collection policies based on current network objectives. For routine operations, a less granular set of key performance indicators (KPIs) might be collected via the O1 interface to conserve resources. In its default state, the Service Management and Orchestration (SMO) collects aggregated cell-level metrics every 60 seconds, including average throughput, packet drop rate, and the number of active UEs, providing a low-overhead view of network health

2. **Event-Triggered Reporting:** For specific optimization tasks, such as handover management or interference mitigation, the Near-RT RIC can request more detailed, real-time data streams from specific E2 nodes for a limited duration. An xApp within the Near-RT RIC is configured with a policy from the Non-RT RIC : IF the average packet delay for a QoS Flow Identifier (QFI) associated with a URLLC slice exceeds 10ms, THEN the Near-RT RIC will issue an E2 Subscription request to the relevant open distributed unit (O-DU)/open central unit (O-CU) for per-UE SINR and buffer status reports every 100ms for the next 5 minutes.

3. **Onboard Pre-processing:** For regenerative payload configurations, some data aggregation and feature extraction can be performed onboard the satellite. This reduces the data volume that needs to be sent over the feeder link, thus optimizing the use of the satellite's backhaul resources while still providing the necessary insights for ground-based RICs. To conserve feeder link resources, the onboard gNB-DU will perform data pre-processing. Instead of transmitting raw per-slot PRB usage data, it will calculate and transmit only the 30-second moving average and standard deviation, reducing telemetry volume by over 95%.

This adaptive approach ensures that the AI engines receive the data they need to perform their functions without overburdening the critical NTN communication links, thereby creating a sustainable and efficient monitoring backbone for the self-organizing network.

2.1.3 Objectives and Scope

This section of the deliverable has the following primary objectives:

- **Provide an overview of the O-RAN architecture** as the reference framework adopted in this deliverable, introducing the main building blocks (e.g., SMO, Non-RT RIC, Near-RT RIC, O-CU, O-DU, O-RU), the relevant interfaces, and the rationale for disaggregation and multi-vendor interoperability in TN–NTN scenarios.

- **Introduce the role of virtualization (VNFs/NFV) in O-RAN-enabled deployments**, clarifying how virtualized network functions support flexible RAN and core/edge deployments and contribute to reducing vendor lock-in in heterogeneous environments.

- **Describe AI data-driven networking functions for integrated TN–NTN management**, with emphasis on the monitoring-to-insight pipeline and on AI-assisted orchestration/control mechanisms that enable network slicing and policy-based optimization across terrestrial and non-terrestrial domains.

- **Present and assess AI/ML-based solutions for dynamic optimization**, covering the modeled use cases discussed in the section (e.g., beam hopping, bearer selection/traffic

steering, and function placement/load balancing), and reporting simulation-based evidence in terms of relevant KPIs (e.g., latency, energy, and resource utilization).

The scope of this study is defined by the following boundaries:

- **Architectural and functional focus:** The discussion concentrates on the architectural principles and functional roles of the O-RAN components used in the section (including Non-RT and Near-RT control functions), and on their interaction with virtualized deployments. The emphasis is set on the **control/management planes** rather than on hardware design.

- **Application to NTN integration:** The scope is tailored to the challenges and opportunities introduced by NTN (e.g., satellite dynamics, multi-tier deployment, and satellite-specific features such as beam hopping), and how these affect data-driven optimization in the integrated TN–NTN architecture.

- **AI-driven network management:** The section focuses on AI/ML methods applied to **resource and service optimization** (notably slicing-related policies and traffic steering/bearer selection, beam hopping decisions, and function placement/load balancing), consistent with the AI data-driven networking goals of the task.

- **Modeling and simulation-based evaluation:** The analysis relies on problem formulations and ML/AI models already presented in the text (including learning-based approaches such as Deep Q-Network (DQN) [44] where applicable) and evaluates performance through simulation results using KPIs such as latency, energy consumption, and resource utilization.

- **Exclusions:** The scope does not include detailed physical implementation aspects (hardware-level specifications, radio frequency, RF, design, or low-level component engineering). The focus remains on system-level control logic, orchestration, and algorithmic optimization.

2.2 O-RAN ARCHITECTURE AND COMPONENTS

2.2.1 O-RAN Architecture Overview

The O-RAN Alliance is heading a transformative shift in RAN design, moving away from traditional, monolithic structures towards an open, intelligent, virtualized, and fully interoperable RAN. The O-RAN architecture is characterized by its modular design, the separation of hardware and software components, and the introduction of open standardized interfaces that enable multi-vendor solutions. This approach aims to foster innovation, increase flexibility, and reduce deployment costs.

At its core, the O-RAN architecture disaggregates the traditional base station into distinct, interoperable components. This modularity is a key enabler for increased vendor diversity and flexibility. The architecture promotes the decoupling of RAN software from the underlying proprietary hardware, allowing software to run on general-purpose hardware platforms (often referred to as "white-box" hardware or Commercial Off-The-Shelf, COTS, hardware) within a cloud-based environment known as the O-Cloud.

This separation offers several advantages:

- **Vendor Diversity:** Network operators are no longer locked into a single vendor's ecosystem for their entire RAN deployment. They can select competitive components from different suppliers.
- **Cost Reduction:** The use of COTS hardware can lead to significant cost savings compared to specialized, proprietary hardware.
- **Faster Innovation:** Open interfaces and a software-centric approach allow for quicker development and deployment of new features and services by a broader ecosystem of software developers.
- **Increased Flexibility and Scalability:** Network functions can be scaled independently and deployed where they are most needed, optimizing resource utilization.

The O-RAN architecture defines open and standardized interfaces between these disaggregated components, ensuring interoperability. Key interfaces include the fronthaul interface (connecting the O-RU to the O-DU), the F1 interface (connecting the O-DU to the O-CU), and various interfaces for management, orchestration, and AI-driven control.

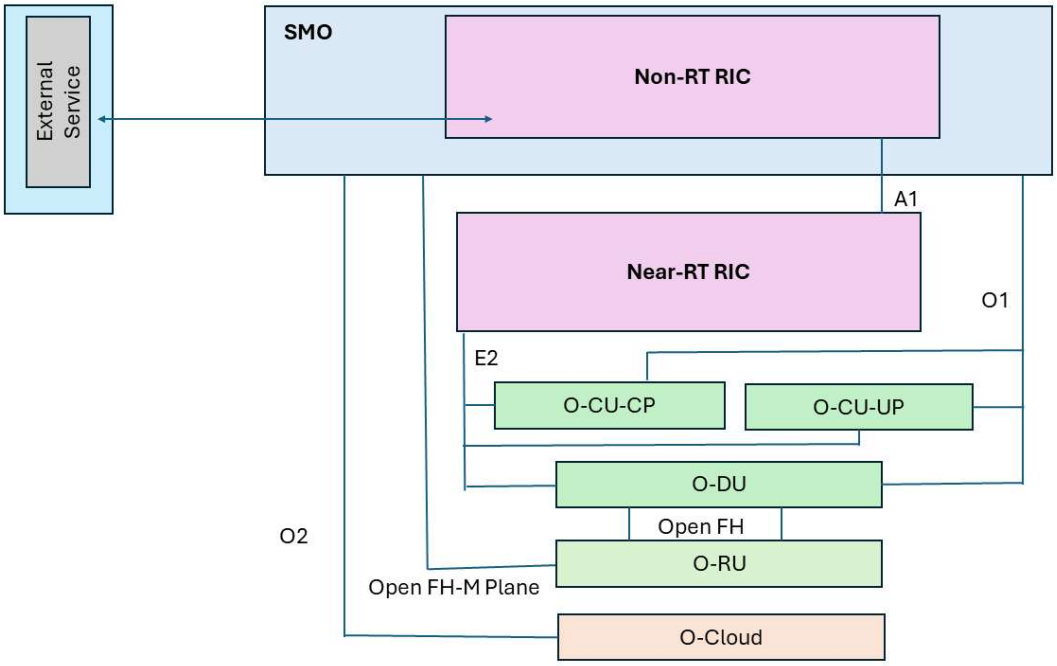


Figure 2-1: O-RAN Architecture

The O-RAN architecture is composed of several key logical components, each with specific functions. The SMO Framework is responsible for the overall management and orchestration of the O-RAN components. It includes functions for fault, configuration, accounting, performance, and security (FCAPS) management. O-RAN RIC is a central piece of the O-RAN architecture, designed to bring intelligence, control, and programmability to the RAN. It is split into two logical forms: (i) the Non-RT RIC that provides higher-level, non-real-time (greater than 1 second) control and policy guidance to the Near-RT RIC and the underlying RAN elements. It enables intelligent RAN optimization by leveraging data analytics and AI/ML capabilities. (ii) The Near-RT RIC provides near-real-time control and optimization of RAN elements and resources, typically with a control loop latency between 10 milliseconds and 1 second. The O-Cloud provides the cloud computing infrastructure (compute, storage, networking resources) and the virtualization/containerization platform necessary to host O-

RAN functions, such as the O-CU, O-DU, Near-RT RIC, and other VNFs/CNFs. The O-CU is a logical node that hosts the higher-layer protocols of the radio stack. In a 5G context, this includes Radio Resource Control (RRC), Service Data Adaptation Protocol (SDAP), and Packet Data Convergence Protocol (PDCP). The O-DU is a logical node that hosts the lower layers of the radio protocol stack. This typically includes Radio Link Control (RLC), Medium Access Control (MAC), and parts of the Physical (PHY) layer (High-PHY). The O-RU is responsible for the RF processing and the lower part of the Physical (PHY) layer (Low-PHY).

These components, interconnected by open and standardized interfaces, form the foundation of the O-RAN architecture, enabling a more flexible, intelligent, and cost-effective approach to building and evolving Radio Access Networks.

2.2.2 Components of O-RAN: RU, DU, CU, Near-RT RIC, Non-RT RIC, SMO

O-RAN architecture achieves its openness and flexibility through the disaggregation of the traditional base station into distinct, interoperable components. Each component has a well-defined role and communicates with others through standardized interfaces. The primary components are the RU, DU, CU, Near-RT RIC, Non-RT RIC, and the SMO framework [2].

- O-RAN Radio Unit (O-RU):
  - The O-RU is a physical or virtualized entity that handles RF transmission and reception. It includes the hardware and software responsible for the digital-to-analog and analog-to-digital conversion, power amplification, and the antenna interface. It also implements the lower parts of the PHY layer processing.
  - The O-RU's primary role is to manage the air interface, converting radio signals received from user equipment into digital data for the O-DU, and converting digital data from the O-DU into radio signals for transmission. It connects to the O-DU via the open fronthaul interface (e.g., based on eCPRI).
- O-RAN Distributed Unit (O-DU):
  - The O-DU is a logical node that typically runs on COTS hardware. It hosts a subset of the baseband processing functions, specifically the real-time L2 functions such as the RLC and MAC layers, as well as parts of the PHY layer (i.e., High-PHY).
  - The O-DU is responsible for real-time baseband processing that is sensitive to latency. It schedules data, manages radio resources for multiple users, and ensures reliable data transfer between the O-RU and the O-CU. It interfaces with the O-RU over the open fronthaul and with the O-CU via the F1 interface.
- O-RAN Central Unit (O-CU):
  - The O-CU is a logical node, also typically running on COTS hardware in a centralized location or at an edge data center. It hosts the higher-layer radio protocols, including the RRC, SDAP, and PDCP in a 5G context. The O-CU can be further split into a Control Plane (O-CU-CP) and a User Plane (O-CU-UP) for independent scaling and deployment.
  - The O-CU manages multiple O-DUs and handles non-real-time and less latency-sensitive L3 functions. This includes call admission control, radio resource management at a higher level, and mobility management. It connects to the core network and to the O-DUs via the F1 interface.

- Near-Real-Time RAN Intelligent Controller (Near-RT RIC):
  - Near-RT RIC is a software platform that enables fine-grained control and optimization of RAN elements and resources. It operates with a control loop latency typically between 10 milliseconds and 1 second. It allows third-party applications (xApps) to be deployed to enhance RAN performance.
  - Its primary role is to provide near-real-time intelligence to the RAN. It hosts xApps that perform tasks such as load balancing, interference management, radio resource optimization, and QoS management by collecting data from RAN elements (O-CU/O-DU) via the E2 interface and providing control commands back to them.
- Non-Real-Time RAN Intelligent Controller (Non-RT RIC):
  - The non-RT RIC is part of the SMO framework. It enables greater-than-one-second control and policy optimization of the RAN. It uses AI/ML models and long-term network data analytics to derive policies and enrichment information.
  - The non-RT RIC provides higher-level policy control and guidance to the Near-RT RIC (via the A1 interface) and directly to RAN elements (via O1). It hosts specialized applications called "rApps" that manage RAN policies, perform AI/ML model training for RAN optimization, and enable intelligent orchestration based on service requirements and network conditions.
- Service Management and Orchestration (SMO) Framework:
  - The SMO is a comprehensive framework responsible for the overall management and automation of the O-RAN network. It encompasses the non-RT RIC and other orchestration and management functions.
  - The SMO's role is to provide end-to-end lifecycle management of O-RAN network functions and services. This includes FCAPS management, service orchestration, resource optimization, and automation of network operations. It interacts with various O-RAN components through standardized interfaces like O1 (for management of O-CUs, O-DUs, O-RUs, and Near-RT RIC), A1 (between Non-RT RIC and Near-RT RIC), and O2 (for O-Cloud resource management).

2.2.3 Functional Splits and Interfaces

O-RAN disaggregates the traditional monolithic base station into interoperable functional entities connected through open interfaces, enabling flexible deployments and multi-vendor interoperability [3]. In practice, this is achieved through functional splits that distribute the gNB protocol stack across the O-RU, O-DU, and O-CU; the chosen split directly affects transport requirements between units and influences latency, processing location, and component complexity.

While various functional splits are technically possible across the RAN protocol stack, the O--RAN Alliance has focused on standardizing specific options to promote interoperability. Two prominent splits are:

- **Split 8 (or CPRI split):** This is a "lower layer" split where the O-RU performs minimal processing, primarily handling the digital front-end and converting between analog and digital signals. The vast majority of the physical layer and higher layer processing resides in the O-DU and O-CU. This split places very high bandwidth requirements on the

fronthaul interface between the O-RU and O-DU due to the transport of raw or lightly processed I/Q samples. While simpler for the O-RU, the fronthaul demands can be a limitation, particularly for large-scale deployments. This split is more aligned with traditional CPRI-based fronthaul and is less common in newer O-RAN deployments aiming for greater flexibility.

- **Split 7-2x:** This is a "lower layer" split that has been standardized by the O-RAN Alliance as the Open Fronthaul interface. In this split, some of the lower PHY functions remain in the O-RU (e.g., FFT/iFFT, Cyclic Prefix addition/removal, digital beamforming), while the more complex PHY functions and higher layers are located in the O-DU. This split significantly reduces the fronthaul bandwidth requirements compared to Split 8, making it more economical to deploy and scale. The "x" in 7-2x denotes variations within this split option (Category A and Category B) based on where the precoding function is located. Split 7-2x strikes a balance between centralizing processing for resource pooling gains and keeping some time-critical functions closer to the antenna.

Functional splits provide several advantages:

- **Flexibility:** Different splits allow operators to choose the most suitable architecture based on deployment scenarios, available transport, and desired performance characteristics. This flexibility enables optimization for various use cases, from dense urban environments requiring low latency to rural areas where fronthaul capacity is limited.

- **Scalability:** By separating functions, O-RAN allows independent scaling of the O-RU, O-DU, and O-CU. Operators can add capacity where needed (e.g., deploying more O-RUs for coverage or scaling O-DU/O-CU resources for capacity) without replacing the entire base station. Centralization of the O-DU and O-CU functions also facilitates resource pooling gains across multiple O-RUs.

- **Vendor Diversity:** Standardized functional splits and interfaces enable operators to mix and match components from different vendors, fostering a more competitive ecosystem and avoiding vendor lock-in.

Open interfaces are a core enabler of O-RAN interoperability and closed-loop control. In this deliverable, we mainly refer to the following interfaces:

- **E2:** connects the **Near-RT RIC** to **E2 nodes (O-CU/O-DU)** to support near-real-time telemetry collection and control actions (xApps).

- **A1:** connects the **Non-RT RIC** to the **Near-RT RIC** to deliver policy guidance, enrichment information, and AI/ML model artifacts.

- **O1:** connects **O-RAN managed elements** to the **SMO** for OAM/FCAPS functions, including configuration and performance (KPI) management.

This interface separation supports multi-vendor integration while enabling different control timescales (Non-RT vs Near-RT) in the O-RAN architecture.

2.3 MULTI-TIER O-RAN TN–NTN SYSTEM MODEL AND PROBLEM STATEMENTS

This subsection introduces the reference multi-tier TN–NTN system model adopted in this deliverable and summarizes the corresponding problem statements addressed within the O-

RAN framework. Building on the architectural options and baseline considerations introduced in D5.3, we consolidate the key elements required to (i) define placement and load-balancing decisions for virtualized functions, and (ii) characterize two additional AI-driven control problems in NTN, namely beam hopping and bearer selection, together with their mapping onto O-RAN control loops.

From an architectural perspective, the baseline distinguishes between direct connectivity (UE-to-NTN access without terrestrial intermediaries) and indirect connectivity (UE access supported through a terrestrial relay, such as an integrated access-backhaul, IAB, node). These alternatives affect where functions and control logic can be placed and how resources can be coordinated across terrestrial and non-terrestrial segments. In all cases, the overarching objective is to enable flexible management of virtualized functions and slicing policies under the constraints imposed by LEO dynamics and distributed resource availability.

The remainder of this subsection is structured as follows. We first describe the scenario assumptions and requirements, then formalize the network function placement and load-balancing problem, and finally present the beam hopping and bearer selection problem statements, including their realization through Non-RT and Near-RT RIC functions.

Preliminary Options for Implementation

D5.3 presents several options for network function placement and slicing, with a focus on implementing an intelligent and optimized network using the O-RAN framework. The most promising options for implementation are those that leverage AI and provide flexibility across the distributed network.

- **VNF Placement:** D5.3 highlights the importance of placing VNFs to fine-tune performance based on service demands and available resources. This includes considering infrastructure data, UE demands, and resource availability. The placement of core network functionalities, such as the UPF and Core Control Functionality (CCF), is also discussed, with various options for placement at different locations in the network (e.g., ground station, satellite).

- **RAN Function Deployment:** The controller can deploy RAN functions over heterogeneous infrastructures. Seven different options for RAN function deployment are presented, which are related to the functional splits of the O-RAN architecture, allowing for a flexible deployment over different networking facilities. The selection of the most appropriate functional split is a key decision to be managed by the controller.

- **Network Slicing:** D5.3 details approaches to network slicing, which maps user requirements to specific RAN/core network (CN) configurations. Two main resource allocation options are proposed: dedicated and shared. The dedicated approach ensures higher reliability for mission-critical applications at the cost of resource utilization, while the shared approach improves overall resource utilization.

- **Multi-connectivity:** D5.3 also focuses on using multi-connectivity solutions to increase network resilience. This involves decisions on which of the multiple connections (e.g., TN and NTN) to use, with decisions taken either by the network (through policies, conditional modifications, or commands) or the UE. The framework is designed to support the implementation of AI optimizations for these decisions.

The preliminary nature of D5.3 implies it provides the foundational framework for these elements rather than the specific optimization algorithms. The detailed application and validation of these concepts are explored in this deliverable.

2.3.1 Scenario Description

In a multi-tier satellite network, the system must accommodate a diverse set of services, each with unique requirements regarding latency, throughput, and reliability. This presents a complex scenario where a single physical infrastructure needs to support multiple logical network slices. The network must also efficiently manage and share functions among these various services to avoid redundant resource allocation and optimize overall system performance. The complexity is further compounded by the need for dynamic function placement across different network layers, from the satellite segment to the ground station, to meet the real-time demands of users and services.

The foundational approach to function placement in this multi-tier network is based on network slicing. As detailed in the deliverable, a network slicing approach involves defining multiple slices, each tailored to specific service requirements, and then dynamically mapping these requirements to the available network resources.

The controller plays a central role in this process, responsible for the VNF placement and the deployment of RAN functions. The document outlines a strategy where the controller decides on the optimal placement of various O-RAN functions—including a generic VNF placement and the specific Beam hopping function placement—across the distributed end-to-end network. This is achieved by considering several factors, such as infrastructure data, UE demands, and resource availability. The baseline approach utilizes this slicing model to ensure that resources are provisioned and allocated in a manner that best serves each service's specific demands while efficiently sharing common functions.

The adoption of this network slicing and intelligent function placement approach offers significant benefits while necessitating a clear definition of requirements.

Requirements

The reference model assumes that user/service demands are expressed through slice definitions with explicit QoS targets. The controller then allocates compute, storage, and communication resources across nodes and continuously evaluates whether placement and control decisions meet QoS requirements. Key requirements include:

- User demand to slice mapping and QoS specification.
- Resource allocation across slices (dedicated vs shared).
- Beam hopping function placement and observability for closed-loop control.
- Continuous performance assessment across layers (satellite/ground/cloud).

Benefits and Optimization Parameters

The key benefits of this approach are performance optimization and resource efficiency. By strategically placing VNFs and shared functions, the network can minimize latency, maximize throughput, and reduce operational costs. Based on the performance evaluation and feasibility analysis, the following are defined as possible optimization parameters and objectives:

- **Minimizing End-to-End Latency:** Placing time-sensitive functions closer to the user or in a location that minimizes hop count.
- **Maximizing Throughput:** Optimizing resource allocation to handle high data rates for throughput-intensive services.

- **Improving Resource Utilization:** The controller can decide on the placement of shared functions to serve multiple slices, thereby improving overall network efficiency.
- **Balancing Costs:** The placement decisions can be optimized to reduce operational and capital expenditure.

These optimization parameters, informed by the performance data, allow the controller to continuously refine its function placement strategy to maintain optimal network performance in a dynamic multi-service environment.

2.3.2 Problem statement: Network Function Placement and Load Balancing

In the integrated TN–NTN O-RAN architecture considered in this deliverable, end-to-end slices can be deployed across terrestrial resources and the space segment. When slice functions are executed on LEO satellites, deployment decisions must account for LEO dynamics and limited, distributed onboard computing resources, while balancing the communication–computation trade-off that arises from routing data to the selected processing nodes.

Deploying slice functions in the space segment introduces additional complexity because processing demand varies with traffic patterns, service types, and active users, and accessing satellite resources may incur costs in communication, processing, and node-switching. Efficient deployment therefore requires jointly considering function demands and resource availability.

This section introduces the VNF placement and load balancing problem, first in separate form and then as a joint optimization [11]. We consider splitting the data flow across multiple satellite nodes to enable parallel processing, which can reduce processing costs at the expense of higher routing and node-switching overhead. Selecting the satellite nodes and the fraction of data processed at each node is therefore key to managing the space computing vs. communication trade-off.

2.3.2.1 System Model and Problem Formulation

The system under consideration is a multi-layered network designed to provide seamless connectivity and computing services to a diverse set of IoT devices. The architecture is composed of a set of M wireless users, denoted by $\mathcal{V} = \{v_1, \dots, v_m, \dots, v_M\}$, a constellation of S LEO satellites, denoted by $\mathcal{L} = \{l_1, \dots, l_s, \dots, l_S\}$, a terrestrial ground station G, and a cloud computing facility C. The LEO satellite nodes are a cornerstone of this model, as they are capable of providing EC services in space. Beyond their computing capabilities, these satellites also serve as crucial satellite relays, thereby forming a distributed in-space computing and communication environment that extends network services far beyond traditional terrestrial coverage. Additionally, the terrestrial ground station G is colocated with powerful edge computing servers, enabling it to serve users directly and act as a gateway to relay data to the distant, high-capacity cloud facility C when local resources are insufficient or for non-time-critical tasks.

We assume that remote devices are distributed across a large service area A and possess the capability to connect to multiple LEO satellites for communication and computing services. To access EC services in this NTN, each device establishes a connection to a specific satellite in the LEO constellation. This connected satellite is referred to as a source satellite or input satellite, denoted by l_s^{in}. This notation signifies that the s-th satellite acts as the initial point of contact for user data, receiving it from devices and potentially routing it to other satellite nodes for distributed processing. This dynamic selection of a source satellite is a key aspect of managing the network's on-demand resource allocation. The set of all such source satellites is defined as $\mathcal{L}^{in} = \{ l_s \in \mathcal{L} \mid l_s \text{ serves as a source node} \}$.In contrast, satellites that are not

currently involved in any communication or processing tasks are considered idle. These idle satellites represent a critical pool of unused resources that can be dynamically recruited as needed for data relaying or parallel processing to handle sudden load spikes or complex tasks. For any satellite $l_s \in \mathcal{L}$, the set of neighboring idle satellites is defined as $I_{l_s} = \{ l_j \in \mathcal{L} \setminus \mathcal{L}^{in} \mid l_j \text{ is idle and directly connected to } l_s \}$. Additionally, we define the global set of idle satellites as $\mathcal{L}^{idle} = \{ l_s \in \mathcal{L} \setminus \mathcal{L}^{in} \mid l_s \text{ is idle} \}$. The remaining satellites in the constellation may be actively participating in inter-satellite relaying or task execution and are grouped under $\mathcal{L}^{active} = \mathcal{L} \setminus (\mathcal{L}^{in} \cup \mathcal{L}^{idle})$.

This classification enables a flexible, role-based management of the satellite network for efficient utilization of in-space computing capabilities, allowing the network to adapt to varying traffic loads and service demands. The entire system is modeled in discrete time to simplify the dynamic nature of the satellite orbits. Network parameters are assumed to be constant throughout each time interval τ, where τ_i identifies the ith time interval.

Each LEO satellite l_s is characterized by a set of physical and operational parameters: its coverage footprint of radius R_{l_s}, a processing capacity c_{l_s} in FLOPS per CPU cycle, a CPU frequency f_{l_s}, and a total of $\mathcal{K}_{l_s}$ CPU cores. It is equipped with advanced communication capabilities designed to support both ground-level devices and high-speed inter-satellite links, using a communication technology with an overall bandwidth B_{l_s}. Finally, each satellite provides storage facilities with a maximum storage capacity equal to G_{l_s}, which is vital for temporary data caching.

During each time interval, the generic mth device may generate a slice request $\rho_k^m(\tau_i) \in \{\rho_1, \dots, \rho_k, \dots, \rho_K\}$ with a maximum of K distinct slices. The generic kth slice request is defined as $\rho_k = \left([f_{1,k}^{up}, \dots, f_{P,k}^{up}], [f_{1,k}^{cp}, \dots, f_{Q,k}^{cp}], T_{up,k}, T_{cp,k}\right)$, where $\mathcal{F}^{up} = [f_{1,k}^{up}, \dots, f_{P,k}^{up}]$ represents the ordered set of user plane functions, and $\mathcal{F}^{cp} = [f_{1,k}^{cp}, \dots, f_{Q,k}^{cp}]$ is the corresponding set of control plane functions. Each slice has specific Quality of Service (QoS) requirements, namely the user plane latency requirements $T_{up,k}$ and the control plane latency demand $T_{cp,k}$. Each generic function, whether it's a user plane function p or a control plane function q, is characterized by its processing and transmission data size and its computation requirements $f_{p,k}^{up} = (D_{p,k}^{up}, D_{p,k}^{up,tx}, \Omega_{p,k}^{up})$ and $f_{q,k}^{cp} = (D_{q,k}^{cp}, D_{q,k}^{cp,tx}, \Omega_{q,k}^{cp})$ where $D_{f,k}$ is the processing data, $D_{f,k}^{tx}$ is the transmission data, and $\Omega_{f,k}$ is the computation requirement in FLOPS.

Problem Formulation

The central challenge addressed in this work is the joint optimization of both slice function placement and dynamic load balancing. We define a binary variable that captures the core placement decision, which takes the value 1 if a slice function is placed on a considered computing node, and 0 otherwise, i.e.:

$$a_{f,k}^{e}(\tau_i) = \begin{cases} 1, \text{ if function } f \text{ of slice } k \text{ is placed on computing node } e \\ 0, \text{otherwise} \end{cases}$$

where e can be a satellite l_s, the terrestrial ground station G, or the cloud computing facility C. To handle the high computational demands and the dynamic nature of the satellite network, we introduce a second binary variable for load balancing, $\alpha_{f,k}^{s,s'}(\tau_i) \in \{0,1\}$, which allows a satellite s to share a processing load of function f with a nearby idle satellite $s' \in I_{l_s}$ for parallel processing. This operation is specifically applied to functions placed on satellite nodes, as they have the unique ability to dynamically leverage nearby idle resources.

The total cost is a combination of computation and communication expenses. These costs are modeled as follows.

Computation Time and Energy: The time required to compute a function f of slice k on node e is determined by its computational requirements and the node's processing capacity. The associated energy cost is directly proportional to this time and the node's power consumption for computation.

$$T_{c,e}^{f,k} = \frac{\Omega_{f,k}}{c_e f_e} \text{ and } E_{c,e}^{f,k} = T_{c,e}^{f,k} P_e^c$$

Communication Time and Energy: Data transmission between two nodes, m and n, involves three components: transmission, reception, and propagation.

$$T_{tx,mn}^{f,k} = \frac{D_{f,k}}{r_{mn}} \text{ and } E_{tx,mn}^{f,k} = T_{tx,mn}^{f,k} P_m^{tx}$$

$$T_{tx,mn}^{f,k} = \frac{D_{f,k}}{r_{mn}} \text{ and } E_{rx,mn}^{f,k} = T_{rx,mn}^{f,k} P_n^{rx}$$

$$T_{pr,mn}^{f,k} = \frac{d_{m,n}}{c}$$

where the propagation time is a function of the distance $d_{m,n}$ between nodes m and n and the speed of light c. The total communication time between nodes m and n along a path $\mathcal{P}_{m,n}$ is the sum of transmission, reception, and propagation times across all hops in the path.

Satellite Operating Costs: An additional cost is incurred when a satellite is switched from an idle, stand-by state to an active state to participate in processing or relaying. This cost is a function of the power $P_{sw,s}$ and time $T_{sw,s}$ required for the state transition:

$$E_{sw,s} = P_{sw,s} \cdot T_{sw,s}$$

The total latency and energy costs for each function are aggregated by combining these models, accounting for the load balancing decisions and node-switching overhead. The total cost for each slice is then the sum of the costs of its constituent functions.

The problem aims to minimize the total latency and energy costs associated with user and control plane functions for demanded slices. The overall cost can be minimized through the selection of proper edge nodes for placement and the distributed processing of slice function data, especially through load balancing.

The objective function is a weighted sum of the latency and energy costs for all slices, allowing for a flexible trade-off between these two critical metrics.

$$\min_{\mathcal{A},\alpha} \frac{1}{K} \sum_{k=1}^{K} \Big[\gamma_1 \left(w_1 T^{k,up}(\mathcal{A}^k, \tau_i) + w_2 T^{k,cp}(\mathcal{A}^k, \tau_i) \right) + \gamma_2 \left(w_1 E^{k,up}(\mathcal{A}^k, \tau_i) + w_2 E^{k,cp}(\mathcal{A}^k, \tau_i) \right) \Big]$$

where $\boldsymbol{\mathcal{A}}$ is a set of all network selection matrices for the K slices and α is a set of all load balancing matrices, the weights γ_1 and γ_2 control the relative importance of total latency and energy, while w_1 and w_2 adjust the balance between user and control plane metrics, subject to:

$$\mathbf{C1}: \sum_{\forall e} a_{f,k}^{e}(\tau_i) = 1 \qquad \forall f, k$$

$$\mathbf{C2}: \sum_{\forall k} \sum_{\forall f \in k} a_{f,k}^{e}(\tau_i)\Omega_{f,k}^{e} \leq \mathcal{K}_e c_e f_e \quad \forall e$$

$$\mathbf{C3}: \sum_{\forall k} \sum_{\forall f \in k} a_{f,k}^{e}(\tau_i)B_{f,k}^{e} \leq B_e \quad \forall e$$

$$\mathbf{C4}: T^{k,up}(\tau_i) \leq T_{up,k} \quad \forall k$$

$$\mathbf{C5}: T_{\tau_i}^{k,cp} \leq T_{cp,k} \quad \forall \mathrm{k}$$

These constraints are fundamental to ensuring a feasible and functional solution. C1 is a logical constraint that ensures each slice function is placed on a single edge node, preventing ambiguity in the placement decision. C2 imposes a crucial computation resource constraint, ensuring that the aggregate computational demand of all functions assigned to a node does not exceed its available capacity, thereby preventing bottlenecks. Similarly, C3 enforces the communication resource constraint, ensuring that the total transmission data of all assigned functions respects the node's available bandwidth. Finally, C4 and C5 are service-level agreement (SLA) constraints that enforce the strict latency requirements for the user plane and control plane of each slice, respectively, which is critical for supporting time-sensitive applications. This comprehensive formulation allows for the joint optimization of both function placement and load balancing to achieve the most efficient deployment while satisfying QoS demands.

2.3.3 Problem Statement: Beam Hopping

NTN, encompassing satellite communication systems (e.g., LEO, MEO, GEO satellites) and high-altitude platforms (HAPs), are a critical component of beyond-5G and 6G architectures. A defining challenge of NTN is the efficient management of limited spectrum and power resources to serve dynamic and geographically heterogeneous user demand. Beam hopping has emerged as a key technique to enhance flexibility, spectral efficiency, and capacity utilization in NTN [8].

2.3.3.1 Concept of Beam Hopping

Beam hopping is a resource allocation strategy in which a satellite's multi-beam coverage pattern is not continuously illuminated; instead, beams are activated sequentially in time according to a predefined or adaptive hopping pattern. Unlike conventional fixed multi-beam systems, where each spot beam is permanently illuminated, beam hopping dynamically allocates radio-frequency power and spectrum only to selected beams when and where user demand exists. This approach relies on temporal multiplexing, with beams activated in specific time slots, provides spatial flexibility by redirecting coverage in response to traffic distribution, and enables resource adaptation by concentrating power and bandwidth on active beams, thereby improving spectral efficiency [7].

2.3.3.2 Role in NTN

Beam hopping addresses fundamental NTN challenges by enabling dynamic and efficient resource management. In particular, it mitigates traffic demand variation arising from highly non-uniform user density and traffic load (e.g., urban versus oceanic areas) by allowing spatiotemporal adaptation of beam illumination. It also alleviates spectrum scarcity through time-domain reuse of spectrum across non-overlapping beams, thereby improving spectral efficiency. In terms of energy efficiency, beam hopping concentrates the transmit power on active beams, reducing wasted transmission energy in underutilized regions. Furthermore, by appropriately designing hopping sequences, beam hopping supports latency and QoS

adaptation, allowing prioritization of low-latency services or high-throughput sessions and enabling effective service differentiation.

2.3.3.3 System Design Considerations

Beam hopping in NTN requires optimization across several dimensions. First, hopping pattern design determines which beams are illuminated at a given time and can be either static, following a predefined schedule, or dynamic, adapting to traffic conditions or channel state. In addition, the hopping cycle must be carefully aligned with the physical-layer frame structure to avoid excessive signaling overhead. Power and spectrum allocation also play a key role, as they involve a trade-off between maximizing overall throughput and ensuring fairness among beams. Finally, efficient coordination between gateways and satellites is essential, since backhaul and feeder link design directly impacts the feasibility of real-time hopping control.

2.3.3.4 Technical Challenges

Beam hopping in NTN also introduces several technical challenges. Accurate synchronization is required, as ground terminals must reliably track the intermittent illumination of their serving beam. Frequent beam switching increases the complexity of handover management and may lead to higher signaling overhead and mobility-related challenges. In addition, interference management becomes more difficult when overlapping hopping patterns across adjacent beams or satellites induces inter-beam interference. Finally, dynamic beam hopping entails complex optimization, as it requires solving large-scale resource allocation problems under stringent latency constraints.

2.3.3.5 Current Research Directions

Current research on beam hopping in NTN is progressing along several key directions. One active area focuses on AI/ML-based beam hopping, where traffic prediction and reinforcement learning are leveraged to optimize hopping sequences in an adaptive manner. Another direction investigates the joint design of beamforming and beam hopping, combining spatial beamforming with temporal hopping to further improve spectral efficiency. Beam hopping is also being studied as an enabling mechanism for integration with terrestrial 6G systems, supporting seamless interoperability between NTN and TN. In parallel, standardization efforts within 3GPP Releases 17 and 18 are exploring beam hopping as part of the NR NTN architecture to enable more flexible and efficient coverage.

2.3.3.6 Application in O-RAN scenario

In O-RAN scenarios, beam hopping presents both opportunities and challenges. O-RAN's disaggregated and software-defined architecture enables near-RT control of beam hopping patterns through the RIC, facilitating multi-vendor interoperability and AI-driven optimization of hopping schedules. This architecture also strengthens NTN–TN integration by supporting dynamic resource sharing across domains. However, there are notable limitations, as O-RAN control loops may introduce additional latency that conflicts with fast beam hopping cycles, and the approach relies heavily on reliable fronthaul and backhaul links, which can be stressed in NTN environments. Beyond these limitations, several challenges remain, including harmonizing beam hopping algorithms with O-RAN's open interfaces (E2, A1, and O1), ensuring scalability when coordinating thousands of beams across heterogeneous RAN domains, and addressing security and resilience concerns arising from the programmable and open nature of O-RAN combined with dynamic NTN operations.

2.3.3.7 Beam hopping for Non-RT RIC and Near RT RIC

The beam hopping optimization on the Non-RT RIC and RT RIC is shown in Figure 2-2, and can consist of the following steps:

1. **Monitoring**

   Monitor the resource distribution across the beams by collecting data on packet arrival waiting in the satellite buffer, including their corresponding delays and the traffic arriving at that moment, and UE performance measurements capabilities. These UE measurement capabilities are treated as variables for processing and decision-making regarding beam hopping

2. **Analysis & decision:** consisting of the following steps:

   - Based on the data traffic, the performance, measurements of the UEs, and traffic prediction and then train the AI/ML model for beam hopping solution to have the optimal choice of beams to be illuminated, and the number of resources to be attributed (e.g. PRB, VNF resources …),

   - The AI/ML model decides each TTI on which beam to illuminate and how many resources will be given, this represents the AI/ML model inference.

3. **Execution:** each TTI based on the previous decision, execute the beam hopping solution, by reallocating the resources between the beams, and update the state of the buffer.

   - Re-configure the resources attribution to the cells based on the beam hopping solution,

   - Update the cloud resources via the O2 interface.

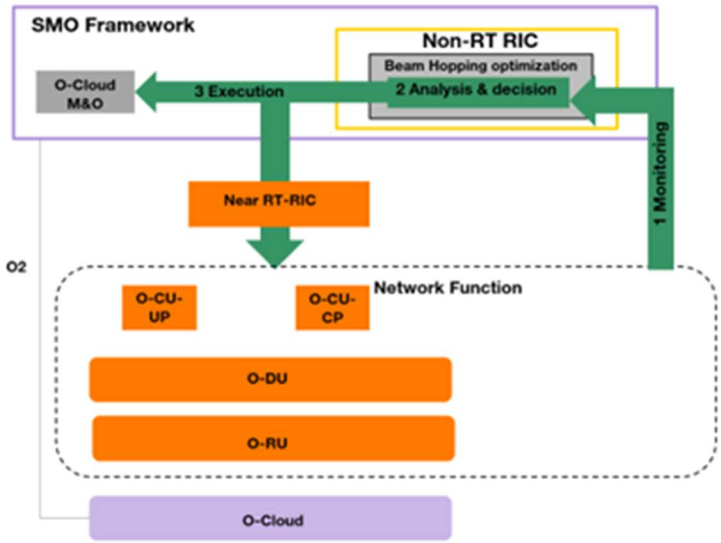


Figure 2-2: The realization of Beam Hopping optimization over non-RT RIC

2.3.3.8 O-RAN Beam hopping entities

Beam hopping control can be naturally mapped onto the O-RAN architecture by separating model training and long-term planning from near-real-time inference and enforcement. The following entities contribute to this closed-loop process at different timescales [1][2].

a. Non-RT RIC

Used for the training model and longer-timescale policy preparation. In this role, the Non-RT RIC leverages the management/analytics view of the network to build models from historical observations and to derive planning indications that can later be refined at runtime. In particular, it can:

- Collect measurements related to SINR, traffic demand, resource usage, etc., through the O1 interface (offline).
- Train the AI/ML model using historical measurement analysis to predict traffic demand.
- Identify the time, date, and location (i.e., specific O-RAN nodes) for adding or reducing resources (e.g., capacity, VNF resources, PRB per beam).

b. Near RT-RIC:

Used for the inference model and near-real-time decision making. Building on the model produced at Non-RT timescales, the Near-RT RIC operates with tighter latency constraints and converts recent measurements into enforceable actions, closing the loop with the RAN. Specifically, it can:

- Collect measurements related to SINR, traffic demand, resource usage, etc., through the O1 interface (in RT).
- Allocate the optimal resource blocks determined by the AI/ML decision (inference) to each illuminated beam via the E2 interface.
- Update cloud resources through the O2 interface.

c. RAN Nodes (O-CU-CP, O-CU-UP, O-DU, O-RU)

Execution and measurement endpoints for beam hopping control. These nodes provide the required observability to support training/inference and apply the actual configuration updates commanded by the controllers. Concretely, they

- Support performance measurement collection with the necessary granularity over the O1 interface.
- Support the data collection for model training and inference.
- Support resource allocation for beam updates via the E2 interface.

2.3.3.9 Beam hopping efficiency: impact of O-RAN function placement

This section aims to analyze the various deployment options for RAN functions as outlined in deliverable D5.3: Preliminary Report on AI-Data Driven Networking and QoS Management, for a beam hopping deployment. We provide an overview of these options and examine the VNF placement strategies associated with each. The discussion emphasizes identifying deployment configurations that enhance the efficiency of implementing beam hopping solutions within the O-RAN NTN architecture.

The efficiency of beam hopping depends on specific requirements related to VNF placements, and the data collection mechanisms impose additional conditions on these placements.

Here are the other requirements for the beam hopping solution to ensure optimal performance:

a. **Data collection for model inference constraint**: Based on the various types of data required for the beam hopping algorithm that is discussed in more detail in the next Section 2.5.1.1, it can be concluded that most of this data is collected from the gNB-CU (either CP or UP). Given the necessity for real-time optimization of the illuminated beam based on traffic arrival and other conditions, additional constraints are imposed on VNF placements. Specifically, the gNB-CU-CP and gNB-CU-UP should be located near the near-RT RIC, where the inference model will be executed, to ensure timely and efficient decision-making.

b. **Model training at the non-RT RIC (potentially offline):** There are no specific placement constraints for the non-RT RIC concerning the gNB-CU for data collection.

c. **Execution of beam hopping decisions via the near-RT RIC at the RAN functions:** Since model inference involves the beam illumination scheme and radio resource management between beams, the decision should target the gNB-CU, RU, and DU for complete execution. Therefore, it is recommended to have the gNB-RU and gNB-DU onboard with the gNB-CU.

This overview highlights the critical considerations for deploying RAN functions to support efficient beam hopping within the NTN architecture.

By consolidating all these requirements and constraints, we can conclude that options 4 and 5 from deliverable D5.3 (Figure 2-3) are the most suitable and efficient choices for implementing the beam hopping algorithm.

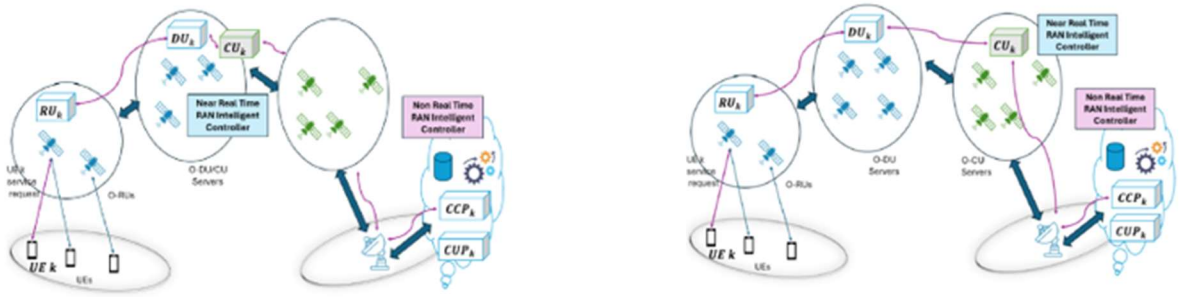


Figure 2-3: RAN Function Deployment

2.3.4 Problem Statement: Bearer Selection

Bearer selection is the function that enables the assignment of service flows to appropriate transmission paths, each represented by a bearer with defined QoS attributes. A bearer in the 5G architecture constitutes the logical instantiation of a QoS flow, ensuring that packets receive differentiated treatment according to service requirements. In NTNs, where connectivity is highly dynamic, the bearer selection function is decisive for guaranteeing end-to-end performance and service continuity [5].

2.3.4.1 Functional Principle

The operational principle is that multiple bearers coexist, each offering distinct trade-offs in terms of latency, throughput, reliability, cost, or availability. A user terminal may be simultaneously connected to two satellites, or to both a terrestrial gNB and a non-terrestrial gNB. Each of these connections can serve as a bearer, with the selection function assigning flows dynamically according to QoS profiles and real-time network measurements. The

process is adaptive: flows may be steered, switched, or split across bearers as conditions change. This ensures that critical services, such as ultra-reliable low-latency communication, are mapped onto bearers with minimal delay, while delay-tolerant or bulk traffic may be assigned to bearers with higher latency but more capacity.

2.3.4.2 Benefits and Limitations

The advantage of bearer selection is that it transforms heterogeneous connectivity into a unified and resilient service framework. Enabling terminals to leverage multiple simultaneous bearers improves reliability, balances load, and enhances spectrum utilization. It also enforces differentiated service treatment across diverse application classes, ensuring that stringent requirements are met even in dynamic environments. However, the function also introduces challenges. Frequent reassignment of flows may increase signaling overhead and complicate synchronization across domains. Prediction errors in link quality may lead to suboptimal assignments, particularly in the presence of fast-moving LEO satellites or impaired feeder links. Furthermore, in multi-vendor environments enabled by O-RAN, ensuring consistent implementation of bearer selection functions remains a non-trivial task.

2.3.4.3 Application in NTN and O-RAN

In the NTN context, bearer selection becomes essential because of the dynamic availability of satellite links. Orbital motion, inter-satellite handovers, and weather-induced feeder link impairments constantly alter the performance landscape of candidate bearers. Selection mechanisms must therefore integrate predictive models that anticipate link degradation and pre-emptively reassign flows to stable bearers. In O-RAN, the openness of the architecture allows the Near-RT and Non-RT RICs to host AI/ML models that optimize bearer assignments in real time, informed by performance monitoring across multiple access domains. This introduces powerful optimization opportunities but also requires harmonization of control logic across vendors and careful management of signaling complexity.

2.3.4.4 Architectural realization of bearer selection

Bearer selection extends the principle of multi-connectivity by introducing the ability to map QoS flows to differentiated logical bearers in a flexible and adaptive manner. Here we address the architectural realisation of bearer selection in heterogeneous 5G environments that integrate terrestrial and non-terrestrial domains. The motivation lies in the requirement to balance performance, cost, and reliability in scenarios where multiple paths coexist and where their characteristics vary significantly in time and space [6].

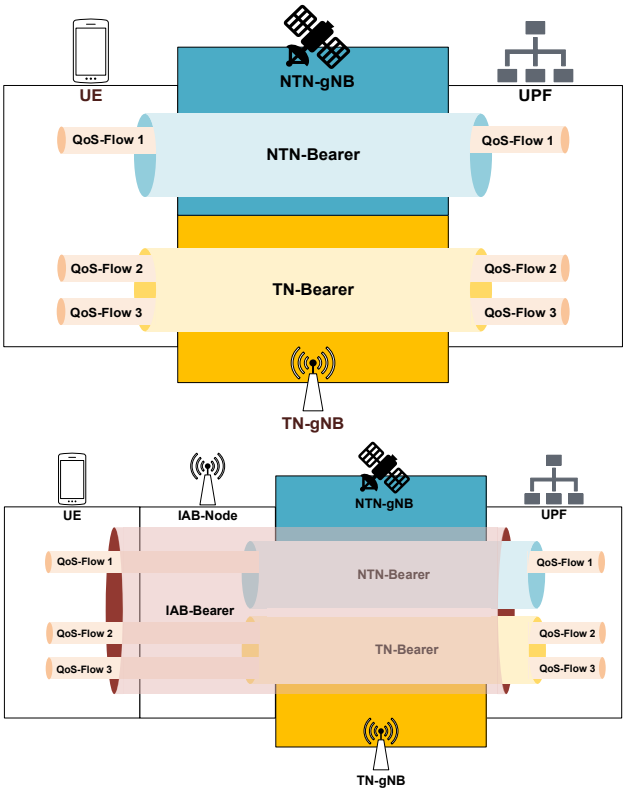


Figure 2-4: The two bearer models: (a) at the UE, (b) at the IAB node

The first architectural option places bearer selection at the UE. In this configuration, the terminal itself is aware of the simultaneous availability of multiple bearers, such as an NTN and a TN path, and it assigns traffic according to policy or measurement reports. As illustrated in the Figure 2-4 (a) the UE maintains parallel connections to both the terrestrial and the satellite domains, and it distributes traffic according to QoS requirements and link conditions. The IAB-node in this case acts as a transparent forwarder, while the UE executes the actual bearer decision logic. This solution preserves a high degree of flexibility for the user and aligns with ATSSS principles, but it also requires the device to handle coordination across links, which may increase complexity at the terminal side.

A second option becomes available in deployments where relay nodes, such as IAB-nodes, are present. In this case, the bearer selection function can be shifted to the relay, which terminates the terrestrial and satellite backhaul connections and provides a unified bearer to the UE. As shown in the Figure 2-4 (b), the IAB-node aggregates the different backhaul connections and hides the mapping logic from the terminal, thereby reducing device-side complexity. This approach centralises the coordination at the relay and can simplify terminal design, but it requires extensions of the relay nodes beyond current specifications, as well as careful dimensioning of the backhaul. If no relay node is deployed in the access network, this option is not applicable, and the bearer selection remains solely at the UE.

From an implementation perspective, UE-based selection leverages existing access traffic steering, switching and splitting (ATSSS) mechanisms and is feasible with advanced device capabilities, whereas IAB-based selection demands functional extensions of the relay nodes and adjustments to their protocol stack. In both cases, consistency of bearer assignment under rapidly changing link conditions remains the primary challenge, especially in LEO constellations with frequent handovers and variable feeder link availability.

In terms of performance, bearer selection enhances resilience by enabling redundancy across heterogeneous paths and optimizes resource usage by steering flows to the most suitable bearer. At the same time, it introduces potential overhead, particularly if frequent reassignment is triggered by satellite dynamics or by impairments of feeder links. Control of signaling load and harmonisation across vendors therefore become essential preconditions for successful deployment in integrated TN–NTN systems.

2.4 AI ALGORITHMS FOR O-RAN OPTIMIZATION

The selection of an appropriate AI/ML methodology depends on the specific control timescale and the nature of the optimization problem. The task requires a comparison between optimal, non-real-time methods and sub-optimal, near-real-time methods. Here, we analyze the trade-offs between three primary families of algorithms relevant to the 5G-STARDUST architecture.

- Deterministic Optimization (e.g., MILP):
  - Mixed-Integer Linear Programming (MILP) is a mathematical optimization technique used to find the best outcome in a given mathematical model for some list of requirements represented as linear relationships. It requires a complete and accurate model of the network environment, including all constraints and objectives.
  - **Pros:** Guarantees finding the optimal global solution.
  - **Cons:** Extremely high computational complexity, which grows exponentially with the problem size. It is not suitable for dynamic environments where conditions change rapidly.
  - **Applicability in O-RAN:** MILP is best suited for the non-RT RIC, where it can be used for strategic, offline tasks with a time horizon of seconds to hours. Examples include long-term resource planning or calculating an initial, baseline VNF placement strategy based on predictable traffic patterns.
- Probabilistic Methods (e.g., DBN):
  - Dynamic Bayesian Networks (DBNs) are probabilistic graphical models used to model sequences of variables. They are well suited for forecasting and reasoning under uncertainty.
  - **Pros:** Can effectively model the probabilistic nature of network events (e.g., traffic load, link quality) and predict future states.
  - **Cons:** Can become computationally complex for systems with very large state spaces.
  - **Applicability in O-RAN:** DBNs can be valuable as a component within the Non-RT or Near-RT RIC for predictive tasks. For example, a DBN could forecast near-term traffic load or satellite link availability, providing crucial input (as Enrichment Information over the A1 interface) to other decision-making algorithms, such as an RL agent.
- Reinforcement Learning (e.g., DRL/DQN):

- RL is a model-free approach where an agent learns an optimal policy by directly interacting with the environment and receiving feedback in the form of rewards or penalties. Deep RL (DRL) uses deep neural networks to approximate the value or policy function, allowing it to handle high-dimensional state spaces.

- **Pros:** Highly adaptive to dynamic and complex environments without requiring an explicit system model. It is scalable and suitable for real-time decision-making.

- **Cons:** Training can be data-intensive and time-consuming. The learned policy is often sub-optimal compared to deterministic methods.

- **Applicability in O-RAN:** DRL is the ideal candidate for tactical, closed-loop control in Near-RT RIC, operating on a timescale of 10ms to 1s. It is perfectly suited for dynamic challenges such as real-time beam hopping, bearer selection, and adaptive VNF load balancing.

    - **Training:** The DRL model can be trained offline within the non-RT RIC, leveraging historical data collected by the SMO via the O1 interface.

    - **Inference:** The trained model can be pushed to Near-RT RIC via the A1 interface. The Near-RT RIC then uses this model to make real-time decisions based on live data received from E2 nodes.

Based on this comparison, while MILP and DBNs have valuable roles for offline planning and prediction, the core challenge of dynamic, real-time management in the integrated TN-NTN network is best addressed by Reinforcement Learning. Therefore, this work will focus primarily on DRL techniques for implementing intelligent controllers.

2.4.1 AI Based VNF Placement and Balancing Solutions

In this deliverable, we propose a comprehensive and adaptive solution to jointly optimize both slice function placement and dynamic load balancing in a multi-layered satellite-ground network. The primary objective is to minimize the end-to-end (E2E) latency and energy consumption within the O-RAN multi-slice environment by making intelligent decisions about where to place slice functions and how to offload data for processing. This complex problem involves diverse slice demands, dynamic user traffic, and the highly mobile nature of the satellite constellation. Traditional one-shot optimization approaches, such as heuristic and meta-heuristic solutions, often struggle to find effective policies in such a dynamic and uncertain environment. Therefore, we have adopted a DRL based approach to learn near-optimal policies that can dynamically adapt to the evolving network conditions and service requirements.

The VNF placement and load balancing solution directly addresses the requirements of several key 5G-STARDUST use cases defined in D2.1. For instance, Vehicle Connected services like NG eCall require ultra-reliable, low-latency communication, which can be achieved by optimally placing critical functions (e.g., a UPF for local breakout) on a passing LEO satellite to shorten the data path. Similarly, for PPDR, the ability to dynamically deploy and balance functions ensures that resilient communication can be rapidly established in a disaster area where terrestrial infrastructure is compromised. Finally, for Global Private Networks, this solution enables the placement of functions to ensure data retention within a trusted domain and create shorter global data paths between distributed enterprise locations.

The joint optimization problem is modeled as a sequential decision-making process within the framework of a Markov Decision Process (MDP). In general, an MDP can be defined by a tuple

$\langle \Sigma, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, where Σ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward, $\mathcal{P}$ represents the state transition probabilities, and γ is the discount factor. Due to the inherent complexity and unpredictability of state transitions in a dynamic satellite environment, modeling the transition probabilities $\mathcal{P}$ is extremely challenging. Therefore, we employ a model-free RL approach, specifically a DQN-based solution, which can learn optimal policies without requiring an explicit model of the environment dynamics. In the following, we define the various elements of our MDP model for both the function placement and load balancing problems in detail.

2.4.1.1 Function Placement Scenarios and MDP Elements

For the function placement problem, it is crucial to consider the total number of slice requests, the users per slice, and the computing path available for each slice. To account for the dynamic nature of user demands and heterogeneous computing resources, we propose a scenario-based approach. Multiple independently trained DQN agents are used, each tailored to a specific slice-processing configuration based on the function placement path and the total slice load at time τ_i.

We define a set of scenarios associated with a total of K slices as:

$$\Psi_K = \{\psi_1^k, \dots, \psi_k^k, \dots, \psi_K^k | \forall k \in K\}$$

We define a mapping from each slice k to a scenario ψ_k, i.e.:

$$\text{Scenario for slice } k \rightarrow \psi_k = \left(\left|\mathcal{P}^k(\tau_i)\right|, N_{total}(\tau_i)\right)$$

where $\mathcal{P}^k(\tau_i)$ denotes the set of computing nodes (e.g., edge or cloud) that form the processing path for the kth slice request, and $N_{total}(\tau_i)$ is the total number of slices requested by all users.

2.4.1.2 State Space (Σ)

The state space for each of the two sub-problems is defined by different metrics to capture the relevant information for decision-making.

- **Function Placement State Space:** For a generic slice k with scenario Ψ_k, we define a state space $\Sigma^k = \{\sigma_1^k, \dots, \sigma_p^k, \dots, \sigma_P^k\}$, with a generic p-th state defined as $\sigma_p^k = \{(\kappa_{up}^k, \kappa_{cp}^k)\}$, where κ_{up}^k and κ_{cp}^k are binary states associated with the latency costs for the user and control planes, respectively. Specifically, $\kappa_{up}^k = 1$ if the user plane latency $T^{k,up}$ exceeds the requirement $T_{up,k}$ and 0 otherwise. Similarly, $\kappa_{cp}^k = 1$ if the control plane latency $T^{k,cp}$ exceeds its requirement $T_{cp,k}$ and 0 otherwise.

- **Load Balancing State Space:** For a generic slice function f deployed over a node n and an idle satellite $n' \in N'$, the state space $\Sigma(f,k)$ is defined by the cost outcomes of the load balancing operation. The state is represented as a tuple $\sigma_h^{(f,k)}(n,n') = (\sigma_T, \sigma_E)$, where σ_T is a ternary value (1, -1, or 0) indicating whether the offloaded computation time is greater than, less than, or equal to the local computation time, and σ_E is a similar ternary value for the energy cost.

2.4.1.3 Action Space (A)

The action space for each sub-problem provides the agent with a set of possible decisions to optimize performance.

- **Function Placement Action Space:** For a generic slice k, we define two action spaces: the user plane action space $A^{k,up}$ and the control plane action space $A^{k,cp}$. Each action corresponds to a valid sequence of function placements over the available computing path $\mathcal{P}^k(\tau_i)$. For example, a user plane action is a chain $(p_1, p_2, \dots, p_{|\mathcal{F}^{up}|})$ where each p_j is a node from $\mathcal{P}^k(\tau_i)$.

- **Load Balancing Action Space:** For a generic slice function f deployed over a node n and an idle satellite n', the action space $A^{(f,k)}$ consists of discrete actions that adjust the load balancing parameter $\alpha_{f,k}^{n,n'}(\tau_i)$. This parameter represents the ratio of the function's processing load that is offloaded from the local node n to the idle satellite n'. An action $a_p^{(f,k)}(n,n')$ is a choice from $\left\{\alpha_{f,k}^{n,n'}(\tau_i) - \Delta, \alpha_{f,k}^{n,n'}(\tau_i), \alpha_{f,k}^{n,n'}(\tau_i) + \Delta\right\}$, where Δ is a small, predefined step value. This allows the agent to incrementally increase, decrease, or maintain the current offloading ratio, enabling a fine-grained search for the optimal load distribution.

2.4.1.4 Reward Function (R)

The reward function for both sub-problems is designed to guide the RL agent toward the objective of minimizing total latency and energy costs. The reward function R associated with a slice function f of slice k is defined as the negative of the cost incurred, specifically the sum of computation and communication latency and energy:

$$R^{(f,k)}(\sigma, a) = -\left(\gamma_1\left(w_1 T^{k,up} + w_2 T^{k,cp}\right) + \gamma_2\left(w_1 E^{k,up} + w_2 E^{k,cp}\right)\right)$$

By maximizing this reward, the agent effectively minimizes the overall latency and energy costs for the requested slice.

2.4.1.5 Deep Q Network (DQN) based solution

Given the high-dimensional state-action spaces and the need for a model-free approach, we employ a DQN solution. The DQN methodology uses two neural networks: a primary network and a target network. The primary network approximates the Q-function, $Q(\sigma, a; \theta)$, which estimates the expected cumulative reward for taking an action a in state σ. The target network, with parameters θ', provides stable targets for the primary network's training.

For the load balancing problem, the DQN agent's goal is to learn the optimal load balancing ratio α for each function f at each node n. The agent receives the state, which is defined by the cost outcomes of the load balancing operation, and selects an action from the discrete action space to adjust the α value. This process is repeated over many steps, with the agent learning from the rewards received.

During training, transitions consisting of the current state, action, reward, and next state are stored in a replay buffer. Mini-batches are randomly sampled from this buffer to update the primary network parameters θ via gradient descent, minimizing the Mean Squared Error (MSE) loss between the primary Q-values and the target Q-values. The target values are computed using the Bellman equation. To capture the cost minimization objective, we use the minimum Q-value of the next state instead of the maximum, as is common in traditional Q-learning. The target network parameters are periodically synchronized with the primary network to ensure training stability.

The proposed DRL framework applies this DQN methodology to two separate RL agents for each slice function—one for function placement and one for load balancing. This hierarchical

approach allows the system to first determine the optimal placement of functions and then, in a separate but coordinated process, manage the load balancing for those functions.

Algorithm 1: RL for Multi-slice Function Placement in Satellite-Ground Networks

```
For each K in the set of slices:
    Generate the set of scenarios Ψ_K.
    For each scenario ψ_k in Ψ_K:
        Generate random user task sizes TS.
        Calculate hop distances and select candidate satellites that correspond to each hop.
        For each slice k in the set K:
            Randomly assign the user to a candidate satellite.
        Use Dijkstra's algorithm to calculate the shortest path to the cloud.
        Identify path satellites and idle satellites.
        For each satellite:
            Associate neighboring idle satellites.
        Construct augmented paths: ['U', path, 'GS', 'C']
            (where U = user, GS = ground station, C = cloud).
        Generate the state space SS and initialize replay buffers.
        For each slice u:
            Generate the action space for the user plane (UP) and control plane (CP).
            Initialize the Q-networks for UP and CP.
        For each episode e in episodes:
            Initialize a random input state In_State.
            For each step in max_steps:
                For each slice u:
                    Select UP and CP actions via the ε-greedy strategy.
                    Place the UP and CP functions along the path.
                    Count the occurrences of computation and communication functions.
                    Calculate the latency and energy via the cost model.
                    Calculate penalties if latency constraints are violated.
                    Calculate the total reward R_w.
                For each slice u:
                    Store the transition tuple in the replay buffer.
                    If there are enough samples:
                        Sample a mini-batch and update the Q-network.
                Update the state to Next_State.
            If episode % target_update_freq == 0:
                Update the target networks.
            If episode % 10 == 0:
                Print episode statistics.
        For each slice u:
            Save the final Q-networks for UP and CP.
```

Algorithm 2: DQN for Load Balancing O-RAN Slice Functions in Space

This algorithm outlines the training procedure for the DQN agent responsible for dynamic load balancing. The agent learns an optimal policy for load balancing by iteratively interacting with the environment, observing the costs, and updating its knowledge.

Requirements: State space Σ, action space $A^{(f,k)}$, replay buffer D, episodes N, steps I, exploration rate ε, discount factor γ, target update interval $\bar{\iota}$.

Output: Trained policy networks $\{\theta_f, \theta_f'\}$, for all functions f in the user and control planes.

1. For each function f in the user or control plane:
2. Initialize the primary network θ_f and set the target network θ_f' as a copy of θ_f.
3. For each episode ε from 1 to N:
4. Sample an initial state σ_0 from the state space Σ.
5. σ becomes σ_0 and i becomes 0.
6. While i is less than I:
7. i becomes $i + 1$.
8. Select an action a using the ε-greedy policy. The ε value gradually decreases over time to encourage the agent to shift from exploring new actions to exploiting its learned knowledge.
9. Update the offloading parameter $\alpha_{s,s'}^{f,k}(\tau_i)$ based on the action a. This action dictates the new ratio of data to offload.
10. Observe the next state σ' and the reward $R^{(f,k)}(\sigma, a)$. The environment provides feedback in the form of costs.
11. Store the transition $\langle \sigma, a, R^{(f,k)}(\sigma, a), \sigma' \rangle$ in the buffer D. This replay buffer allows the agent to learn from a diverse set of past experiences.
12. Update the DQN networks.
13. σ becomes σ'.

Algorithm 3: DQN Update Function for O-RAN Load balancing

Requirements: Replay buffer D, primary network θ, target network θ', iteration index i, target update interval $\bar{\iota}$, discount factor γ.

Output: Updated parameters θ, θ'.

1. Sample a random mini-batch of transitions $(\sigma, a, R^{(f,k)}(\sigma, a), \sigma')$ from the buffer D.
2. Calculate the current Q-values $Q(\sigma, a; \theta)$ using the primary network.
3. Calculate the target values $y = R^{(f,k)}(\sigma, a) + \gamma * min_{a' \in A^{(f,k)}} Q(\sigma', a'; \theta')$. The min function here is critical because our objective is to minimize costs (i.e., maximize the negative reward), so we want to select the action that leads to the lowest expected cost in the next state.
4. Calculate the loss using MSE: $L(\theta) = E\left[\left(y - Q(\sigma, a; \theta)\right)^2\right]$
5. Update the primary network parameters θ via gradient descent on $L(\theta)$.
6. If i modulo $\bar{\iota}$ is equal to 0:
7. Update the target network: θ' becomes θ. This periodic update ensures the target values are stable and prevents the training from becoming unstable.
8. Return θ, θ'.

This function details the core learning process of the DQN agent, where the neural network's weights are adjusted based on the observed rewards to improve the agent's policy.

2.4.1.6 Scalability and Complexity Analysis

While the proposed DRL framework offers a powerful and adaptive solution, its deployment in a full-scale mega-constellation as envisioned by 5G-STARDUST presents scalability challenges that must be addressed.

- **State-Action Space Growth:** The complexity of the VNF placement problem grows exponentially with the number of network slices (K), functions per slice (F), and available placement nodes (satellites, ground stations). The size of the action space can become intractably large, making it difficult for a single DRL agent to explore and learn an effective policy in a reasonable amount of time.

- **Computational Cost:**
  - **Training:** The training of the DQN models is computationally intensive and is best suited for powerful, ground-based servers within the SMO/Non-RT RIC framework. The use of a replay buffer and dual networks add to the memory and processing requirements.
  - **Inference:** The inference phase must be lightweight enough to be executed by the Near-RT RIC within its near-real-time control loop (10ms - 1s). As the neural network size increases to handle more complex state spaces, the inference time can become a limiting factor.
- **Mitigation Strategies:** To manage this complexity, a hierarchical or hybrid approach can be adopted. The non-RT RIC can run more complex, long-term optimization models (such as the MILP or a high-level DRL agent) to determine a strategic subset of valid placement options or policies. This information can then be passed down to Near-RT RIC via the A1 interface, effectively constraining the action space for the near-real-time agent. This reduces the learning complexity for the Near-RT RIC, allowing it to focus on tactical adjustments within the strategic boundaries set by the non-RT RIC, thus ensuring both scalability and real-time responsiveness.

2.4.1.7 Performance Evaluation

VNF Placement Problem

To analyze the performance of a proposed DRL-based function placement solution to process slice function data, we have implemented an integrated T-NTN scenario in a Python-based simulator with the support of various libraries, including NumPy, Keras, Matplotlib, Math, and Pandas. We have considered a LEO satellite network with 15 satellites distributed in three planes as shown in Figure 2-6. As shown in the Figure, we have considered input, cloud, and other generic satellite nodes to place slice functions. The inter-satellite and intra-plane distances are 500 and 100 Km. Each satellite node has 3 GFLOPs of computing resources to process slice data. We have considered up to 3 different slice requests with 50 users per slice request. These requests include a RU, three UPFs, and five CPFs, as illustrated in Figure 2-5.

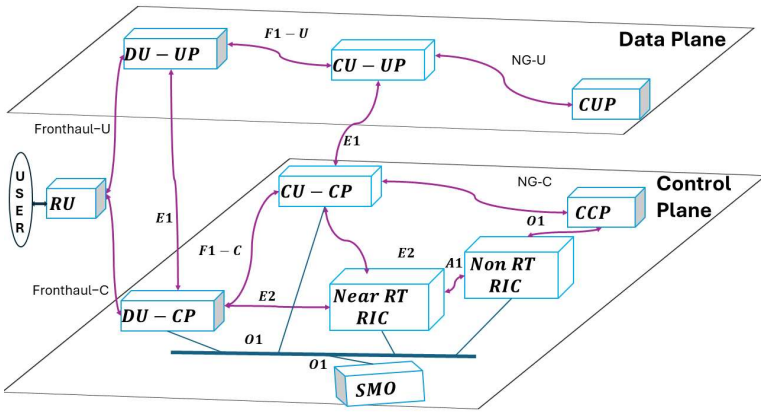


Figure 2-5: E2E O-RAN-based Architecture with UPFs and CPFs

Each slice request includes a RU, 3 UPFs, and 5 CPFs as defined in Figure 2-6. Additionally, a ground station with edge computing facilities and a cloud computing site is also included in a system model. The ground station has 1 GFLOPs of computing resources, while the cloud is able to provide 100 GFLOPs at the cost of longer communication distances. While

implementing the DQN solution, we have considered a 3-layer deep learning model with 20 episodes, 50 steps, γ=0.95, ε=0.1, batchsize=16, buffersize=10000.

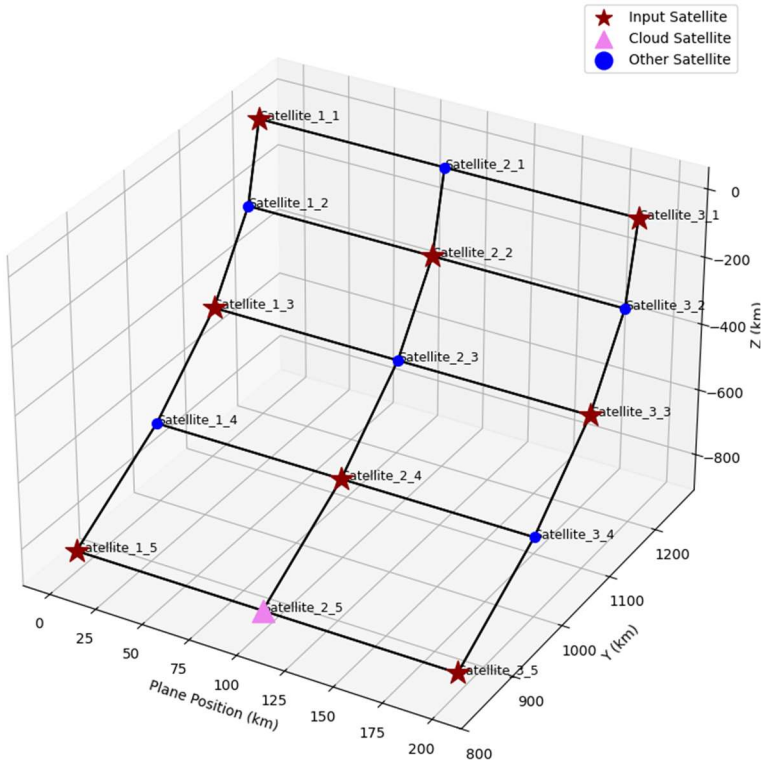


Figure 2-6: Multi-plane LEO Satellite Network

Benchmark Solutions

To evaluate the performance of our proposed DRL-based O-RAN slice function placement policy, we compare it against four distinct benchmark strategies. These strategies represent a range of fixed and non-adaptive placement paradigms, from fully centralized to fully decentralized, providing a robust baseline to highlight the advantages of our dynamic and intelligent approach.

- **Cloud-based Fix Function Placement Approach (CCFA):** The CCFA represents a traditional, fully centralized deployment model. In this strategy, all O-RAN functions for every network slice are placed on remote, terrestrial cloud servers. This approach leverages the vast computational resources of the cloud at the cost of high end-to-end latency, as all traffic must be backhauled from the satellites.

- **Input Satellite-based Fix Function Placement Approach (SSFA):** In direct contrast to the CCFA, the SSFA embodies a pure-edge computing model. Under this policy, all O-RAN functions for a given slice are placed on the initial LEO satellite that serves the ground users. This strategy aims to minimize propagation delay but is constrained by the limited onboard computational resources of a single satellite, creating potential performance bottlenecks.

- **Distributed Satellite-based Fix Function Placement Approach (SGFA):** The SGFA benchmark represents a static, hybrid cloud-edge strategy. In this approach, O-RAN slice functions are distributed between the LEO satellites and the terrestrial ground nodes

according to a predefined and fixed policy. While it uses hybrid architecture, it lacks the adaptability to respond to dynamic network conditions.

- **Random Satellite-based Function Placement Approach (RSGFA):** The RSGFA serves as a baseline for hybrid placement strategies. It distributes O-RAN slice functions across both LEO satellites and ground nodes, but the selection of the placement node for each function is done randomly from the set of all feasible nodes. It does not optimize for any specific performance metric and is used to establish a lower performance bound.

Numerical Results

Figure 2-7 illustrates the average E2E user-plane latency for O-RAN slices as a function of the number of active slice requests. Since all user-plane functions are placed on the same satellite, the ISFA approach incurs the minimum user-plane data transmission latency cost. In contrast, the random function placement solution suffers from the highest transmission cost, even worse than the cloud-based approach, due to the randomness in its placement policies. Both the fixed and RL-based function placement approaches improve user-plane data transmission latency compared with CCFA and RSFA. The performance of the RL-based approach can be further enhanced with a more extensive training process.

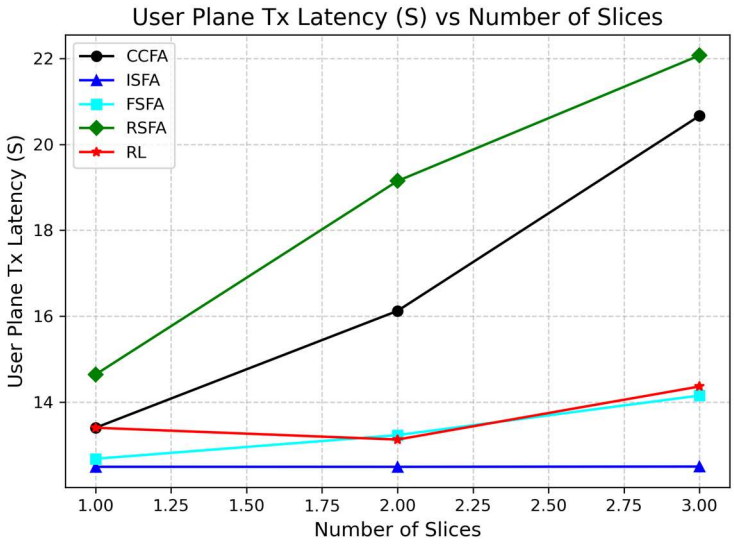


Figure 2-7: User-Plane Data Transmission Latency

Figure 2-8 shows the average E2E control-plane latency for O-RAN slices as a function of the number of active slice requests. Both ISFA and CCFA approaches require zero control-plane data transmission latency, as all control-plane functions are placed at the same node. The performance of the RL-based solution is currently unstable but can be improved with a more comprehensive training process. The FSFA solution may suffer from higher latencies,

especially as the number of slice requests increases. With random function placement policies, the RSFA approach cannot guarantee optimal performance.

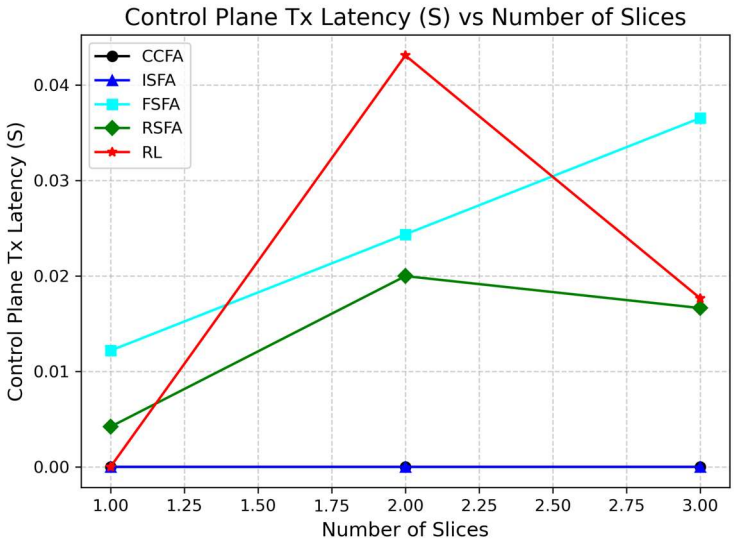


Figure 2-8: Control-Plane Data Transmission Latency

Figure 2-9 shows the average E2E user-plane data computation latency for O-RAN slices as a function of the number of active slice requests. The ISFA approach exhibits the worst performance due to stacking all slice functions on the same satellite, resulting in congestion and poor utilization of distributed satellite resources. The FSFA approach also experiences slightly higher latency, particularly as the number of active slices increases. The RL-based solution's performance can be improved with a proper training process. The RSFA approach may show improved performance but cannot guarantee consistent results due to the random

nature of function placements. The cloud-based CCFA approach achieves reduced computing latencies but incurs higher data transmission costs.

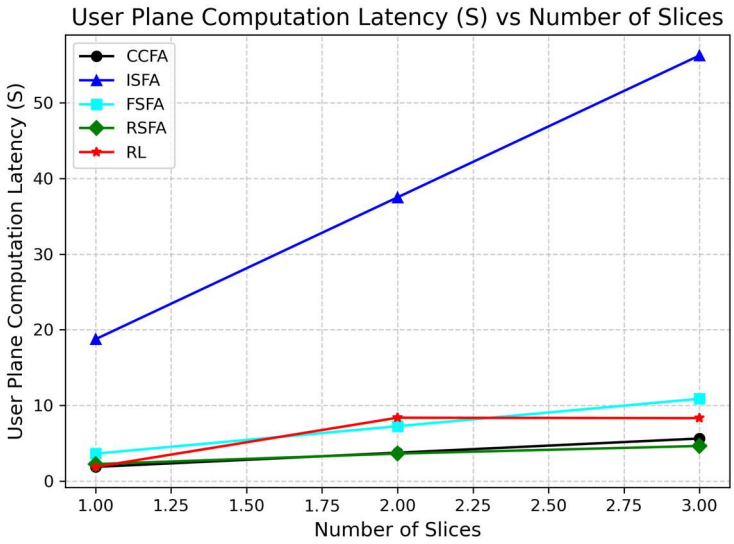


Figure 2-9: User-Plane Data Computation Latency

Figure 2-10 shows the average E2E control-plane data computation latency for O-RAN slices as a function of the number of active slice requests. Similar to the user-plane performance, the ISFA approach suffers from higher latency due to poor utilization of satellite resources. The other solutions exhibit trends similar to those observed in user-plane performance, while the RL-based solution's performance can be improved with more extensive training.

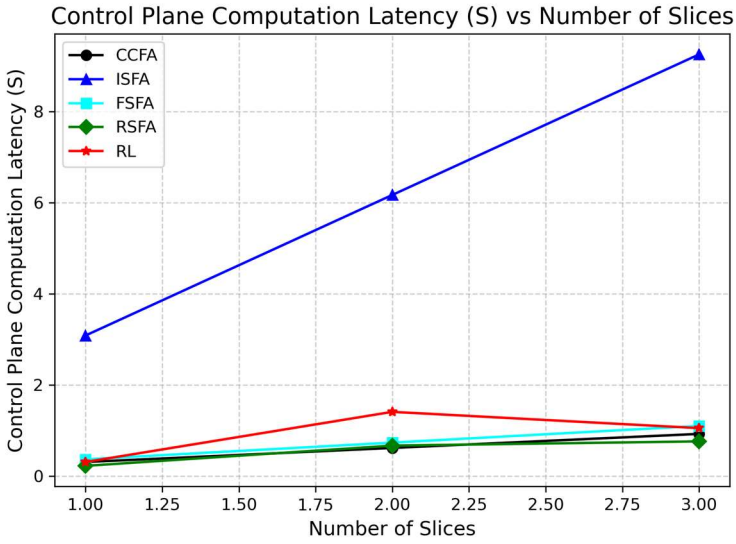


Figure 2-10: Control-Plane Data Computation Latency

Figure 2-11 shows the average E2E user-plane data transmission energy for O-RAN slices as a function of the number of active slice requests. The ISFA approach achieves the lowest and most stable transmission energy since all user-plane functions are placed on the same satellite, avoiding inter-satellite transmissions. In contrast, the RSFA approach suffers from the highest energy consumption due to inefficient and random function placements, which lead to

longer transmission paths. Both CCFA and RL approaches show moderate energy performance, with RL exhibiting room for improvement through more extensive training. The FSFA approach incurs steadily increasing energy costs as the number of slices grows but remains below the RSFA case.

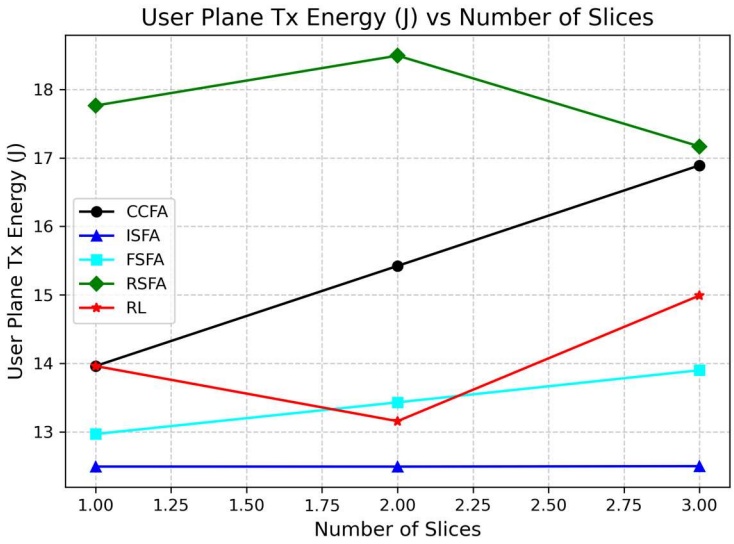


Figure 2-11: User-Plane Data Transmission Energy

Figure 2-12 illustrates the average E2E control-plane data transmission energy. Both ISFA and CCFA approaches result in nearly zero transmission energy since all control-plane functions are collocated at the same node. The FSFA approach shows a gradual increase in energy consumption as the number of slice demands grows, while the RL approach exhibits sharp peaks due to unstable training performance. The RSFA-based solution lies between ISFA and FSFA, with random policies.

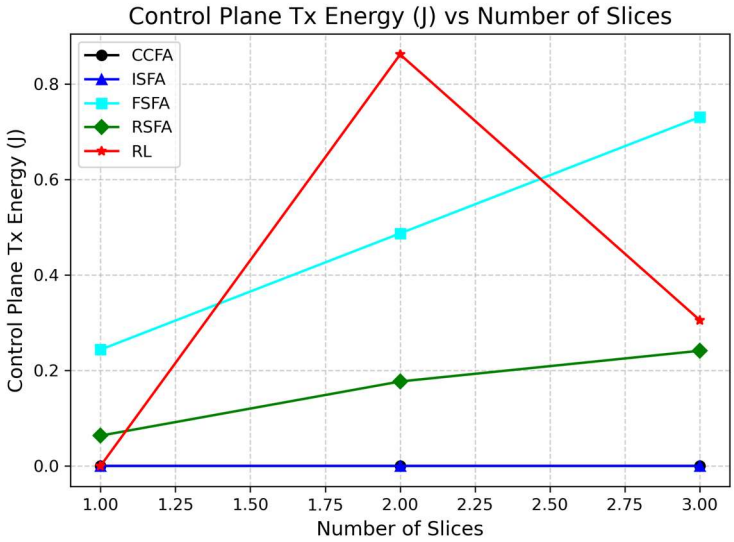


Figure 2-12: Control-Plane Data Transmission Energy

Figure 2-13 presents the average E2E user-plane data computation energy. The CCFA approach achieves the lowest computation energy by leveraging cloud resources, which are

more energy efficient. The FSFA approach remains relatively stable but higher than CCFA, while the RSFA approach decreases slightly with more slices due to unintentional balancing from randomness. The RL-based solution provides moderate performance, with further improvements possible through refined training.

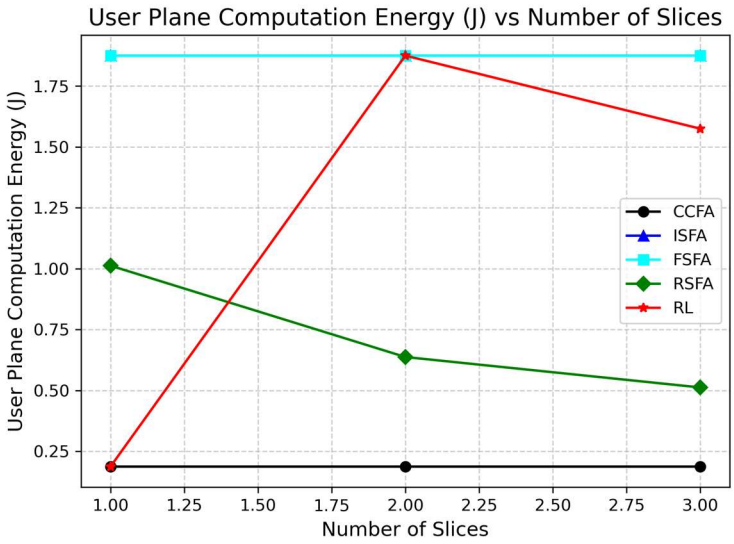


Figure 2-13: User-Plane Data Computation Energy

Figure 2-14 shows the average E2E control-plane data computation energy. The CCFA approach again maintains the lowest computation energy, while FSFA incurs moderate increases. The RSFA approach shows highly unstable behaviour with sharp peaks, reflecting inefficiencies from random placements. The RL-based solution demonstrates intermediate performance between FSFA and CCFA and could be further optimized with a more comprehensive training process.

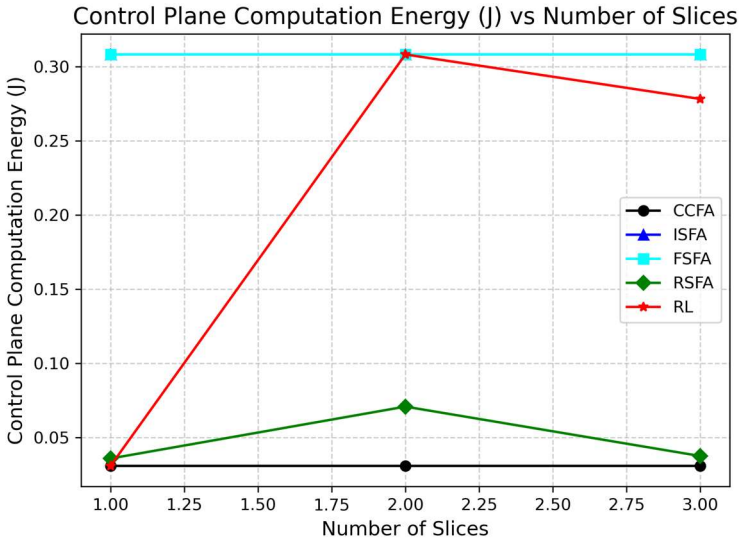


Figure 2-14: Control-Plane Data Computation Energy

Load Balancing Problem

In order to evaluate the performance of the proposed DRL-based load balancing solution for processing slice function data the same integrated T-NTN scenario considered for the network function placement problem has been used. Up to three distinct slice requests, each accommodating a variable number of users, have been considered.

Benchmark Solutions

To analyze the performance of the proposed DQN solution, we have considered the following benchmark methods. In all cases, the function placement is fixed at the selected LEO satellite nodes, i.e., $a^{e}_{f,k}(\tau_i)$ are pre-determined and not optimized.

- **Local Function Execution (No Offloading):** In this baseline, all O-RAN functions are processed locally at the assigned LEO satellites without load balancing. This corresponds to $\alpha^{f,k}_{s,s'}(\tau_i) = 0$, such that the entire function data is executed at the co-located satellite. The resulting cost is therefore given by the local communication and computation phases.

- **Random Approach (RA Offloading):** In this case, the load balancing ratios are assigned randomly, without considering the state $\sigma \in \Sigma$ or any system dynamics. Specifically, for each function f of slice k, the variable $\alpha^{f,k}_{s,s'}(\tau_i)$ is drawn uniformly at random from the feasible set $[0,1]$, subject to the constraint $\sum_{s'} \alpha^{f,k}_{s,s'}(\tau_i) = 1$. This random strategy serves as a naive baseline to highlight the performance gains of learning-based approaches. This approach is simple to implement and provides a naive baseline, but it completely ignores system dynamics and leads to highly sub-optimal performance.

- **Q-Learning Approach**: This method employs the classical tabular Q-learning algorithm, adapted to the cost minimization objective. At each iteration, the load balancing ratio $\alpha^{f,k}_{s,s'}(\tau_i)$ is updated by discrete steps $\{-\Delta, 0, +\Delta\}$ according to an ε-greedy policy over the action space $\mathcal{A}^{(f,k)}$. The Q-values are updated using the cost-minimization Bellman equation:

$$Q_{t+1}(\sigma, a) \leftarrow Q_t(\sigma, a) + \eta \left(R^{(f,k)}(\sigma, a) + \gamma \min_{a' \in \mathcal{A}^{(f,k)}} Q_t(\sigma', a') - Q_t(\sigma, a) \right)$$

  where η is the learning rate and γ is the discount factor. The greedy policy is obtained as

$$\pi(\sigma) = \arg \min_{a \in \mathcal{A}^{(f,k)}} Q(\sigma, a)$$

  with ε -greedy exploration during training. This method can exploit simple cost structures effectively, but it suffers from scalability issues in high-dimensional state-action spaces, motivating the need for function approximation through DQN.

Numerical Results

Figure 2-15 illustrates the average E2E user plane latency for O-RAN slices as the number of users per slice increases. With a fixed placement of UPFs across satellite and terrestrial computing nodes, the latency remains constant across all methods.

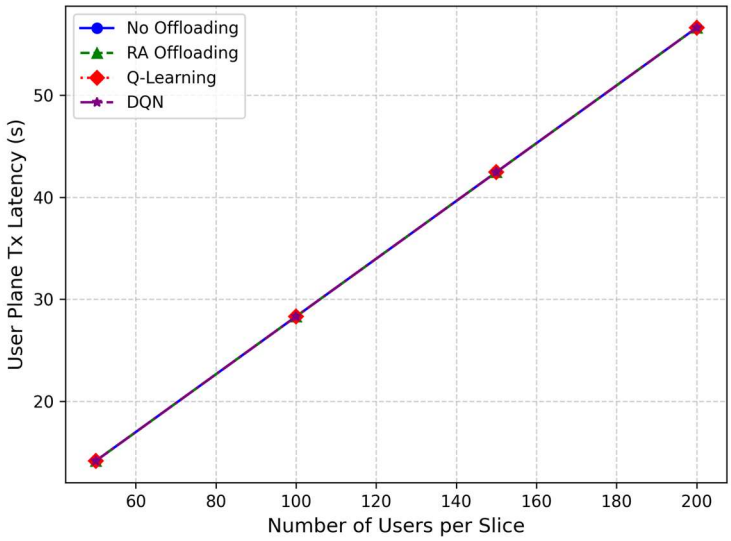


Figure 2-15: User-Plane Data Transmission Latency

Figure 2-16 shows the average end-to-end (E2E) control plane latency for O-RAN slices as the number of users per slice increases. Since the placement of CPFs on satellite and terrestrial computing nodes is fixed, latency values are identical across all methods. This analysis focuses on the influence of load balancing policies for O-RAN functions under fixed CPF placements.

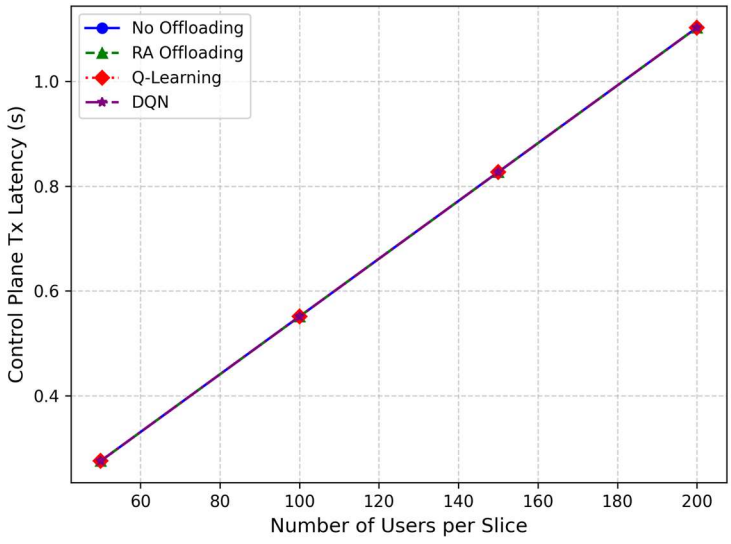


Figure 2-16: Control-Plane Data Transmission Latency

In Figure 2-17, we analyze the performance in terms of user-plane data computation latency with a varying number of users per slice. It can be observed that the DRL-based approach reduces the user-plane computation latency compared with local execution, highlighting the importance of distributing the computing workload in space. By intelligently allocating UPF workloads across idle satellites, DRL avoids congestion at overloaded nodes. However, achieving consistent performance gains requires the DQN training process to converge. In this work, we limited the training process to only a few iterations to avoid excessive complexity, which resulted in reduced DQN performance compared with RA and Q-learning solutions.

Although Q-learning demonstrates better performance than DQN, it faces scalability issues. On the other hand, RA offloading policies cannot guarantee optimized performance due to their random decision-making nature.

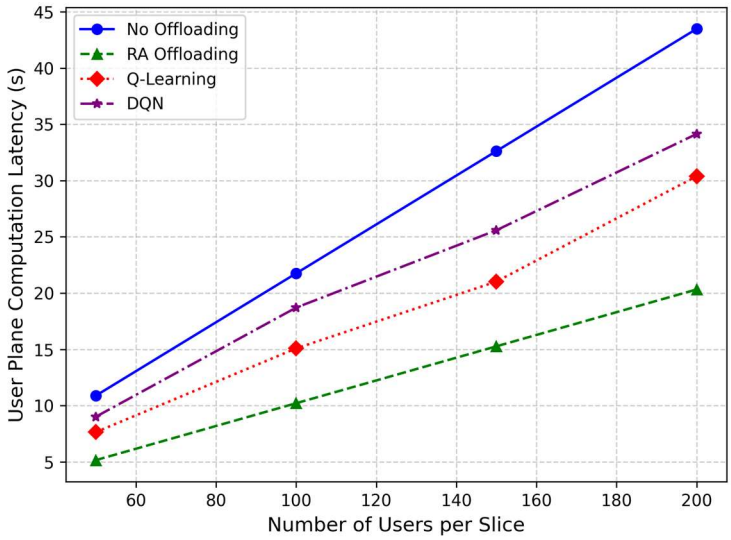


Figure 2-17: User-Plane Data Computation Latency

In Figure 2-18, we analyze the performance in terms of control-plane data computation latency with a varying number of users per slice. Similar to the user-plane performance, it can be observed that the DRL-based approach reduces the control-plane computation latency compared with local execution, especially as the number of slice users increases. However, the performance of DQN can be further improved through a more rigorous training process.

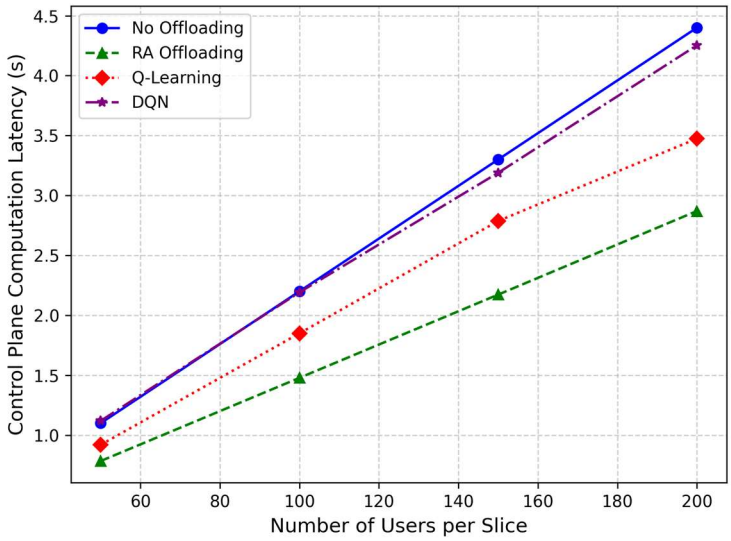


Figure 2-18: Control-Plane Data Computation Latency

Figure 2-19 illustrates the average end-to-end (E2E) user-plane data transmission energy for O-RAN slices as the number of users per slice increases. Similar to the latency performance,

when the placement of UPFs across satellite and terrestrial computing nodes is fixed, the energy cost remains constant across all methods for transmitting slice data.

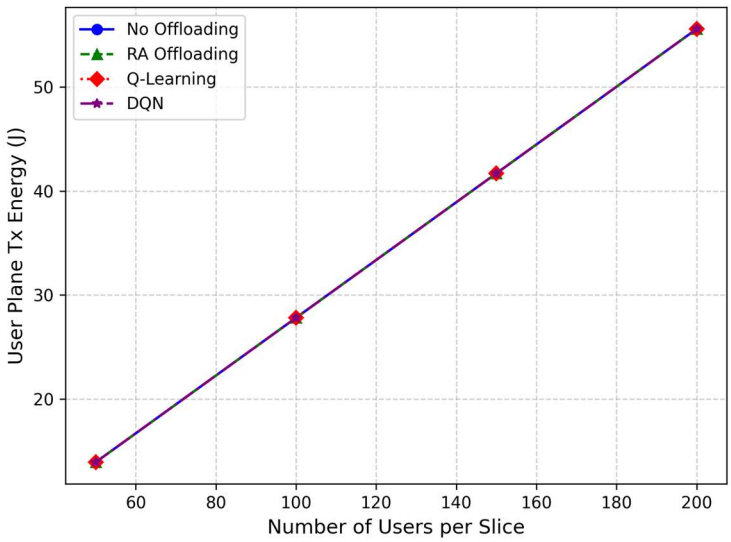


Figure 2-19: User-Plane Data Transmission Energy

Figure 2-20 illustrates the average end-to-end (E2E) control-plane data transmission energy for O-RAN slices as the number of users per slice increases. Similar to the user plane performance, when the placement of UPFs across satellite and terrestrial computing nodes is fixed, the energy cost remains consistent across all methods for transmitting slice data.

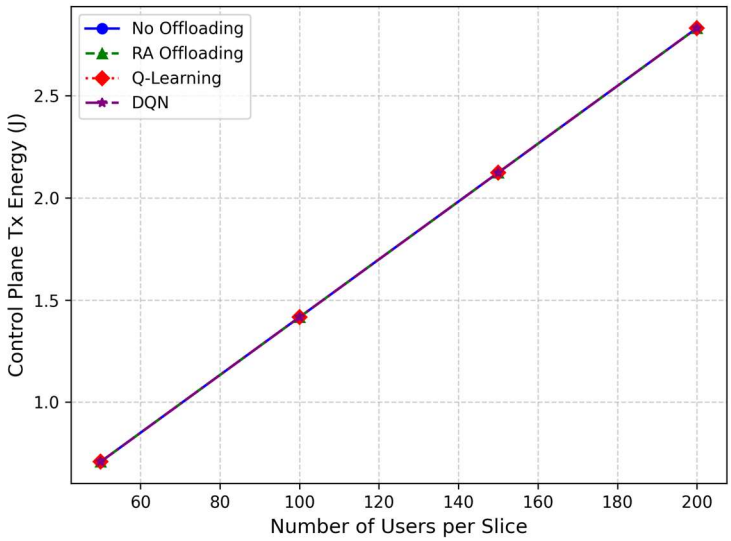


Figure 2-20: Control-Plane Data Transmission Energy

In Figure 2-21, we analyze the performance in terms of user-plane data computation energy with a varying number of users per slice. Distributing slicing function data across space can improve satellite resource utilization; however, it may also result in slightly higher energy costs.

This is evident in Figure 2-21, where the energy cost for processing slice data locally—without any load balancing—is marginally lower compared to load balancing-based solutions.

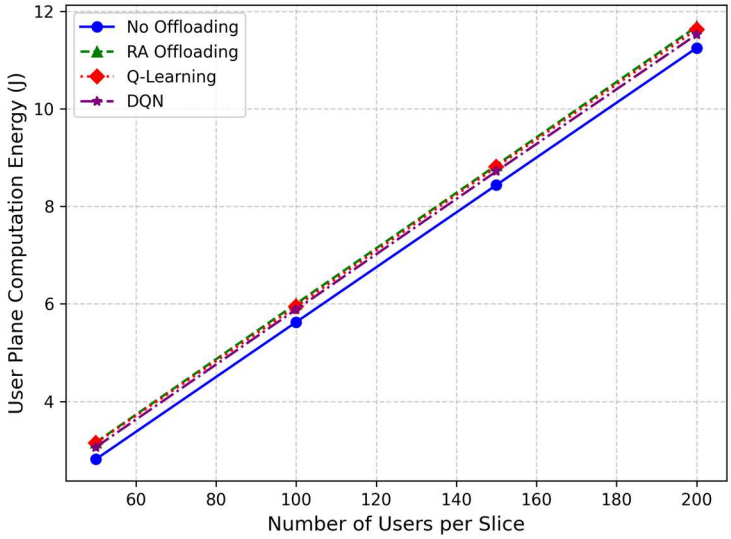


Figure 2-21: User-Plane Data Computation Energy

Similar to the user-plane performance, the energy required to process the control-plane function data shows comparable trends in Figure 2-22. Since the load balancing process

induces additional energy costs, it is important to optimize load balancing policies to ensure proper benefits.

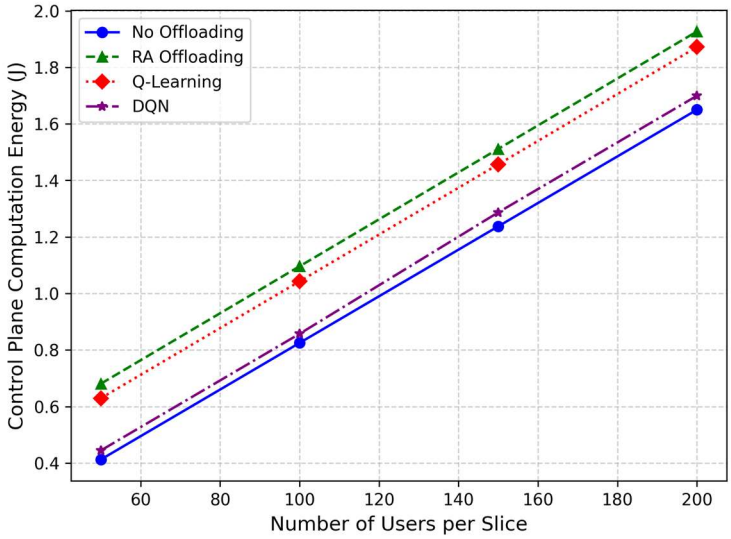


Figure 2-22: Control-Plane Data Computation Energy

Figure 2-23 shows the user-plane data transmission latency performance per slice. As expected, given the similar requirements of each slice and the fixed function placement approach, the latency performance remains the same across all three slices.

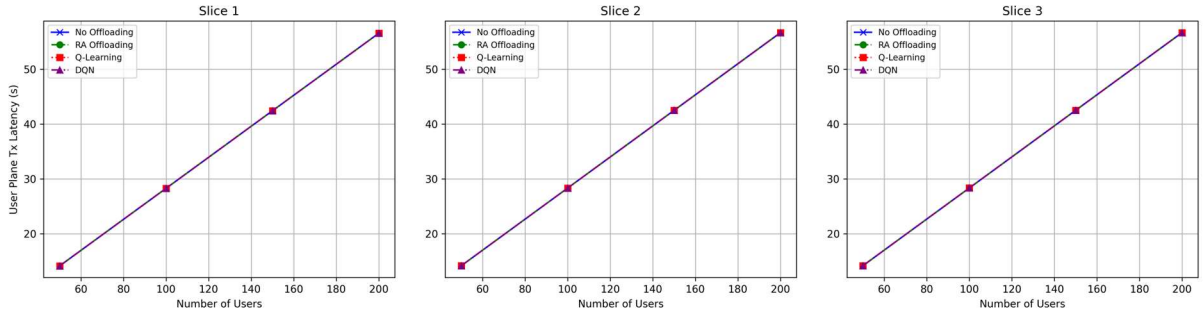


Figure 2-23: Per-slice User-Plane Data Transmission Latency

Similarly to the performance of the user plane, Figure 2-24 shows the latency performance of the data transmission from the control plane per slice. As expected, given the similar

requirements of each slice and the fixed function placement approach, latency performance remains the same across all three slices.

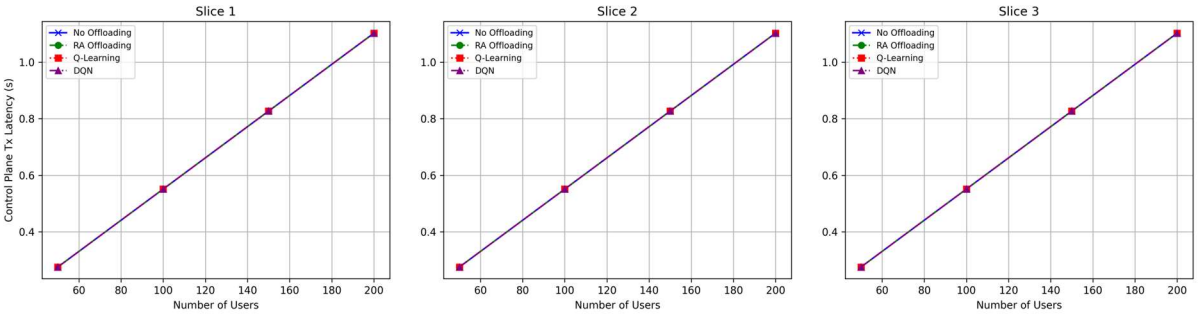


Figure 2-24: Per-slice Control-Plane Data Transmission Latency

Figure 2-25 shows the per-slice performance in terms of user-plane data computation latency with a variable number of users. It can be seen that, with integrated load balancing, different slices exhibit different latency requirements. For the random approach, the performance of slices one and two is better, while the performance of slice three degrades compared to the Q-learning solution. This highlights the importance of jointly optimizing the load balancing

process in all slices. The DQN performance, with a limited training process, appears slightly worse than RA and Q-learning; however, we expect it to improve with more training iterations.

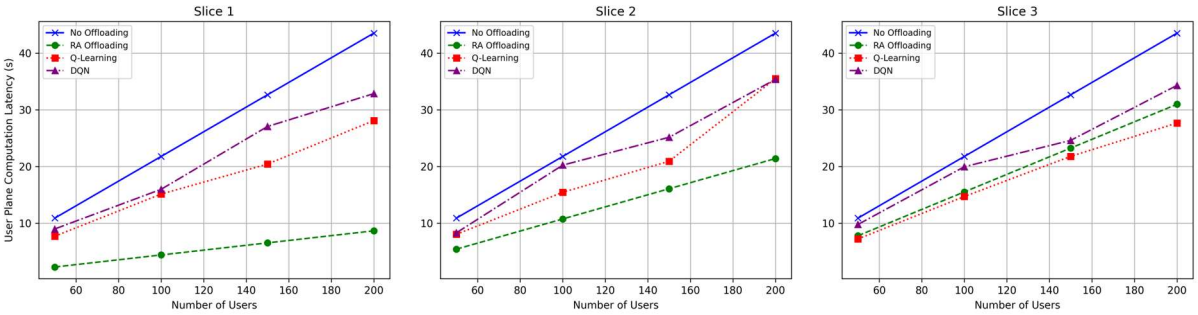


Figure 2-25: Per-slice User-Plane Data Computation Latency

Similarly to the latency performance of the user plane, Figure 2-26 shows the performance per slice for the latency of the data computation of the control plane over the increasing number of users per slice. The performance of DQN can be improved with a proper training process.

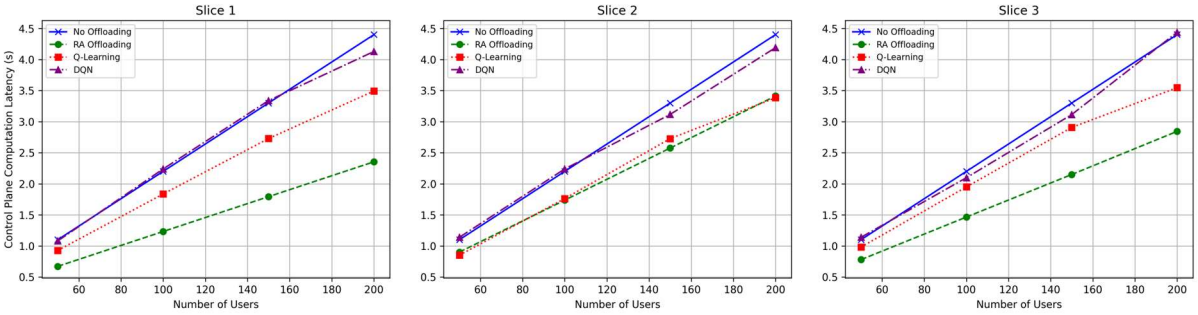


Figure 2-26: Per-slice Control-Plane Data Computation Latency

In Figure 2-27, the per-slice energy performance is shown to transmit data from the user-plane function. With similar requirements and fixed function placement, performance remains the same across all slices.

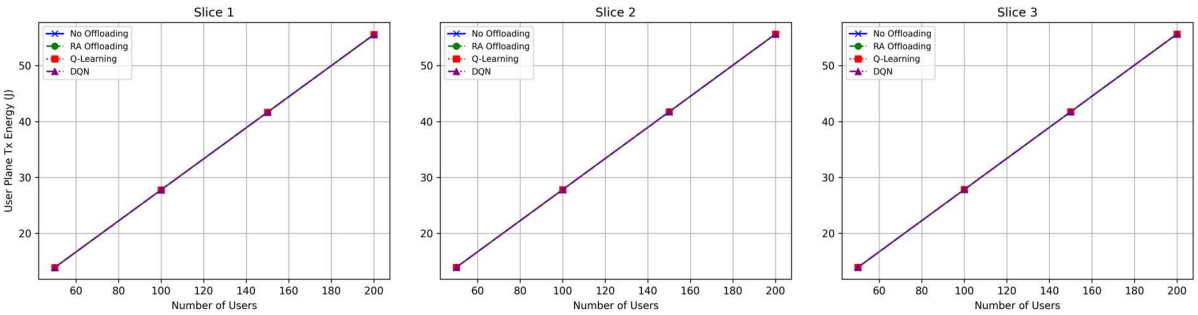


Figure 2-27: Per-slice User-Plane Data Transmission Energy

In Figure 2-28, the per-slice energy performance is shown to transmit the control-plane function data. With similar requirements and fixed function placement, performance remains the same across all slices.

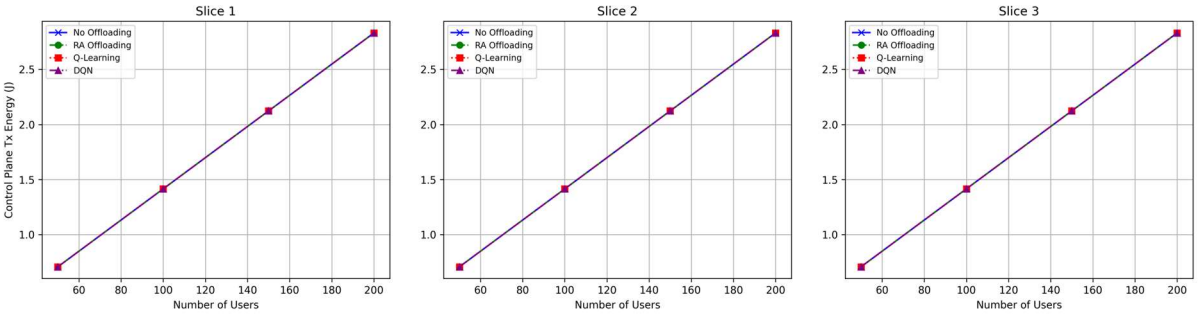


Figure 2-28: Per-slice Control-Plane Data Transmission Energy

In Figure 2-29, the per-slice energy performance for computing user-plane function data is shown over a varying number of slice users. Similarly to overall energy performance, this figure also highlights a slight increase in energy cost due to load balancing. However, this trade-off can be beneficial in terms of latency performance, as demonstrated in the figures related to latency analysis.

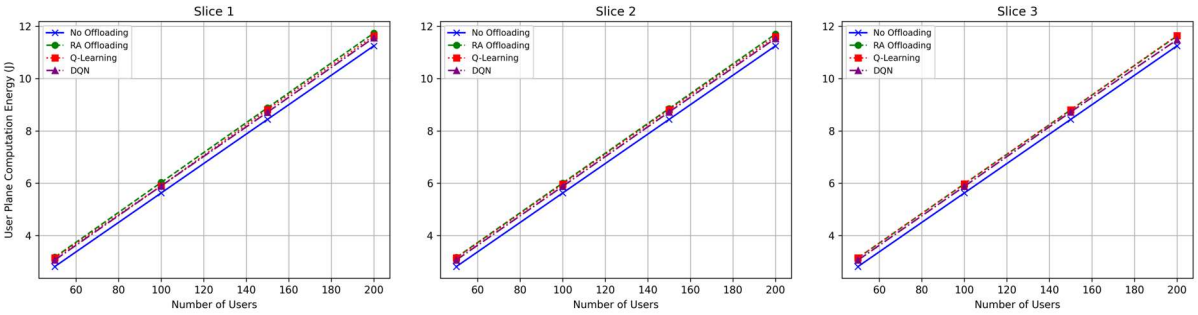


Figure 2-29: Per-slice User-Plane Data Computation Energy

In Figure 2-30, the per-slice energy performance for computing control-plane function data is shown over a varying number of slice users. Similarly to overall energy performance, this figure also highlights a slight increase in energy cost due to data offloading. However, this trade-off

can be beneficial in terms of latency performance, as demonstrated in the figures related to latency analysis.

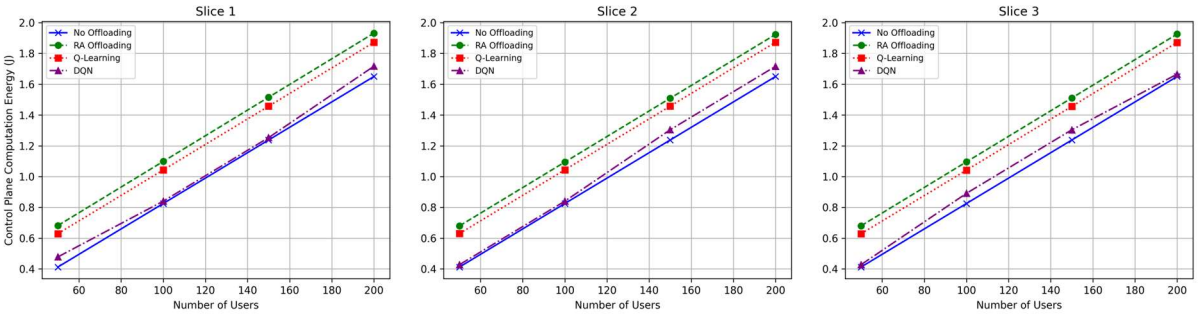


Figure 2-30: Per-slice Control-Plane Data Computation Energy

Joint Network Function Placement and Load Balancing

In this section the numerical results for the joint Network Function Placement and Load Balancing problem are shown. In particular we assume to jointly optimize the NFV placement through the DQN approach and following this the load balancing solutions previously introduced are applied in order to balance the load among the different satellite nodes. To this aim the same parameters and benchmark solutions are used.

Numerical results

Figure 2-31 illustrates the average end-to-end (E2E) user-plane latency for O-RAN slices as the number of users per slice increases. With a DQN-based UPF placement and load balancing approach, the overall latency can be reduced compared to static function placement

without load balancing. The performance of the DQN solution can be further improved with proper training.

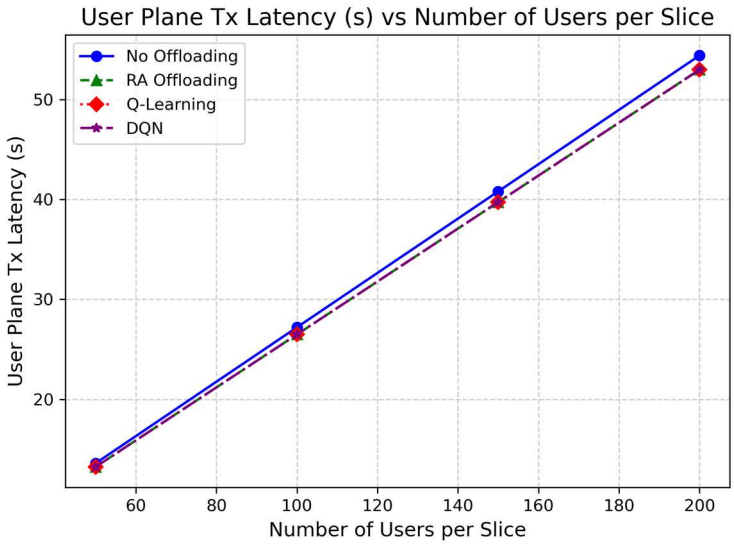


Figure 2-31: User-Plane Data Transmission Latency

Figure 2-32 shows the average end-to-end (E2E) control plane latency for O-RAN slices as the number of users per slice increases. The performance of the DQN solution can be further improved with proper training.

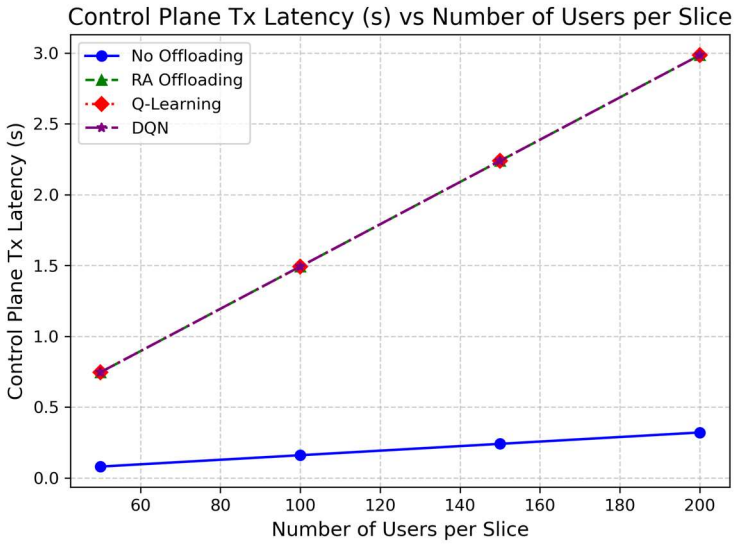


Figure 2-32: Control-Plane Data Transmission Latency

In Figure 2-33, we analyze the performance in terms of user-plane data computation latency with a varying number of users per slice. It can be observed that the DRL-based approach reduces the user-plane computation latency compared with local execution, highlighting the importance of proper function placements and distributing the computing workload in space. By intelligently placing the functions in space and allocating UPF workloads across idle satellites, DRL avoids congestion at overloaded nodes. However, achieving consistent performance gains requires the DQN training process to converge. In this work, we limited the

training process to only a few iterations to avoid excessive complexity, which resulted in reduced DQN performance compared with RA and Q-learning solutions. Although Q-learning demonstrates better performance than DQN, it faces scalability issues. On the other hand, RA offloading policies cannot guarantee optimized performance due to their random decision-making nature.

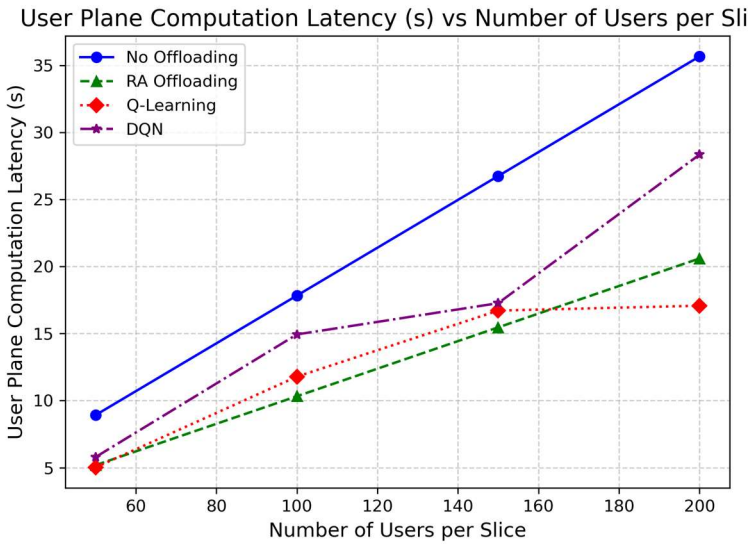


Figure 2-33: User-Plane Data Computation Latency

In Figure 2-34, we analyze the performance in terms of control-plane data computation latency with a varying number of users per slice. Similar to the user-plane performance, it can be observed that the DRL-based approach reduces the control-plane computation latency compared with local execution, especially as the number of slice users increases. However, the performance of DQN can be further improved through a more rigorous training process.

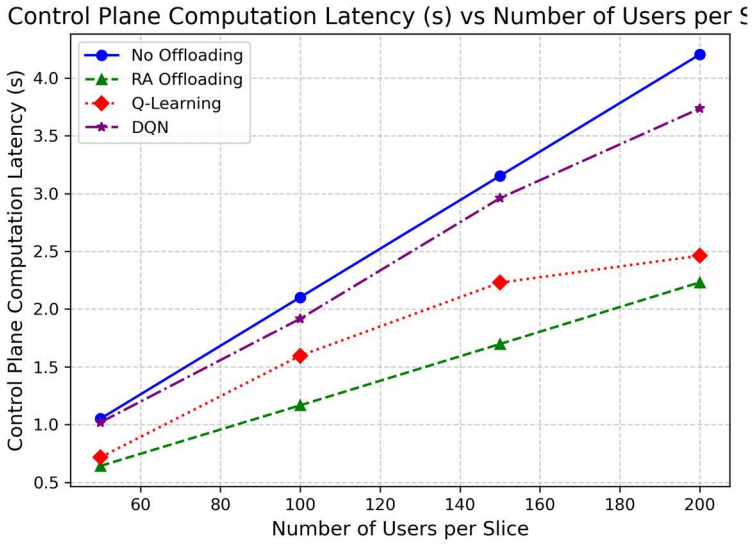


Figure 2-34: Control-Plane Data Computation Latency

Figure 2-35 illustrates the average end-to-end (E2E) user-plane data transmission energy for O-RAN slices as the number of users per slice increases. Similar to the latency results, the

energy performance of the DQN-based function placement and load balancing approach is improved compared to the solution where function data is not distributed across satellite nodes.

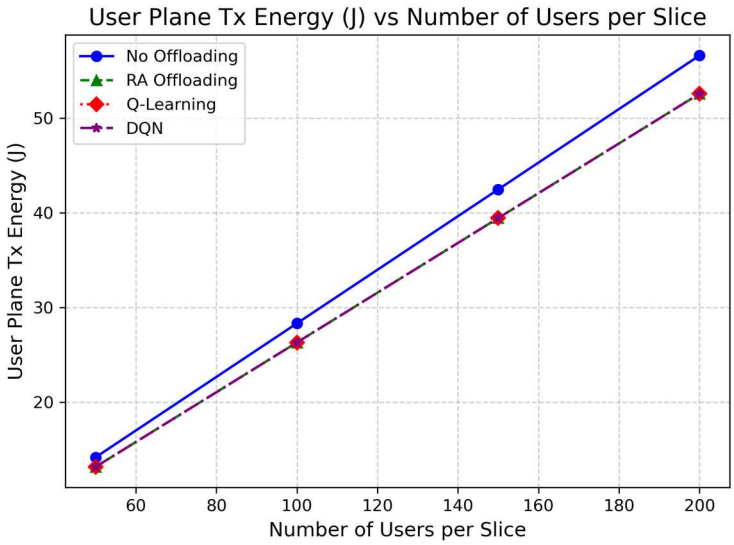


Figure 2-35: User-Plane Data Transmission Energy

Figure 2-36 illustrates the average end-to-end (E2E) control-plane data transmission energy for O-RAN slices as the number of users per slice increases. DQN performance is expected to improve with a more rigorous training process.

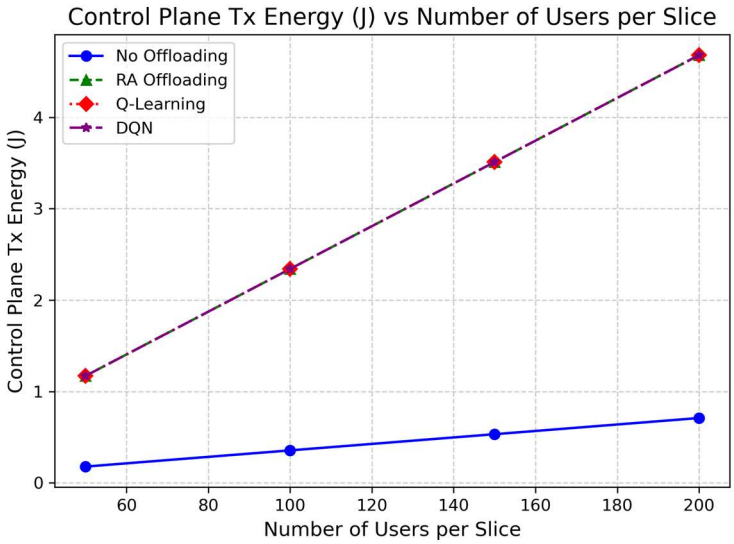


Figure 2-36: Control-Plane Data Transmission Energy

In Figure 2-37, we analyze the performance in terms of user-plane data computation energy with a varying number of users per slice. Defining appropriate function placements and then jointly distributing slice function data across space can improve satellite resource utilization, resulting in reduced energy costs. This is evident in Figure 2-37, where the energy cost for

processing slice data locally—without any load balancing—is higher than the proposed DQN-based solution.

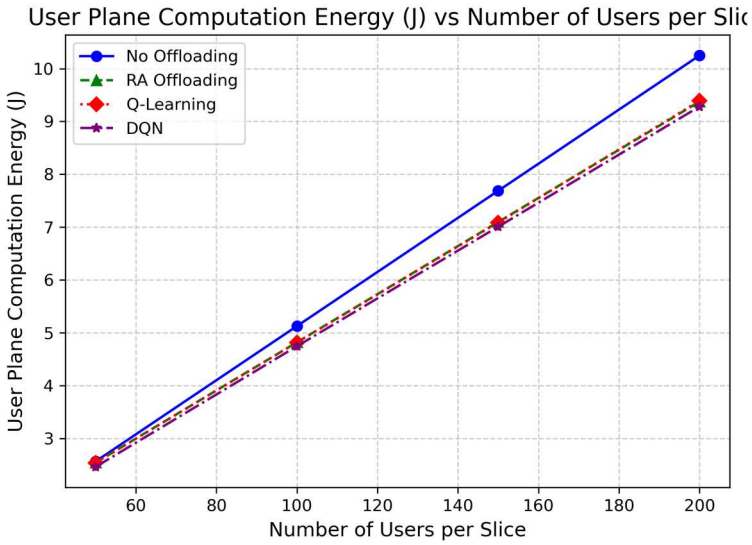


Figure 2-37: User-Plane Data Computation Energy

Similar to the user-plane performance, the energy required to process the control-plane function data shows comparable trends in Figure 2-38. The DQN performance is slightly

reduced compared to the one without load balancing. This gap in performance is expected to reduce with higher training iterations.

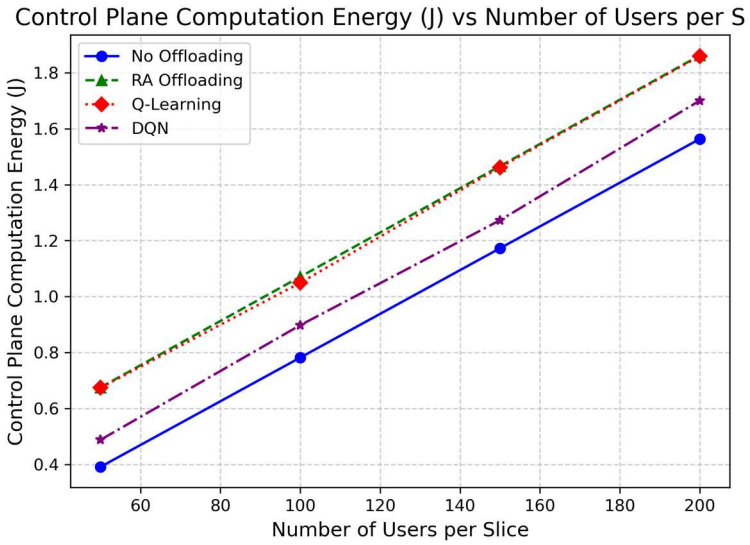


Figure 2-38: Control-Plane Data Computation Energy

Figure 2-39 shows the user-plane data transmission latency performance per slice. Different slices with similar requirements can induce differing latency costs based on function placements and load balancing decisions.

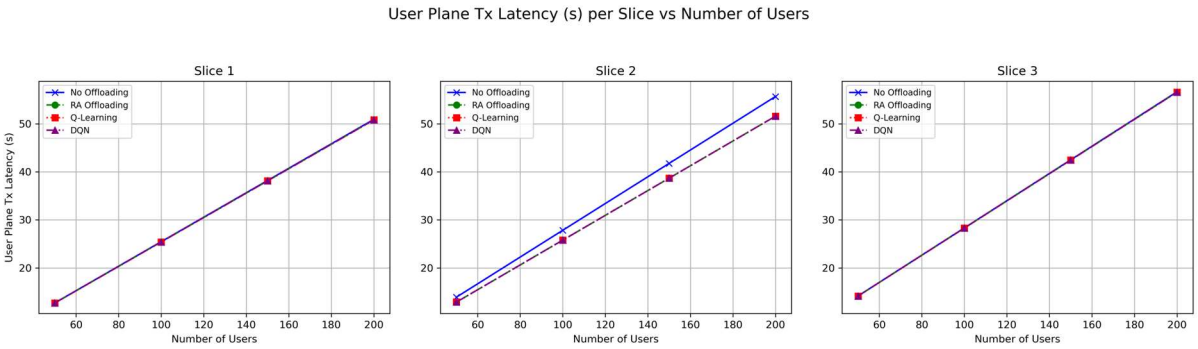


Figure 2-39: Per-slice User-Plane Data Transmission Latency

Similar to the user plane performance, Figure 2-40 shows the control-plane data transmission latency performance per slice. As expected, the latency performance varies across all three slices.

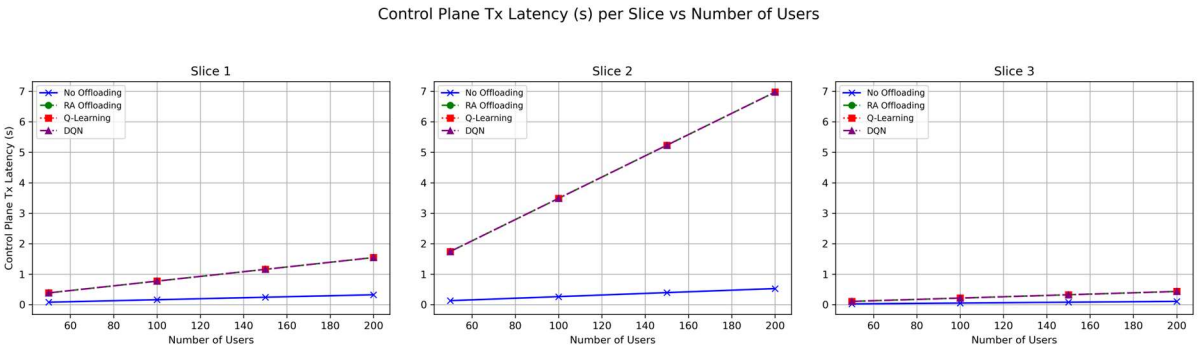


Figure 2-40: Per-slice Control-Plane Data Transmission Latency

Figure 2-41 shows the per-slice performance in terms of user-plane data computation latency with a varying number of users. It can be observed that with integrated function placement and load balancing, different slices exhibit different latency requirements. For the random offloading approach, the performance of slices one and three is better, while the performance of slice two degrades compared with the Q-learning solution. This highlights the importance of jointly optimizing the function placements and load balancing process across all slices. The DQN performance, with a limited training process, appears slightly worse than RA and Q-learning; however, we expect it to improve with more training iterations.

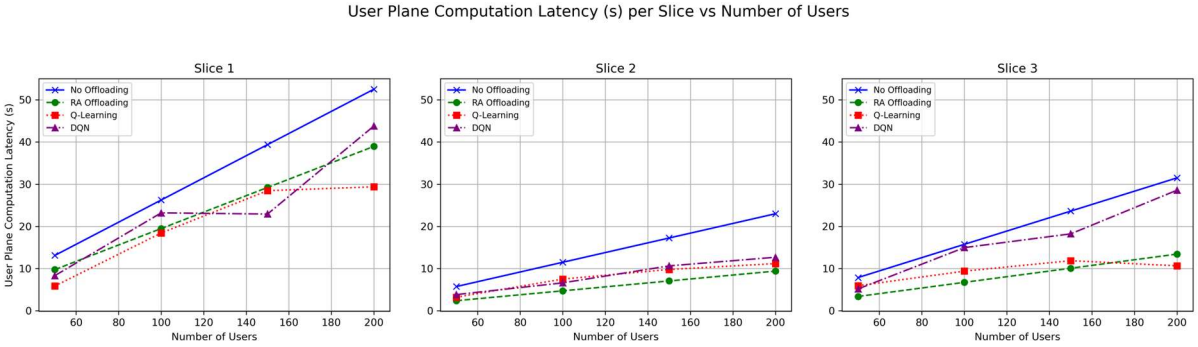


Figure 2-41: Per-slice User-Plane Data Computation Latency

Similar to the user plane latency performance, Figure 2-42 shows the per slice performance for the control plane data computation latency over increasing number of users per slice. The performance of DQN can be improved with proper training process.

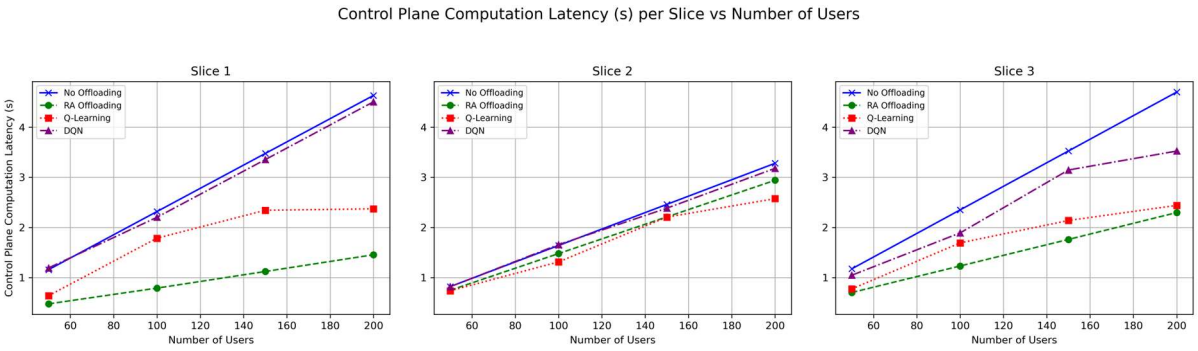


Figure 2-42: Per-slice Control-Plane Data Computation Latency

In Figure 2-43, the per-slice energy performance for transmitting user-plane function data is shown. The DQN performance is slightly reduced compared to the one without load balancing solution. However, the performance is expected to improve over higher training iterations.

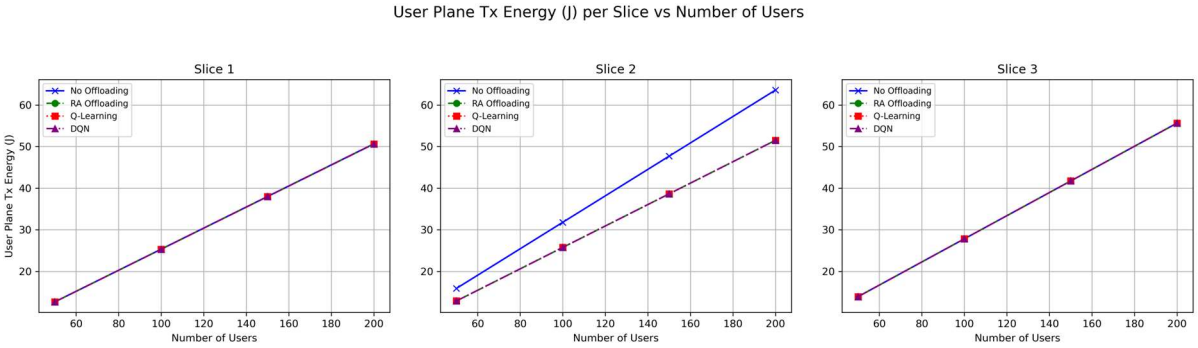


Figure 2-43: Per-slice User-Plane Data Transmission Energy

In Figure 2-44, the per-slice energy performance for transmitting control-plane function data is shown.

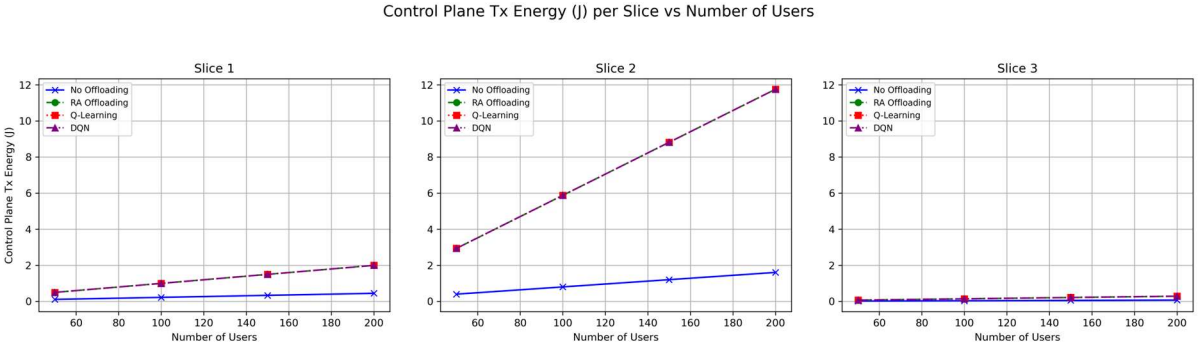


Figure 2-44: Per-slice Control-Plane Data Transmission Energy

In Figure 2-45, the per-slice energy performance for computing user-plane function data is shown over a varying number of slice users. Similar to the overall energy performance, this figure also highlights a slight increase in energy cost due to load balancing. However, this trade-off can be beneficial in terms of latency performance, as demonstrated in the figures related to latency analysis.

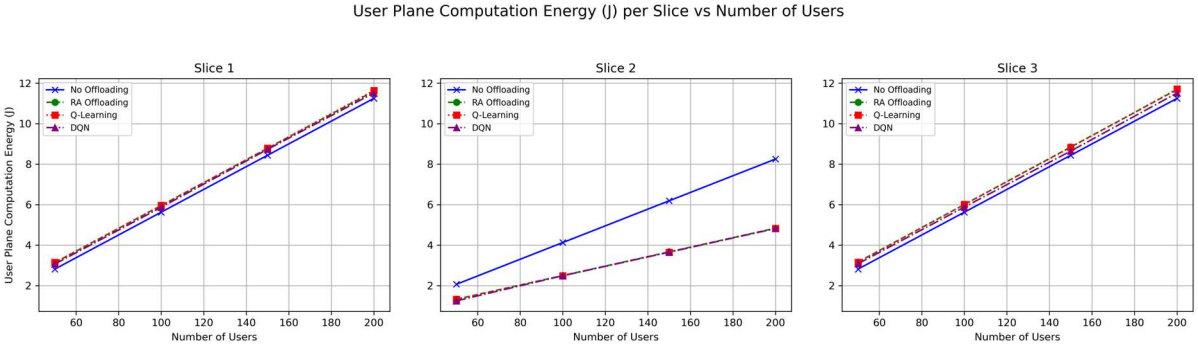


Figure 2-45: Per-slice User-Plane Data Computation Energy

In Figure 2-46, the per-slice energy performance for computing control-plane function data is shown over a varying number of slice users. Similar to the overall energy performance, this figure also highlights a slight increase in energy cost due to load balancing. However, this trade-off can be beneficial in terms of latency performance, as demonstrated in the figures related to latency analysis.

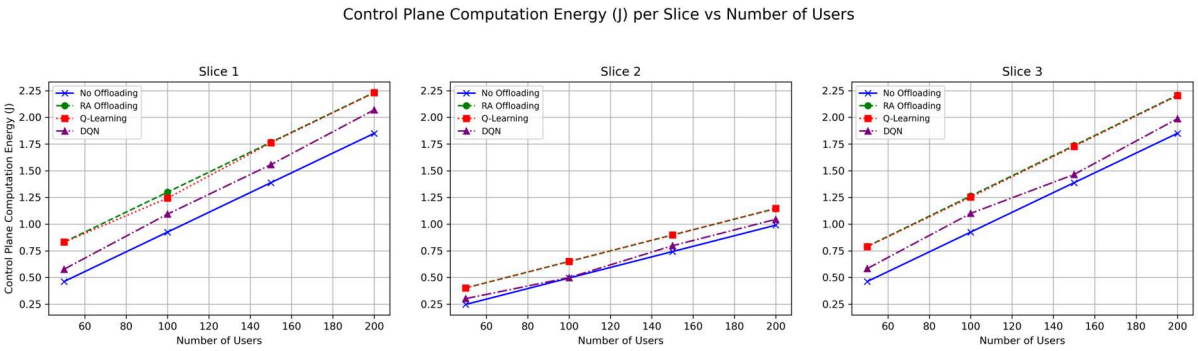


Figure 2-46: Per-slice Control-Plane Data Computation Energy

2.4.2 AI based Beam Hopping

This subsection describes the proposed AI/ML-based beam hopping solution and its integration in the O-RAN control loop [9]. The aim is to automate beam illumination and resource assignment based on traffic and radio conditions.

2.4.2.1 Description of the AI/ML proposed solution for beam hopping

This subsection describes the proposed AI/ML solution for beam hopping within the O-RAN TN–NTN architecture. It summarizes the problem formulation, defines the learning components (state, action, and reward), and outlines the required inputs and control outputs for execution through the Non-RT and Near-RT RIC functions.

1) Problem Summary

Goal: Dynamically select which K out of N cells to illuminate using satellite beams and allocate radio resources among them.

Inputs:

- Traffic demand in each cell (arrival rate, type: RT and non-RT)
- Number of beams K
- Total radio resource blocks available (**R**)

Output:

- Beam hopping decision: selected **K** cells to serve
- Resource allocation per beam (share of **R**)

Reward:

The learning objective reflects a trade-off between service differentiation goals:

- Minimizing latency for real-time traffic
- Maximizing throughput for non-real-time traffic

This framing allows the controller to adapt beam activation to spatial/temporal traffic variations while preserving QoS differentiation.

2) Deep Q-Learning Design

State (S)

The state represents the network status at each time step and includes, for each cell: traffic type (RT/NRT), arrival rate a_n, and current buffer occupancy b_n. A flattened representation of this multi-cell state vector is used as input to the DQN.

Action (A)

The action combines two coupled decisions [10]:

- **Beam selection:** choose **K** cells to illuminate
- **Resource allocation:** distribute **R** resource blocks among the selected cells

To keep the action space tractable, two design options can be considered:

- **Discrete action space:** predefine a limited set of beam-selection and resource-split combinations
- **Hybrid design:** use DQN for beam selection and a complementary regressor/actor module for continuous resource allocation

Reward (R)

A composite reward is used to encode QoS priorities for heterogeneous traffic, for example:

$$\text{Reward} = \alpha \cdot f(\text{latency}_{RT}) + \beta \cdot g(\text{throughput}_{NRT})$$

where latency_{RT} is the average latency of real-time traffic in the selected cells and throughput_{NRT} is the aggregate throughput of non-real-time traffic. The weights α and β control the trade-off and are tuned empirically.

Q-Network Architecture

The Q-network maps the state vector to Q-values associated with the available actions:

- **Input:** flattened state vector
- **Hidden layers:** fully connected layers (e.g., 2–3 ReLU layers)
- **Output:** Q-values per action

If the action space becomes large or learning stability is a concern, variants such as Double DQN and/or Dueling DQN can be adopted to improve convergence and robustness.

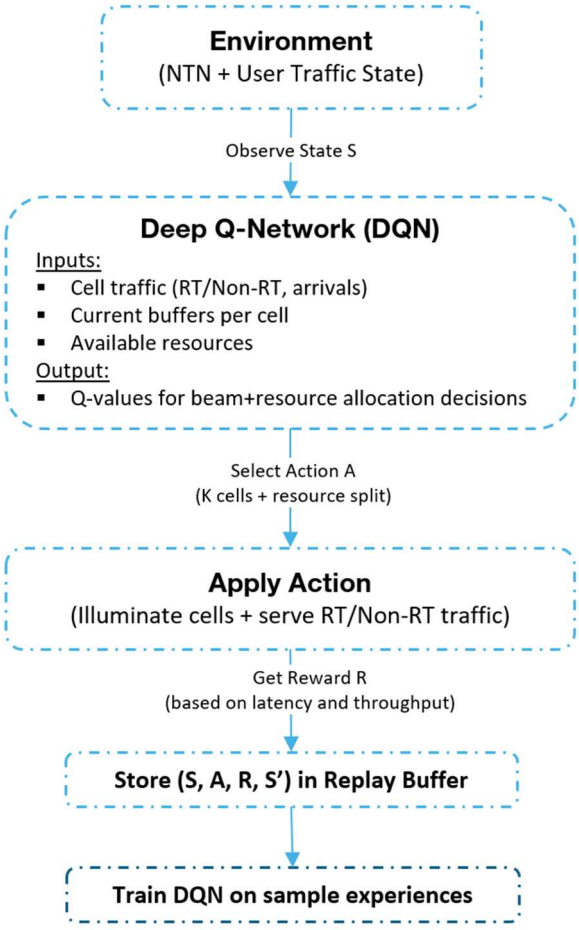


Figure 2-47: Flow chart of beam hopping optimization as a DQL algorithm

This subsection focuses on the definition of the AI/ML approach and its integration within the O-RAN control loop; a quantitative evaluation is left for future work, as it requires end-to-end implementation and curated datasets consistent with the assumed measurement pipeline.

2.4.2.2 Required Data

This subsection summarizes the key data required for training and inference of the beam hopping model and indicates the entities from which these data are collected. In particular, the model relies on measurements from network nodes and UEs, including:

- *DL data volume in the gNB-CU-UP buffer:* size of the downlink queue to be served at gNB-CU-UP
    - Example source: in real-time, **gNB internal counters** or **protocol-specific statistics** (e.g., via the gNB's management interface) can provide this information.
- *Frequency resource (available PRB for use over PDCCH from gNB-CU-CP to UE)*: scheduling-related indicators used to determine resource availability.
    - Example source: from Downlink Control Information (DCI) messages carrying:
        - Information to schedule (allocate physical resources) for Downlink Data (PDSCH),
        - Information to schedule (allocate physical resources) for Uplink Data (PUSCH),
        - Information to adjust Uplink Power (PUSCH, PUCCH power) for power control.
- *SINR per user and per cell (reported via RRC signaling at gNB-CU-CP)*
    - **CSI-SINR:** It is CSI Reference Signal – SINR measurement and similar to **SS-SINR** it can also be used for connected mode mobility procedures. Here wanted signal power and the interference plus noise power are measured from REs used by the CSI Reference Signals. UE measures CSI Reference Signal transmissions on **antenna port 3000** when providing **CSI-SINR** measurement,
    - **SS-SINR:** It is Synchronization Signal – SINR measurement and can be used for **connected mode** mobility procedures (Handovers). Both the wanted signal power and the interference plus noise power are measured from REs used by the Secondary Synchronization Signal **(SSS).** UE measures PBCH Demodulation Reference Signal (DMRS) when generating SS-SINR results,
        - In 5G networks is SINR reported as a coded value via measurement report to gNB (RRC signaling control plan data collection),
        - The reported range is from 0 to 127, total 128 values where reported value 0 is equal to SINR <-23 dB and value 127 represents SINR > 40dB

This dataset enables the model to correlate traffic pressure and radio conditions with beam illumination and resource allocation choices.

2.4.2.3 E2E Solution description

Table 2-1 summarizes the beam hopping optimization use case and its realization within the O-RAN framework.

Table 2-1: Beam Hopping Optimization

Use Case stage	Evolution/Specification
Goal	To automatically optimize the resource allocation across the beams by leveraging the AI/ML model that was trained via the analysis of performance measurements and the arrival traffic collected from the RAN nodes.
Actors and Roles	**Non-RT RIC:** trains AI/ML models offline using historical measurements (e.g., SINR, traffic demand, resource usage) to predict when and where resources should be added or reduced. **Near-RT RIC:** then applies these models in real time to allocate optimal resources via the E2 interface and update cloud resources through the O2 interface.
Assumptions	• All relevant functions and components are instantiated. • Non-RT RIC is able to receive performance measurements from RAN nodes via the O1 interface. • Near-RT RIC is able to receive performance measurements from RAN nodes via the E2 interface • Near-RT RIC has a low-latency control logic, to illuminate the beams at the appropriate time
Pre-conditions	Dataset based on historical measurements for model training
Begins when	An AI/ML model has been trained based on the historical measurements and traffic, and provided a model for dynamic resource allocation to the different beams
Step 1	Near-RT RIC collects RAN performance measurements and traffic demand from RAN nodes.
Step 2	The Near-RT RIC leverages the trained model to make real-time decisions on which beams to activate, how many resource blocks to allocate to each, and for what duration.
Step 3	RAN nodes execute this decision, and update cloud resources through the O2 interface
Step 4	Retrain the model with newly collected datasets to improve its accuracy and precision.
Ends when	**All the above steps are successfully completed**
Post conditions	One of the entities to continue the monitoring tasks.

The AI-driven beam hopping solution is a critical enabler for managing satellite resources to meet the demands of high-capacity use cases. In scenarios like Airway and Residential Broadband, user demand for eMBB services is highly concentrated and dynamic. A static beam illumination plan would waste significant power and spectrum on empty areas (e.g., oceans or unpopulated regions). Our DQL-based solution allows the NTN to intelligently focus satellite resources on active aircraft or populated residential areas, maximizing throughput and spectral efficiency. For the PPDR scenario, the algorithm can be guided to prioritize beams covering a disaster zone, ensuring that first responders have guaranteed and resilient connectivity.

2.4.3 AI Based Bearer Selection

NTN handovers between satellites require the UE to decide locally under tight timing and incomplete information. The delivery window for a network command is often too short, and the control plane can be transiently unreachable. As such, the UE must act and select the next satellite with a partial view of the system and still preserve service continuity.

What the UE is expected to know (observable inputs)

- A shortlist of currently visible candidates, the top-K beams or satellites derived from instantaneous visibility checks and geometry such as elevation and slant range, or from a local visibility table.
- Lightweight broadcast hints per candidate, such as load or utilization indicators and basic quality flags, recognized as noisy and non-authoritative – we assume that the satellites will broadcast not only their identity but also a load indication as a means to inform the UEs on their potential capabilities to fulfill the required QoS characteristics
- Recent local measurements of service quality for the selected bearer, including throughput, delay, and loss or jitter proxies, together with a small context history such as time since last handover, coverage presence, and simple motion or elevation cues – information which is locally gathered by the UEs.
- The UE may have only partial or unreliable access to future pass schedules, which limits its ability to plan handovers far ahead.
- It has no complete visibility of the core network state, such as real-time load distribution or mobility management context.
- It has no assurance that a network-triggered handover command will arrive on time, since signaling can be delayed or lost under backhaul congestion or feeder link outages.

What decision the UE must take (control output)

At each decision time ttt, the UE selects exactly one bearer a_t from the currently valid set B_t of candidates. The dual objective is:

- Maximize expected service utility over time, favouring stable throughput and low delay or loss.
- Minimize unnecessary switching, because each handover incurs interruption, buffer disruption, signaling, and energy cost.

Why the handover decision must be local and fast

- Non-stationary environment: LEO geometry, feeder weather, and periodic passes change conditions at minute scale; centralized advice can arrive late or not at all.
- Uncertain control plane: Even if the network recommends a handover, delivery can fail due to backhaul congestion, feeder loss, or RRC timing, so a robust fallback policy is required on the UE.
- Application sensitivity: Low-latency and real-time services cannot tolerate dithering or late commands; the UE must anticipate the next viable bearer and switch at the right moment rather than the first possible moment to assure a seamless service delivery.

Constraints and trade-offs

- Switching penalty: Each handover has a tangible cost and flapping must be avoided – one of the key issues in satellite handovers due to the many fleeing options.

- Partial observability: Only a limited shortlist and noisy hints are available while many network states remain hidden – complete information sending to the UE is way too much not to mention the long delay of gathering this information.
- Computational budget: The policy must fit UE or relay hardware, operate in millisecond level and remain robust to missing data.
- Safety: Candidates that fail hard sanity checks, such as lack of coverage, must be excluded immediately.

What a handover algorithm must deliver

- A learned on-device policy that ingests recent QoS and context together with the current candidate set to select a bearer likely to remain good for longer, maximizing dwell before the next handover while avoiding obviously overloaded choices.
- Hysteresis that suppresses micro-advantages which would otherwise trigger needless switches.
- Anticipation: recognition of periodic patterns to time the switch in a way that minimizes outage and switching cost instead of reactive behaviour.

Why we need learning rather than simple rules

- Rules cannot anticipate non-stationary patterns robustly when hints are noisy, and the candidate set size varies over time.
- The optimal trade-off between stability and quality is context-dependent and evolves with conditions.
- A reinforcement-learning approach can encode experience, infer temporal structure from short histories, and adapt online to local conditions, yielding better handover timing and fewer detrimental switches than static thresholds.

2.4.3.1 Contextual Bandit Reinforced Learning (CBRL) UE handover algorithm

State, actions, shortlist

Decision time: $t \in \{0, \dots, T-1\}$. The environment provides the state vector:

$$s_t = \left[t/T \, , c_t \, , d_t \, , \ell_{min,t} \, , \ell_{best,t} \, , \sigma_t \right]$$

where:

- $t/T \in [0,1]$ is normalized time within the episode
- $c_t \in \{0,1\}$ indicates coverage availability at time t
- $d_t \in [0,1]$ encodes time-in-current-best normalization (proxy for dwell stability)
- $\ell_{min,t} \in [0,1]$ is the minimum broadcast load hint among the current shortlist
- $\ell_{best,t} \in [0,1]$ is the load hint of the best-by-range satellite if present in the shortlist (0 otherwise)

- $\sigma_t \in \{0,1\}$ indicates whether a switch occurred at $t-1$ (1 if the chosen real ID changed, else 0). Shortlist (dynamic action set): from the visibility engine, visible satellites of constellation 0 are sorted by slant range; we keep:

$$V_t = \{(i, r_i, \ell_i)\} \quad (i = 1, \dots, K_t), \quad K_t = min\left(|visible|, \underline{K}\right)$$

where $\underline{K} = 8$ (default), r_i is the slant range (km) of candidate i, and $\ell_i \in [0,1]$ is its broadcast load hint.

Action: $a_t \in \{0, \dots, K_t - 1\}$ *selects an index in* V_t. *A helper mapping returns the real satellite ID* $I_{real,t}(a_t) \in \{0, \dots, 431\}$. *When* $K_t = 0$, *a null step is taken with* $r_t = 0$ *and time advances.*

Reward

Let b_t be the real ID of the best visible satellite at time t (minimum slant range among constellation 0), and R_t^{best} its slant range (km). Let R_{95} be the 95th percentile of $\{R_t^{best}\}$ over the episode (finite values).

If $I_{real,t}(a_t) = b_t$

$$r_t = \alpha \cdot \min(1, dwell_t/10) + (1-\alpha) \cdot \max\left(0, 1 - R_t^{best}/R_{95}\right) + \beta \cdot \min(1, future\_dwell_t/S)$$

If $c_t = 1$ and $I_{real,t}(a_t) \neq b_t$

$$r_t = -\lambda$$

Else if $c_t = 0$:

$$r_t = 0$$

Switching penalty (applies whenever the chosen real ID differs from the previous step's chosen real ID):

$$r_t \leftarrow r_t - \kappa$$

Load penalty (applies when an action is taken, and a load hint exists for the chosen candidate):

$$r_t \leftarrow r_t - \mu \cdot \ell_{chosen,t}$$

Here $dwell_t$ is the current consecutive run length of the best satellite up to t, $future\_dwell_t$ is the run length starting at t, and $\ell_{chose\ ,t}$ is the chosen candidate's load hint. Parameters β and S control the future-dwell bonus magnitude and saturation.

Default parameters:

$$\alpha = 0.5; \lambda = 0.05; \kappa = switch\_cost = 0.02; \mu = load\_weight = 0.15; \beta = dwell\_bonus = 0.20; S = dwell\_scale = 6.0$$

Learning rule

Tabular Q-learning with discretized state bins; dynamic masking restricts action evaluation to the valid prefix $\{0, \dots, K_{t-1}\}$.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \cdot \left[r_t + \gamma \cdot \max_{a' < K_{\{t+1\}}} Q(s_t + 1, a') - Q(s_t, a_t) \right]$$

Action selection is ε-greedy over $\{0, \dots, K_{t-1}\}$. When $K_t = 0$, the agent takes a null step (no update to Q other than advancing state).

Parameters (defaults from implementation)

Symbol	Meaning	Default
K	Maximum shortlist size (Top-K)	8
α	Blend between dwell-term and range-term	0.5
λ	Penalty for wrong choice under coverage	0.05
κ	Switching cost (handover penalty)	0.02
μ	Load penalty weight	0.15
β	Future-dwell bonus weight	0.20
S	Future-dwell saturation scale (minutes)	6.0
γ	Discount factor	0.95 (per code)
η	Learning rate	0.1 (per code)
ε	Exploration probability (decaying)	start 1.0 → min 0.05

3) Pseudo-code of the algorithm

```
Inputs: orbit files, visibility engine, K (default 8), T (episode length),
        α=0.5, λ=0.05, κ=0.02, μ=0.15, β=0.20, S=6.0, γ=0.95, η=0.1, ε schedule

Precompute (for t = 0..T−1):
  visibles_t      = sorted visible satellites by slant range (constellation 0)
  best_t          = visibles_t[0].id if visibles_t ≠ ∅ else None
  Rbest_t         = visibles_t[0].range_km or NaN
  shortlist_idst = [id of first min(K, len(visibles_t)) entries]
  shortlist_rngst= [range_km of those entries]
  shortlist_loadst = [load_hint(id, t) for those entries]
R95 = 95th percentile of {Rbest_t} over finite values
future_dwellt and dwell_t are precomputed per step (current and future run lengths
of best_t).

Episode loop:
  Initialize previous_chosenreal = None; σ_0 = 0
  Observe s = [t/T, c_t, d_t, ℓ_min,t, ℓ_best,t, σ_t] at t = 0
  For t = 0..T−1:
    Kt = len(shortlist_idst)
    if Kt == 0:
        r = 0; previous_chosenreal stays; t ← t+1; continue
    With prob ε choose a ∈ {0..Kt−1} uniformly
    else a = argmax_{0..Kt−1} Q[s, ·]
    chosen_real = shortlist_idst[a]
```

```
    correct = (chosen_real == best_t)
    if c_t == 1 and correct:
        dwell_term = min(1, dwell_t / 10)
        range_term = max(0, 1 – Rbest_t / R95)
        r = α*dwell_term + (1–α)*range_term + β*min(1, future_dwellt / S)
    elif c_t == 1 and not correct:
        r = –λ
    else:
        r = 0
    # switching penalty
    if previous_chosenreal is not None and chosen_real != previous_chosenreal:
        r ← r – κ; σ_{t+1} = 1
    else:
        σ_{t+1} = 0
    # load penalty
    ℓ_chosen,t = shortlist_loadst[a]
    r ← r – μ * ℓ_chosen,t

    Observe s' = [ (t+1)/T, c_{t+1}, d_{t+1}, ℓ_min,t+1, ℓ_best,t+1, σ_{t+1} ], compute
K_{t+1}
    Q[s,a] ← Q[s,a] + η * [ r + γ * max_{a' < K_{t+1}} Q[s',a'] – Q[s,a] ]
    previous_chosenreal = chosen_real
    s ← s'

Return: learned Q; policy π(s) = argmax_{a < K(s)} Q[s,a]
```

2.4.3.2 Simulation Implementation

The implementation is built on top of a reference orbit and visibility engine that computes realistic satellite positions and their per-minute visibility from a fixed ground site. The constellation used in this study corresponds to a representative low Earth orbit (LEO) design, with altitude of 600 km, inclination of 65°, and 12 orbital planes with 36 satellites per plane, yielding 432 satellites in total. Ephemerides for this constellation are generated once at the beginning of the simulation and provide the deterministic satellite ground tracks used throughout the training process. For the ground site, “clear skies” are assumed, meaning that no atmospheric blockage or terrain hindrance is modeled, and visibility is determined solely by geometry.

At each simulation step, corresponding to one minute, the visibility engine determines which satellites of constellation 0 are above the local elevation mask. The set of visible satellites is sorted by slant range, and the environment only exposes to the agent the shortlist of the K=8 closest candidates. Each candidate in this shortlist is associated with its geometric range and a broadcast load hint. The load hints are not obtained from actual traffic, but are simulated as smooth, satellite-specific signals. Each satellite ID is assigned unique phases, and the load is generated as a weighted sum of two sinusoids with periods of roughly 120 and 29 minutes, plus a small ID-dependent jitter. A constant baseline is added, and the signal is clipped to the interval [0,1]. This yields deterministic but time-varying values that mimic broadcast utilization indicators: they are noisy, smoothly varying, and distinct per satellite. The intent is to force the UE policy to learn to balance geometry with load avoidance under partial observability.

The environment state at time ttt is constructed from six compact features: normalized episode time, coverage indicator, normalized dwell duration in the current best satellite, the minimum load across the shortlist, the load of the best-by-range satellite, and a binary indicator of

whether the previous action incurred a switch. The action at selects one entry from the current shortlist, which is mapped back to its unique satellite identifier. The reward integrates multiple components: a base term for correct selection, a penalty for wrong choices, an explicit switching cost, a penalty proportional to the load of the chosen candidate, and a future-dwell bonus that rewards choices expected to remain optimal for longer.

Learning is realized through a tabular Q-learner with discretized bins over the six-dimensional state. The update rule follows standard Q-learning with discount factor $\gamma = 0.95$ and learning rate $\eta = 0.1$, while exploration follows an ε-greedy schedule decaying from 1.0 to a floor of 0.05. The agent therefore gradually transitions from random exploration to stable exploitation of its learned policy.

2.4.3.3 Evaluation

As evaluation, the default training configuration was used, consisting of 300 episodes, each of length 360 minutes, with a cone angle of 70° and Berlin as the ground site. The analysis presented here focuses on the final training episode, in which the learned policy is applied without further exploration. Panels (a–f) in Figure 2-48 highlight complementary aspects of decision-making, correctness, reward composition, geometry, dwell stability, and sensitivity to broadcast load hints.

(a) Chosen versus best geometrical satellite. The first panel shows the identifier of the chosen satellite compared with the best-by-range candidate in constellation 0. Extended periods of overlap indicate that the policy often selects the geometrically optimal satellite. However, multiple divergences are observed, particularly around handover boundaries where the best satellite changes frequently. These divergences are not arbitrary: the policy occasionally latches onto a non-best candidate and maintains it across several minutes. This reflects the joint influence of the switching penalty and load penalty, which together encourage stability even at the expense of instantaneous optimality.

(b) Correctness and coverage. The second panel provides a binary view of coverage and correctness. Coverage is present almost continuously, as expected under clear-sky assumptions and dense LEO deployment. Correctness, however, exhibits sharp drops to zero in localized intervals. These correspond to moments when the chosen satellite differs from the best-by-range despite coverage. The insight is that the policy tolerates controlled suboptimality to suppress flapping. Instead of chasing every geometric transition, the agent applies hysteresis implicitly through its reward function, which yields more stable bearer attachment.

(c) Reward per step. The third panel illustrates the instantaneous reward. Values remain clustered in the range 0.05–0.25, demonstrating that the policy consistently earns positive reinforcement. Outliers with negative values occur sparsely, aligned with coverage errors or unnecessary switches. Importantly, the reward trace lacks long negative streaks, underlining that the agent does not fall into pathological regimes but maintains robust decision-making.

(d) Best slant range. The fourth panel depicts the slant range of the best satellite. A regular oscillatory structure is visible, caused by the periodic overpasses of satellites across Berlin. No unexpected spikes or gaps are present, confirming that the orbital geometry and visibility model operate as intended. The amplitude and periodicity illustrate that the agent must operate in a non-stationary environment where the “best” candidate evolves on minute-scale dynamics.

(e) Dwell span. The fifth panel reports that the current best remains optimal. The majority of values fall between one and three minutes, reflecting the high density of the constellation and the rapid change in elevation ordering among nearby satellites. This has a direct consequence: frequent handovers are inevitable, and the agent’s task is not to eliminate them but to time

them correctly. The limited dwell horizon also underlines the importance of the future-dwell bonus; without it, the policy would chase micro-transitions and degrade stability.

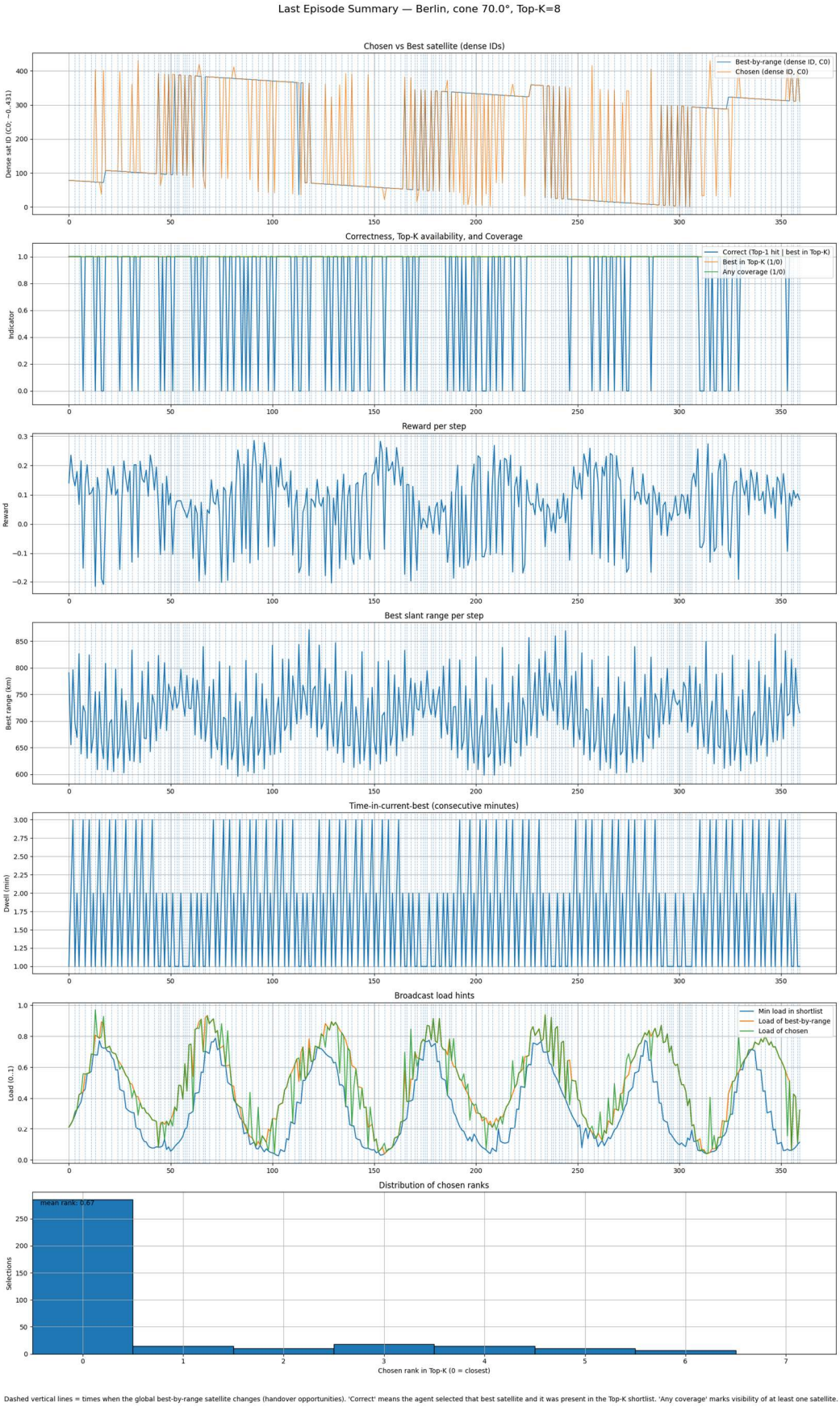


Figure 2-48: Contextual Bandit Reinforced Learning UE Handover Algorithm Results

(f) Broadcast load hints. The sixth panel compares three signals: the minimum load in the shortlist, the load of the best-by-range satellite, and the load of the chosen satellite. All three follow smooth sinusoidal dynamics, as expected from the synthetic model. Critical insight is provided by intervals where the chosen load deviates from the best-by-range load: these are moments where the policy deliberately avoids an overloaded but geometrically optimal satellite. This confirms that the agent has internalized the noisy broadcast hints and uses them as a discriminant. The system therefore goes beyond geometry-based handovers, incorporating a form of implicit load balancing at the terminal.

(g) Distribution of chosen ranks (Top-K). This panel shows a histogram of the shortlist rank that the policy selected at each step of the last episode (rank 0 = closest satellite, rank 1 = second-closest, …). The mass is concentrated at rank 0 with a light tail up to K–1 and a mean rank around 0.6–0.7. We see this shape because the reward function encourages picking the current best-by-range, but also penalizes switching and high load; around handover boundaries or when the closest satellite is heavily loaded, the agent occasionally selects a non-closest candidate, pushing the rank upward for those steps. The implication is that deviations from rank 0 are purposeful trade-offs, not random mistakes. In practice, this confirms the intended behavior: geometry dominates most decisions, while the policy departs from the closest option sparingly to gain stability or avoid congestion. Increasing switch_cost or load_weight would grow the right-hand tail; reducing them (or the dwell incentive) would shrink it.

We introduce this figure ahead of the finer per-step diagnostics because it establishes the basic preconditions for any downstream interpretation: (i) that the agent learns and converges rather than oscillating; (ii) that the shortlist is adequate so “misses” are not merely availability effects; (iii) that we can decouple availability from decision quality via a conditional accuracy metric; and (iv) that we understand the training budget (episodes to stabilization) that frames all subsequent results.

In Figure 2-49, the first panel shows the total reward per episode rising from negative to positive values and then flattening, indicating convergence to a stable trade-off among the four forces encoded in the reward—range, switching, dwell, and load. The second panel aggregates policy quality and coverage: with K=8 in this geometry, Top-K coverage remains essentially 1.0 throughout, while the conditional Top-1 hit—the probability of selecting the global best when it appears in the shortlist—improves from ~0.2 to ~0.7–0.8 over training. This pattern arises because action-masked tabular Q-learning receives a clean signal: the distance and future-dwell components reward picking and staying with strong candidates, whereas switching and load penalties suppress flapping and overloaded links without swamping the update. The residual gap to 1.0 is therefore not a detection failure; it reflects intentional policy choices to avoid costly handovers or heavily loaded satellites when the trade-off warrants it.

Taken together, the curves confirm that the reward shaping is coherent, that K=8 is sufficient for this site and cone angle (coverage is not the bottleneck), and that the agent stabilizes within ~60–120 episodes (typically a bit earlier for reward than for conditional hit). Where faster convergence is desired, modest adjustments to exploration decay or learning rate are appropriate; where unconditional accuracy lags at other sites, verify coverage first (e.g., by increasing K) before returning the reward weights, since availability—not ranking—may be the limiting factor.

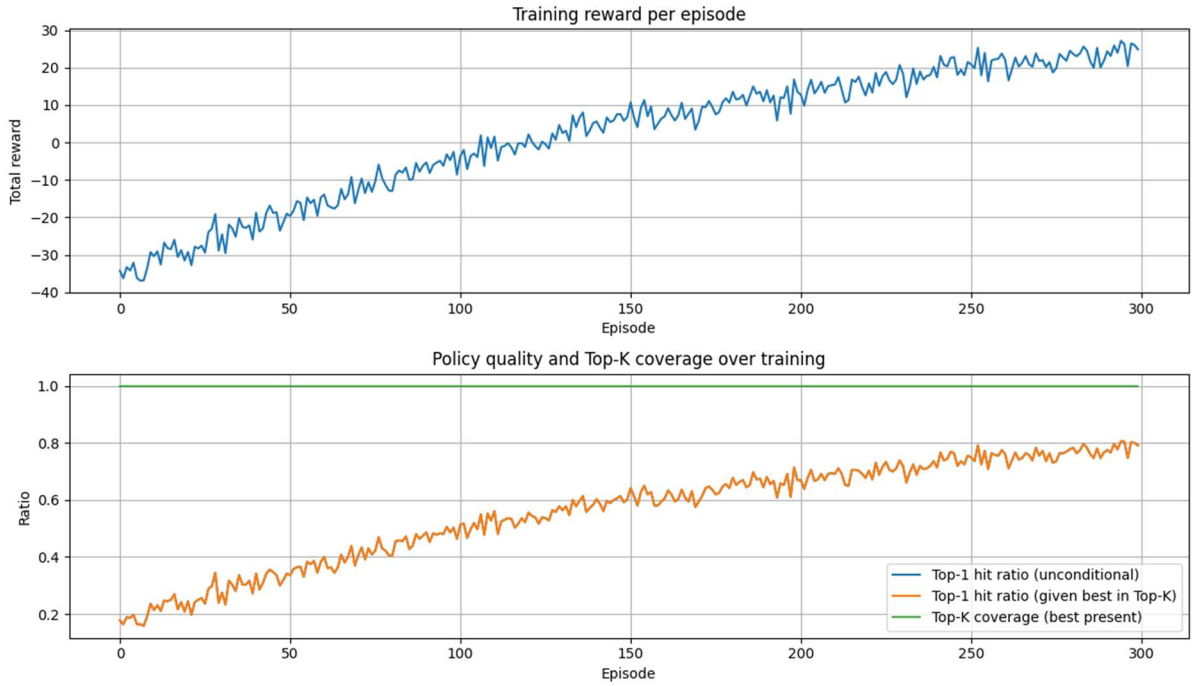


Figure 2-49: Learning across the different episodes

Figure 2-50 presents four coordinated views that isolate congestion effects while keeping handover and stability incentives constant. In the top-left panel (reward vs. episode), three learning curves, corresponding to low, mid, and high load_weight, rise from their initial transients and progressively flatten. The low-load_weight curve settles highest, the mid curve slightly lower, and the high curve lowest, reflecting the fact that, under heavier congestion penalties, the same sequence of actions accrues less reward even when decisions are correct. The early episodes exhibit modest variance and occasional dips; these diminish as exploration decays and the policy converges. The top-right panel (conditional Top-1 hit vs. episode) shows all three curves climbing steadily toward ~0.7–0.8, with only small separation between them; decision quality—conditioned on the global best being present in the shortlist—remains robust across congestion levels. The initial phase (low accuracy) transitions into a plateau where the curves oscillate narrowly around their asymptotes, indicating that residual errors are not due to uncertainty but to deliberate trade-offs encoded by the cost terms.

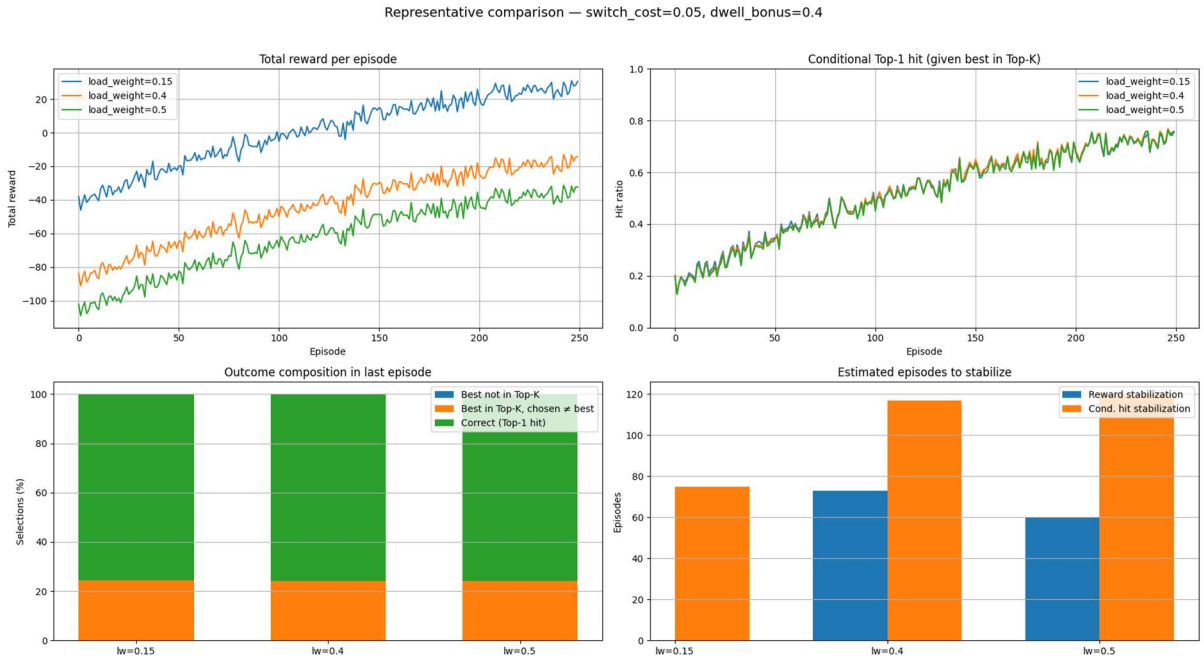


Figure 2-50: How load impacts selection evaluation

The bottom-left panel (last-episode outcome composition) decomposes terminal behavior into three mutually exclusive categories: Correct (best in Top-K and chosen), Best in Top-K, chosen ≠ best (intentional deviation), and Best not in Top-K (coverage limitation). The Best not in Top-K segment is negligible across the three loads—consistent with Top-K coverage ≈ 1—so remaining misses are concentrated in the Best in Top-K, chosen ≠ best slice. That slice grows modestly as load_weight increases, indicating that, in heavier congestion, the policy more often selects a less-loaded alternative even when it is not the closest or the absolute best by range. The bottom-right panel (episodes to stabilize) summarizes the estimated convergence horizon for both reward and conditional hit. Reward typically stabilizes earlier; the conditional metric stabilizes within ~75–120 episodes, with the high-load_weight bar shifted modestly to the right relative to low and mid loads, consistent with a tighter, more conflicting objective surface under stronger load penalties.

Read together, the panels make the visual story explicit. What is seen: ascending reward curves with load-ordered asymptotes, accuracy curves that converge to a narrow band near ~0.7–0.8, a last-episode composition dominated by Correct selections and intentional deviations rather than coverage failures, and stabilization bars clustering around ~100 episodes with a modest rightward shift at high congestion. Why it appears this way: increasing load_weight raises the penalty for choosing busy satellites, lowering attainable reward and incentivizing occasional departures from the range-optimal pick, while leaving the identification of the best candidate essentially intact when it is available. What it means: congestion primarily affects utility and convergence speed, not decision quality per se; misses are policy choices, not shortlist gaps. Conclusions: when congestion avoidance is a priority, increasing load_weight is the appropriate level; a slightly longer training horizon and a lower steady-state reward should be expected, while accuracy remains high given the strong coverage with K=8.

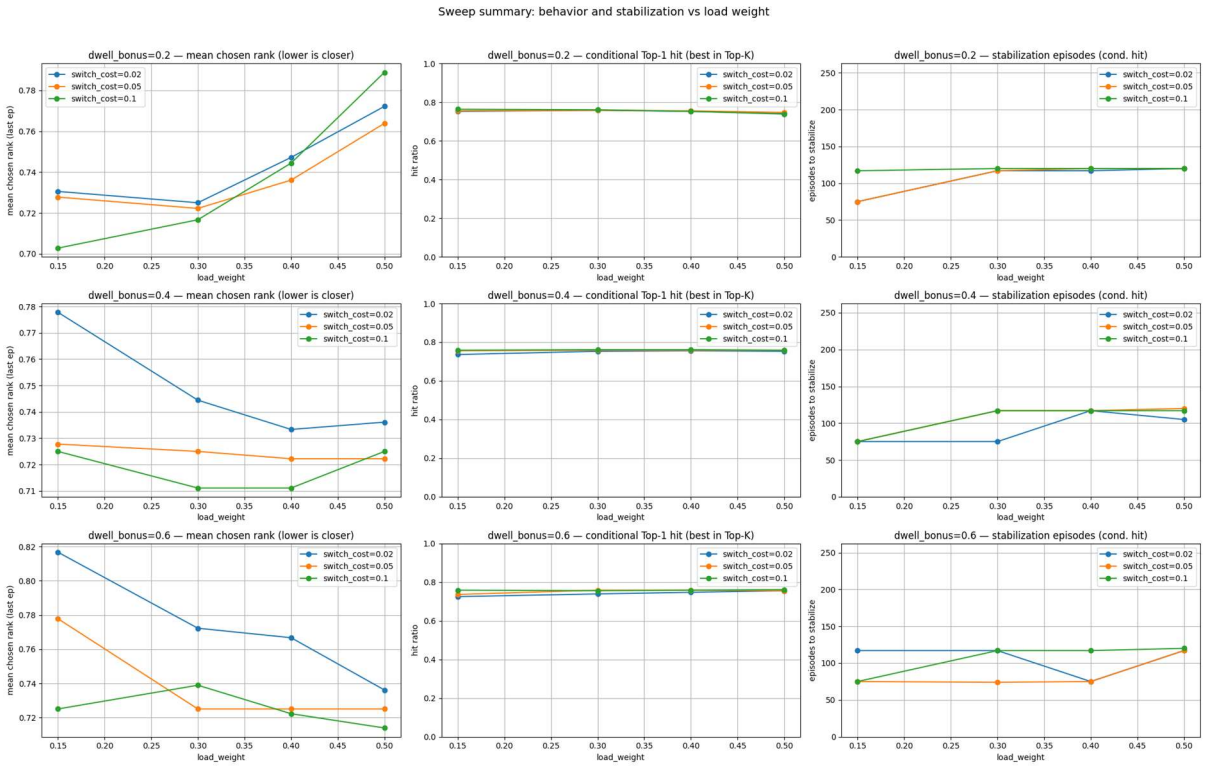


Figure 2-51: Sweep summary: behavior and stabilization vs. load_weight across switch_cost and dwell_bonus

Figure 2-51 organizes the parameter space into rows by dwell_bonus ∈ {0.2, 0.4, 0.6} and, within each row, shows curves over load_weight ∈ {0.15, 0.30, 0.40, 0.50} with line styles/colors indexing switch_cost ∈ {0.02, 0.05, 0.10}. The left column (mean chosen rank vs. load_weight) exhibits a clear ordering: curves corresponding to higher switch_cost sit consistently above those with lower switch_cost, and slopes are steeper when dwell_bonus is small. Mean rank increases with load_weight, most markedly in the low–dwell_bonus regime, reflecting a greater propensity to select non-closest shortlist items as congestion pressure grows. As dwell_bonus rises to 0.4–0.6, slopes flatten and, in places, nearly plateau: stability is already incentivized by the bonus, so additional congestion pressure produces less marginal shift in rank selection. The middle column (conditional Top-1 hit vs. load_weight) remains high and comparatively flat across the grid, with only mild degradation at the most punitive combinations; once the global best is present in the shortlist, identification and selection remain reliable despite stronger penalties elsewhere in the reward. The right column (episodes to stabilize for the conditional metric) shows compact bars that cluster around the ∼100-episode mark in most settings, with a discernible rightward drift where load_weight and switch_cost are simultaneously high; the increased horizon reflects a more conflicting objective surface in which the policy must reconcile congestion avoidance with handover aversion.

The pattern appears this way because the three knobs load the objective along distinct axes. load_weight magnifies the penalty for busy satellites, encouraging occasional departures from the range-optimal choice and thereby lifting mean rank with increasing values. switch_cost punishes changes in attachment, promoting persistence; higher levels raise mean rank at all loads by making “stay” decisions relatively cheaper than “chase the closest.” dwell_bonus rewards foresight—seeking options that remain good—so moderate to high values partially substitute for explicit switching penalties and damp the incremental effect of added congestion pressure on rank. Conditional Top-1 hit remains stable because these forces reshape preferences among available candidates rather than undermining the ability to recognize the

best candidate when it appears in the shortlist. Stabilization slows when penalties jointly sharpen trade-offs (high load_weight + high switch_cost), as the Q-surface becomes flatter along some directions and more rugged along others.

The meaning is straightforward. Mean rank serves as a behavioral readout (how often the policy accepts a non-closest choice); conditional hit serves as a decision-quality readout (how well the policy acts when the best is available); stabilization episodes serve as an engineering readout (how long to allocate for training). The sweep demonstrates that behavior can be steered predictably along these three axes without sacrificing conditional accuracy: raising load_weight increases congestion sensitivity (higher mean rank, lower attainable reward), raising switch_cost increases handover aversion (higher mean rank, fewer flaps), and raising dwell_bonus delivers stability with softer penalties (flatter rank–load curves). Conditional accuracy remains high because coverage is strong and the recognition task is unchanged.

The conclusions follow directly. A practical operating region emerges with switch_cost around 0.05 and dwell_bonus in 0.4–0.6 to achieve smooth, stable behavior, and load_weight selected in 0.3–0.5 to match anticipated congestion. Planning for approximately ~100 episodes is appropriate for stabilization of the conditional metric, with additional budget advisable in the high-penalty, high-congestion corner. If conditional accuracy degrades in other geometries, shortlist sufficiency should be examined first (increase K to restore coverage) before retuning reward weights; if excessive stickiness is observed, reduce switch_cost or dwell_bonus, or moderate load_weight to relax congestion pressure.

Taken together, the 4 figures show that the reinforcement-learning policy has internalized the intended trade-offs. Across the last training episode, the policy attains an average correctness of approximately 70–80% and a handover hit rate above 80%, indicating a consistent mapping from local state to effective actions. Figure Z establishes the methodological baseline: convergence of the composite reward, near-perfect Top-K coverage for this geometry (K=8), and conditional Top-1 accuracy rising to ~0.7–0.8, which confirms that, given availability, decision quality is high. Figure Y demonstrates that this accuracy is robust to increased congestion penalties: heavier load_weight depresses attainable reward and slightly delays stabilization but leaves the ability to identify the best candidate essentially intact. Figure X then maps the broader design space, showing predictable shifts in behavior (mean chosen rank), stabilization horizon, and utility as load_weight, switch_cost, and dwell_bonus are varied, without sacrificing conditional accuracy.

The deviations from geometric optimality observed in panels (a)–(b) are deliberate responses to load and handover cost. Panel (f) and Figure 2-50's outcome decomposition highlight intervals where the best-by-range satellite is intentionally avoided when heavily loaded, with the chosen load tracking the shortlist minimum rather than the geometric best. Panel (g) corroborates this with a rank distribution concentrated at 0 and a light tail to higher ranks (mean rank ≈0.6–0.7): geometry dominates, but the policy departs from the closest option sparingly to gain stability or avoid congestion. Figure and Figure 2-50 jointly indicate a stabilization horizon of roughly ~75–120 episodes for the conditional metric (typically earlier for reward), establishing a practical training budget for comparable scenarios.

The sweep in Figure 2-51 provides actionable tuning guidance. load_weight governs congestion sensitivity: higher values promote avoidance of busy satellites (higher mean rank, lower steady-state reward, modestly slower stabilization). switch_cost governs handover aversion/persistence: higher values increase stickiness (higher mean rank, fewer flaps), particularly near handover boundaries. dwell_bonus provides stability without hard penalties: moderate–high values (≈0.4–0.6) flatten the rank–load curves by rewarding foresight, reducing the need for large switching penalties. A practical operating region emerges with switch_cost around 0.05, dwell_bonus in 0.4–0.6, and load_weight selected in 0.3–0.5 to match anticipated

congestion; planning for ~100 episodes to reach steady performance is appropriate, with additional budget advisable in the high-penalty, high-congestion corner.

At the same time, the evaluation highlights structural limitations that bound external validity. Average dwell durations of one to three minutes make frequent handovers inevitable in dense LEO; hysteresis can occasionally prolong attachment to suboptimal candidates. The congestion signal is a synthetic, deterministic load-hint process; real networks exhibit burstiness, traffic coupling, and measurement noise. The geometry is anchored to a single site (Berlin) and cone angle (70°); different latitudes, masking, or weather would alter visibility and coverage. The objective optimizes proxies (range, switching, dwell, load), not end-to-end QoS: SINR, beam gain, interference, Doppler, feeder availability, and ISL congestion are not yet explicit. The analysis is single-UE and does not address fairness or contention between UEs. These caveats suggest concrete next steps: enrich propagation and interference models; replace load hints with measured or high-fidelity simulated utilization; expose additional state (e.g., time since last switch, normalized distances of Top-K, prior attachment ID); move from tabular Q-learning to a compact function approximator (e.g., DQN or actor–critic with action masking) for cross-site generalization; incorporate route-aware terms (e.g., path to gateway, latency to core); and extend to multi-UE scenarios to surface fairness and scheduling effects. Under these extensions, the methodology evidenced by learning/stability, congestion sensitivity, and global tuning remains applicable and provides a clear path from proof-of-concept to operational NTN handover strategies.

2.4.3.4 Conclusions and Further Work

This section has introduced a basic reinforcement-learning simulation for NTN bearer selection. The intent was not to produce a deployable prototype but to explore whether a simple on-device policy can internalize stability–quality trade-offs when operating under partial information. The simulation shows that even a tabular Q-learner, supplied with limited visibility and noisy broadcast hints, can achieve correctness in the 70–80% range and handover hit rates above 80% across a dense 600 km constellation. Importantly, moments where the hit rate drops or the chosen satellite is not the best by geometry do not necessarily mean degraded service. In these cases, the UE deliberately selects an alternative candidate with lower load, thereby ensuring that sufficient resources are available to sustain user experience. Suboptimality in geometric terms is thus a reflection of QoS-aware behavior, not a failure of the method, and continuity of service is maintained throughout the episode.

At the same time, this remains a stripped-down environment with clear-sky assumptions, deterministic load, and a single ground site. Important aspects such as fading, blockage, feeder outages, or interference are not yet represented. A central direction for further work is to integrate line-of-sight impediments into the simulation, so that the policy learns to identify and exclude satellites that will soon lose visibility due to terrain, buildings, or low elevation angles. This would allow the UE to anticipate shortened dwell times and filter out candidates that are not realistically usable. Beyond this, future extensions should replace the tabular learner with lightweight function approximators capable of generalizing across positions and time, enrich the load model with burstiness and multi-user contention, and include route-aware metrics such as gateway distance or latency. Evaluations across multiple sites and mobility patterns will also be necessary to assess generalization.

It should be underlined that the approach described here is independent of the current 5G NTN specifications or to any radio link specificity. The idea of on-device learning from partial observability, balancing geometry with quality indicators, is equally relevant to the design of 6G NTN systems or for any other mega-constellation. In fact, 6G research is expected to move toward more autonomous and adaptive terminals, where lightweight reinforcement learning agents could become integral to mobility management as a means to compensate for the potential handover failures in case of fast LoS loss where handover commands may not arrive.

Thus, while the present results are limited to a basic simulation, they outline a conceptual path that remains applicable as NTN standards evolve.

2.5 ML POLICY IMPLEMENTATION

The Network Data Analytics Function (NWDAF) acts as the central intelligence hub of the system, collecting vast amounts of data from various network functions. As detailed in the provided content, the NWDAF gathers analytics on E2E data volume transfer time, PDU session traffic, slice load levels, and network performance. This wealth of information serves as the essential input for an intelligent, closed-loop network management system. To effectively leverage this data and translate it into actionable decisions, a robust ML policy is required. This subsection provides a comprehensive overview of the implementation, evaluation, and optimization of such policies. We will examine various feasible ML deployment options, including centralized, distributed, and hybrid strategies, and discuss the key evaluation criteria based on response time, resource requirements, and Quality of Assurance (QoA) towards different user demands. We will also outline the procedures for testing and validating these policies in real-world scenarios, define KPIs to measure their effectiveness, explore different optimization techniques such as reinforcement, supervised, and unsupervised learning, and analyze the scalability and flexibility required to adapt to varying network conditions [4].

An E2E network slicing approach necessitates multi-tier network orchestrator to handle the slices in each domain. The suggested approach is shown here in Figure 2-52. NWDAF plays a major role in aiding the E2E orchestration. We will discuss the role of each block in the following sections.

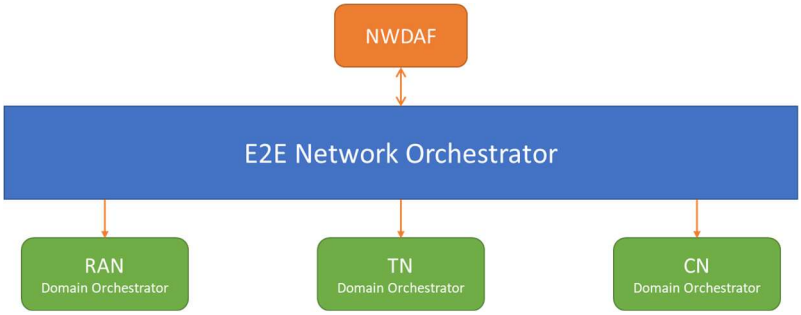


Figure 2-52: NWDAF integration in the E2E NS architecture

In integrated TN–NTN deployments, ML policy implementation must consider not only the control objective (e.g., slicing, resource allocation) but also the feasibility of executing each phase of the AI lifecycle across ground and space segments. In particular, satellite SWaP constraints and feeder-link capacity/availability directly affect whether telemetry can be streamed to ground (O1) or whether inference must be moved on-board and updated through compact model artifacts (A1). Therefore, the ML policy is extended to include lifecycle-aware placement decisions for training, model dissemination, and near-real-time inference in the O-RAN control hierarchy.

2.5.1 Network Data Analytics Function (NWDAF)

NWDAF collects data from the NFs and also provides analytics to NFs. An NWDAF may contain the following logical functions [1]:

- **Analytics logical function (AnLF)**: A logical function in NWDAF, which performs inference, derives analytics information (i.e. derives statistics and/or predictions based on Analytics Consumer request) and exposes analytics service.

- **Model Training logical function (MTLF)**: A logical function in NWDAF, which trains Machine Learning (ML) models and exposes new training services (e.g. providing trained ML Model).

NWDAF can contain an MTLF or an AnLF or both logical functions. Analytics information are either statistical information of past events, or predictive information. NWDAF contains information from all the NFs, but we concentrate here on the data that can be used for Network slicing.

- End-to-end data volume transfer time analytics

  The E2E data volume transfer time refers to a time delay for completing the transmission of a specific data volume from UE to AF, or from AF to UE. The data can be filtered at UE Level or at slice level. This can be used by the E2E orchestrator to monitor the slices. The fact that these analytic data can be requested for Geographic Distribution and specific time periods enables the orchestrator to make decisions at slice Level for different time periods. If the orchestrator does not have enough resources to do the prediction of various metrices, NWDAF can also provide data volume Transfer time predictions.

  The Inputs for calculating the metrices include RAN and Transport Information in both uplink and downlink direction.

- PDU Session traffic analytics

  PDU Session traffic Analytics gives the UEs that do not match the expected traffic characteristics. The details regarding the Network slice on which the UE belongs, data volume transferred and traffic descriptor with the traffic Destination are also included. These metrices can be used by the orchestrator to decide whether more resources have to be allocated and which domain orchestrator should act to satisfy the requirements.

- Slice load level related network data analytics

  The NWDAF provides slice load Level Information to a consumer NF at Network slice level. It is possible to share this information since NWDAF has slice level details such as the mean number of UEs registered by the slice, mean number of PDU sessions established, and the load of each NFs constituting the network slice. The predictions for load Level statistics can also be shared by the NWDAF.

- Observed Service Experience related network data analytics

  The NWDAF can provide Observed Service Experience analytics, in the form of statistics or predictions. The different Analytics include Service experience for a Network Slice, Service experience for an application, Service experience for an Edge Application. The analytics also contain a contribution weight value which indicates the importance of the Service experience. For example, the experience might not be important to a UE if the Service is not used frequently, or all the features of the service are not used.

This can be provided for a UE or a group of UEs or globally.

- NF load analytics

  The NF load analytics in the form of statistics or predictions, or both, can be provided by the NWDAF. The predictions are shared with a validity period. This parameter gives an idea of the load handled by the NFs in the Core Network. The current resources assigned to a NF in the Core Network can be re-assigned based on this info.

- Network Performance Analytics

  NWDAF provides either statistics or predictions on the gNB status information, gNB resource usage, communication performance and mobility performance in an Area of Interest. In addition, NWDAF can provide statistics or predictions on the number of UEs located in that Area of Interest.

2.5.2 Data Pre-processing and Feature Engineering

The raw data collected by the NWDAF, while comprehensive, is often not in a format directly usable by machine learning models. Here we detail the essential steps of data pre-processing and feature engineering, which transforms the raw analytics into a high-quality dataset. This process is crucial for the successful training and performance of any ML policy.

- **Data Cleaning and Imputation**: Real-world network data can be messy, with missing values, noise, and outliers caused by sensor errors or network disruptions. Before any analysis, this data must be cleaned. Missing values are handled using techniques like mean, median, or predictive imputation. Outliers, such as sudden and unexplainable spikes in latency, are identified and either removed or capped to prevent them from skewing the model's training.

- **Data Normalization and Scaling**: Different metrics from the NWDAF, such as data volume (in bytes) and transfer time (in milliseconds), operate on vastly different scales. To ensure that no single feature dominates the learning process, the data is normalized or scaled to a common range (e.g., [0, 1] or a zero mean and unit variance). This step is particularly important for gradient-based optimization algorithms used in deep learning, as it leads to faster convergence and better performance.

- **Feature Selection**: The NWDAF provides a wide array of analytics, and not all of them are equally relevant for a specific ML policy. Feature selection involves identifying the most impactful features that contribute to the model's accuracy while reducing dimensionality. For instance, for a load balancing policy, a model might rely heavily on slice load and NF load, while other metrics might be less important. This process helps to reduce model complexity, mitigate the risk of overfitting, and improve computational efficiency.

- **Time-Series Feature Engineering**: Since network behaviour is dynamic and changes over time, time-series data from the NWDAF must be converted into features that capture temporal dependencies. This includes creating lagged features (e.g., load at time t-1), rolling averages (e.g., average latency over the last minute), and trend indicators. For predictive analytics, these engineered features are essential for a model to forecast future network states and make proactive decisions.

- **Handling Categorical Data**: Metrics such as slice type or a user's geographical distribution are often categorical. These must be encoded into a numerical format that

ML models can process. Techniques like one-hot encoding or label encoding are used to transform these non-numerical features into a suitable representation.

2.5.3 ML Deployment and Integration Strategy

The effectiveness of an ML policy is not only determined by the model's performance but also by its deployment strategy and seamless integration into the network's operational framework. This subsection discusses the various approaches to deploying ML policies and how they are integrated to enable a closed-loop automation system.

- **Centralized vs. Distributed ML Deployment**:
  - **Centralized Deployment**: In this model, the ML policy is trained and executed on the central E2E orchestrator. This approach provides a global view of the network, allowing the model to make holistic decisions based on comprehensive data from all domains (RAN, Core, Transport). The main benefits are simplified data management and a single point of control. However, this can introduce higher latency for real-time decisions, and the central orchestrator becomes a potential bottleneck and single point of failure.
  - **Distributed Deployment**: Here, smaller ML models or policy agents are deployed on each domain orchestrator (e.g., RAN and Core). These local models are trained in domain-specific data and make decisions with very low latency. This approach enhances resilience and scalability. The key challenge lies in ensuring that the local decisions do not conflict with each other and that global policy objectives are still met.
- **Hybrid Deployment Strategy**: A hybrid approach combines the strengths of both models. A powerful MTLF on the central NWDAF can train a global model using a vast dataset from the entire network. This global model then generates smaller, pre-trained models that are deployed and fine-tuned locally on the domain orchestrators. This allows for rapid, low-latency decisions at the edge while benefiting from the global intelligence of the central training function.
- **API-based Integration**: The ML policies, regardless of their deployment location, must expose their capabilities to the orchestrators through well-defined APIs. The orchestrators send a request with the current network state (the pre-processed data), and the ML model returns a recommended action (e.g., a resource allocation change, a function placement decision). This standardized interface ensures that different network vendors and functions can interoperate seamlessly with the ML policy.
- **Closed-Loop Automation**: The ultimate goal is to create a closed-loop system where data collection, analysis, decision-making, and action are fully automated. NWDAF continuously feeds analytics to the ML policies. The policies provide decisions to the orchestrators, which then execute the required actions in the network. The effects of these actions are then reflected in the new data collected by the NWDAF, completing the feedback loop and enabling the system to continuously learn and optimize itself.

2.5.3.1 Lifecycle-aware ML policy

In 5G/6G systems, “ML policy” is often interpreted narrowly as the set of actions produced by an ML model (e.g., a recommended configuration, a slice adjustment, a resource allocation decision). In integrated TN–NTN deployments, this interpretation is incomplete because the *ability to execute the ML workflow itself* becomes a first-order constraint. In particular, satellite SWaP limitations, feeder-link capacity, and link availability/coverage windows may dominate the end-to-end performance and cost of ML-based closed-loop control. For this reason, ML

policy implementation in NTN must be lifecycle-aware, i.e., it must explicitly govern *where and when* each phase of the AI lifecycle is executed, and how frequently the learning loop is refreshed, in addition to specifying the control action to apply.

A lifecycle-aware ML policy manages the ML workflow as a sequence of phases that may be distributed across ground and space segments:

1. **Data acquisition and aggregation**: collection of telemetry, measurements, and service indicators (e.g., RAN counters, QoS/QoE metrics, transport conditions, satellite payload status). In NTN, continuous streaming of raw telemetry to ground may be infeasible due to feeder constraints; therefore, the policy must determine the degree of local aggregation and compression, and whether raw data, features, or only summary statistics are exported.

2. **Data preparation and feature generation**: filtering, normalization, windowing, and feature extraction to create the model input. This phase is particularly relevant in NTN because feature generation can drastically reduce the volume of information to be transported across the feeder link. A lifecycle-aware policy must decide whether feature generation is executed locally (e.g., on-board or at an edge gateway) or centrally (e.g., in the non-real-time domain).

3. **Model training and validation**: creation or refresh of ML models (e.g., xApps/rApps logic, reinforcement learning agents, anomaly detection models). Training is typically compute-intensive and is commonly centralized in high-capacity environments (e.g., cloud/edge datacenters). In NTN, training may also be constrained by dataset transfer latency (time to deliver training data to the training node) and by the cadence at which new data becomes available due to orbital intermittency.

4. **Model dissemination and activation**: delivery of trained model artifacts (weights, rules, policies, hyperparameters, thresholds) to the execution points, followed by versioning, rollback, and activation. This phase is essential for safe operation: the policy must define model version control, activation conditions, and fallbacks if dissemination fails or is delayed.

5. **Inference and decision execution**: repeated evaluation of the model to produce near-real-time decisions. This phase is often latency-critical and must be aligned with the O-RAN control timescales. In NTN, inference may need to be executed closer to the controlled resources (e.g., on-board or at a local edge node) to avoid round-trip delays and to ensure continuity during feeder outages.

6. **Monitoring, drift detection, and refresh scheduling**: continuous assessment of model performance, detection of data/model drift, and scheduling of retraining or parameter adaptation. For NTN, refresh scheduling must consider both operational dynamics (how quickly the environment changes) and connectivity constraints (when and how refresh data and/or model artifacts can be exchanged).

From an implementation viewpoint, lifecycle awareness means that the ML policy is no longer only “what configuration should be applied,” but also a placement and cadence policy, comprising at least the following decisions:

- **Placement**: selection of the execution domain for each lifecycle phase (e.g., on-board LEO processing, GEO hub, ground cloud, edge gateway).

- **Granularity**: selection of the information exchanged across domains (raw telemetry vs features vs model artifacts).

- **Cadence**: selection of retraining periodicity, model update periodicity, and inference periodicity.
- **Fallback behavior**: definition of safe-mode policies when a phase cannot be completed (e.g., feeder unavailable, model update delayed, local compute saturated).

These decisions must be resolved subject to multiple constraints and objectives. Typical lifecycle-aware constraints include:

- **Communication constraints**: feeder-link throughput limitations, time-varying availability, and the overhead of transferring telemetry, features, or model artifacts.
- **Compute and energy constraints**: limited processing capability and energy budget on-board (or at remote edge sites), which cap model complexity and inference rate.
- **Latency constraints**: end-to-end control-loop delay requirements, especially for Near-RT decisions, that may be incompatible with "ground-only" inference.
- **Reliability and continuity constraints**: ability to maintain acceptable control behavior during intermittent connectivity, including local autonomy requirements.

In practice, these constraints translate into a set of lifecycle KPIs that the policy aims to optimize or keep within bounds. Representative KPIs include:

- **Lifecycle latency**: time to complete one learning-update cycle (data acquisition → training → dissemination → activation), and time to complete one inference-and-actuation cycle.
- **Lifecycle energy/cost**: total energy spent across communication and computation over a given horizon, or equivalently the operational cost of sustaining a given ML loop.
- **Control quality under intermittency**: degradation of policy optimality when the model is stale (time since last refresh) or when decisions must be taken locally with partial/global information.

A lifecycle-aware ML policy therefore acts as a supervisory layer that coordinates ML operations with network control. It can be implemented as a rule-based decision layer (e.g., feasibility thresholds that trigger moving inference on-board or changing refresh periodicity) or as an optimization layer that selects lifecycle placement and cadence to minimize a cost function under constraints. Importantly, this supervisory policy is complementary to the internal logic of the ML model: it governs *how the model is produced, distributed, and executed* in a way that is consistent with end-to-end service requirements and the operational characteristics of TN–NTN connectivity.

Finally, lifecycle awareness aligns naturally with O-RAN's hierarchical control paradigm. Non-real-time functions (e.g., training, global optimization, long-term analytics) can be placed in centralized domains, while Near-RT functions (e.g., inference-driven adaptations) can be placed closer to the controlled elements when required by latency or continuity. In NTN, this mapping is not only a design preference but often a necessity: if the feeder link cannot support timely telemetry exchange or if round-trip delays violate the Near-RT control timescale, then the lifecycle-aware policy must shift parts of the inference pipeline toward local execution and rely on periodic, compact updates of model artifacts rather than continuous centralized control.

2.5.3.2 Mapping policy functions onto O-RAN interfaces

In O-RAN–compliant deployments, the implementation of ML-driven control is not only defined by where analytics and learning functions are executed, but also by which O-RAN interfaces are used to transport data, policies, and model artifacts across the control hierarchy. This mapping is particularly important in integrated TN–NTN systems, where feeder-link capacity and intermittent connectivity constrain the volume, timing, and direction of information exchange. A practical ML policy implementation must therefore specify (i) the *control timescale* of each decision, (ii) the *information objects* that must be exchanged (telemetry, features, models, policies), and (iii) the *O-RAN interface* used to exchange them.

A lifecycle-aware mapping can be described along three main O-RAN interfaces, O1, A1, and E2**,** which jointly support (a) management and telemetry, (b) policy and model distribution, and (c) near-real-time control, respectively.

O1 interface (management and telemetry plane). The O1 interface provides FCAPS/management-plane functions, including configuration, fault management, performance monitoring, and collection of measurements and counters. From an ML-policy perspective, O1 is the primary channel to acquire the *observability* needed for analytics and learning: KPI streams, logs, traces, counters, and status information from RAN and associated elements. In TN–NTN deployments, however, raw telemetry transported via O1 can be high-volume and continuous, potentially exceeding feeder-link capabilities. Consequently, ML policy implementation should explicitly define what is exported on O1 and at what granularity, for example:

- Exporting aggregated KPIs rather than raw per-UE/per-cell measurements when feeder bandwidth is scarce,
- Performing local feature extraction (or local aggregation) before O1 export,
- Adapting O1 reporting periodicity to link availability windows and operational priorities. In this sense, O1 is best suited for *bulk monitoring and long-timescale data collection*, while lifecycle-aware policies must prevent O1 telemetry from becoming the dominant cost driver in NTN.

A1 interface (non-real-time policy, guidance, and model artifacts). The A1 interface connects the Non-RT RIC (within the SMO domain) to the Near-RT RIC and is designed for conveying high-level guidance, intent, and policies over longer timescales. For ML-driven control, A1 provides the natural mechanism to distribute policy configurations and ML artifacts produced in the non-real-time domain to the near-real-time execution domain, including:

- Policy parameters and constraints (e.g., thresholds, objectives, guardrails),
- Model artifacts (e.g., weights, rules, hyperparameters, feature definitions, reward shaping for RL),
- Activation conditions, validity intervals, and rollback rules (model governance). In TN–NTN systems, A1 is particularly attractive because model artifacts and policy descriptors are typically orders of magnitude smaller than raw telemetry, making them more compatible with feeder constraints. Therefore, an efficient implementation pattern is to use O1 sparingly for telemetry (preferably aggregated/features), while using A1 as the primary channel for compact, versioned dissemination of models/policies from the training domain to the execution domain.

E2 interface (near-real-time control loop). The E2 interface connects the Near-RT RIC to RAN nodes (e.g., gNB components, O-CU/O-DU/O-RU) to support near-real-time monitoring and control via E2 service models (E2SM). In ML policy implementation, E2 carries the time-critical part of the loop: subscription to near-real-time indications and delivery of control actions to enforce the selected policy (e.g., RAN parameter tuning, admission decisions, scheduling-related configurations exposed through the relevant service models). Because E2 is intended for near-real-time control, it is sensitive to RTT and availability: in NTN, depending on the topology, routing E2 interactions through a ground-based controller may violate latency constraints or become unreliable during feeder outages. For this reason, the ML policy implementation must ensure that E2-based enforcement is placed such that:

- Near-RT inference and action selection occur close to the controlled RAN elements when latency requirements are strict,
- The system can maintain local autonomy (degraded but safe control) during connectivity disruptions,
- E2 control traffic is bounded and stable, avoiding excessive oscillations due to delayed feedback.

Putting the mapping together: a lifecycle-driven interface usage pattern. A consistent mapping emerges when the ML lifecycle is aligned with O-RAN timescales:

- Training / global optimization / long-term analytics are executed in the non-RT domain (SMO/Non-RT RIC), using O1 primarily for telemetry acquisition (preferably aggregated or feature-level) and A1 for distributing the resulting model/policy artifacts.
- Near-RT inference and enforcement are executed in the Near-RT domain, using E2 to apply control actions and collect near-real-time indications, while receiving periodic guidance and updated models via A1.
- Governance and safety (versioning, guardrails, rollback) are implemented through A1 policy descriptors and SMO procedures, ensuring that even when connectivity is intermittent, the Near-RT domain operates with a well-defined and bounded policy behavior.

In summary, mapping ML policy functions onto O-RAN interfaces is a key enabler for practical deployment. In NTN scenarios, the mapping must be made explicit and lifecycle-aware: O1 should be treated as a constrained channel for observability, A1 as the preferred mechanism for compact and versioned model/policy dissemination, and E2 as the channel for time-critical execution—ideally positioned to satisfy latency and continuity requirements under feeder-link limitations and intermittent coverage. The end-to-end placement of these functions and the corresponding information flows across analytics, orchestration and control domains are consistent with the integration view shown in Figure 2-52**.**

2.5.3.3 Split-RIC feasibility scenarios for NTN

To operationalize the centralized, distributed, and hybrid ML deployment options discussed above in an NTN setting, we consider a set of Split-RIC feasibility scenarios that instantiate *where* the Non-RT and Near-RT control functions are placed and *how* the ML lifecycle is executed across ground and space segments. The objective is to translate “deployment strategy” into implementable alternatives whose feasibility can be assessed against feeder-link capacity/availability, on-board processing limits, and control-loop latency requirements.

The scenarios below capture representative and practically relevant design points for integrated TN–NTN O-RAN deployments.

Scenario S1 — Ground-centric intelligence (centralized ML control)

In this baseline scenario, the Non-RT RIC and Near-RT RIC functions remain on the ground, and satellites primarily provide transport and connectivity. Network measurements and telemetry are exported from the space segment toward the ground domain (e.g., via O1 reporting), where feature engineering, model training, and inference are executed. Control decisions are then sent back to the relevant RAN elements through the standard O-RAN control chain.

This scenario aligns with a fully centralized ML implementation and is attractive for its simplicity and for leveraging abundant ground/cloud computing resources. However, it can become infeasible or inefficient in NTN when (i) high-rate telemetry must be continuously transported over feeder links, (ii) round-trip delays prevent meeting Near-RT control timescales, or (iii) feeder availability windows introduce gaps that slow down the learning loop and reduce the timeliness of control decisions.

Implementation characteristics:

- **O1** is used predominantly for telemetry export.
- **A1** is used for policy configuration but not essential for inference placement.
- **E2** is used for enforcement occurs from a ground-based Near-RT RIC, which may be challenged by NTN latency.

Scenario S2 — Ground–LEO Split-RIC (distributed inference, centralized training)

Scenario S2 implements a functional split consistent with O-RAN's hierarchical control: training and long-term optimization remain centralized (ground Non-RT domain), while near-real-time inference is pushed closer to the controlled resources, e.g., on-board LEO payload processing or at a LEO-associated edge node. In this approach, raw telemetry is not continuously streamed to ground; instead, local processing generates features and executes inference, thereby reducing feeder traffic and improving control-loop responsiveness.

The learning loop is maintained by periodically transferring compact training data summaries and/or selected samples to ground (or by storing them until connectivity allows transfer), and by disseminating updated model artifacts back to the LEO execution point. This scenario is typically preferable when the input data for inference is high-volume, when Near-RT latency is strict, and when the model complexity is compatible with on-board compute and energy budgets.

Implementation characteristics:

- **O1** reduced to aggregated KPI export and/or event-driven reporting;
- **A1** used for **model/policy artifact dissemination** from Non-RT to Near-RT execution;
- **E2** control loop executed locally (or closer to the RAN elements), improving Near-RT feasibility.

Scenario S3 — GEO–LEO multi-layer Split-RIC (hierarchical learning and continuity)

Scenario S3 extends S2 to address a fundamental NTN issue: orbital intermittency and waiting time can make timely model refresh impractical when the training domain is only on the ground. In S3, near-real-time inference remains on/near LEO (as in S2), but the non-real-time learning and/or aggregation functions are moved to an intermediate high-availability domain, such as a GEO node acting as a training hub (or an always-on space segment with stable inter-satellite connectivity). LEO nodes exchange data and model updates with GEO via ISLs, enabling more continuous learning/update cycles than ground-pass opportunities alone.

This scenario is suitable when the environment is dynamic, and model updates must be frequent, or when service requirements demand a tighter bound on *learning-loop latency* than ground connectivity can guarantee. The trade-off is that S3 may increase overall system complexity and potentially energy consumption due to additional space-segment processing and data transfers, but it can significantly improve update timeliness and autonomy.

Implementation characteristics:

- **O1** primarily local within the space segment; ground O1 export can be minimized.
- **A1** can be implemented along the hierarchy (ground→GEO, GEO→LEO) for policy/model dissemination.
- **E2** remains local for Near-RT control; GEO supports continuity of the non-real-time learning loop.

Feasibility rationale and selection drivers

The three scenarios provide a structured design space for NTN ML policy implementation:

- **S1** is preferred when telemetry volume is modest, feeder capacity is sufficient, and RTT allows near-real-time control from ground.
- **S2** becomes advantageous when feeder constraints make telemetry export expensive, or when control must operate at Near-RT timescales that are not compatible with ground-based inference.
- **S3** is motivated when waiting time / connectivity intermittency would otherwise slow down model refresh and degrade control quality, making a multi-layer approach necessary to sustain timely learning and policy updates.

In the following, these scenarios can be evaluated against lifecycle KPIs (e.g., lifecycle latency and energy/cost) and mapped to O-RAN control functions and interfaces, enabling the ML policy engine and orchestration layer to select the most appropriate Split-RIC implementation for a given TN–NTN operating regime. The three Split-RIC feasibility scenarios are summarised in Figure 2-53, highlighting how the AI lifecycle functions (training, model dissemination and inference) can be distributed across ground and space segments under TN–NTN constraints.

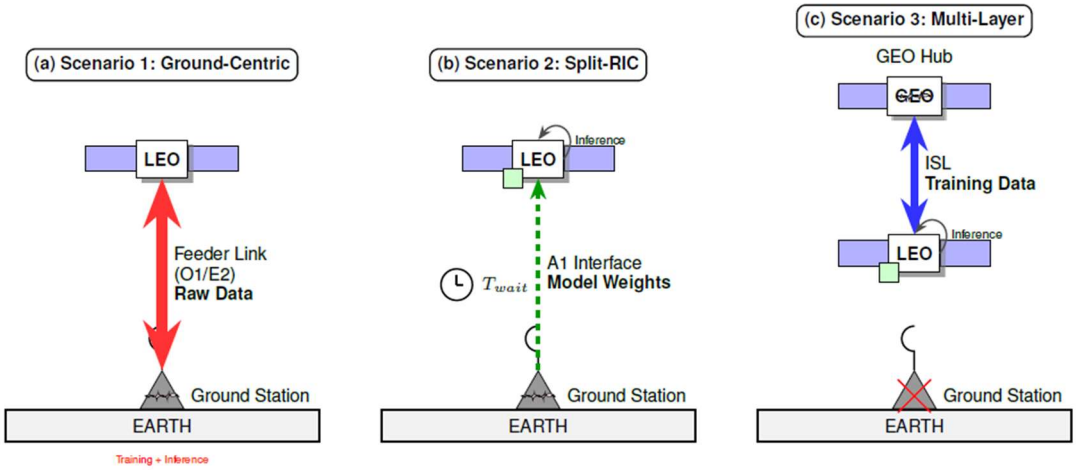


Figure 2-53: Split-RIC placement options for TN–NTN

2.5.3.4 Quantitative feasibility criteria

The Split-RIC scenarios introduced above (S1–S3) are not only architectural alternatives; they can be selected and dimensioned using quantitative criteria that capture the dominant constraints of TN–NTN deployments. In particular, NTN feasibility is largely determined by the interplay between (i) the *communication burden* imposed on feeder links, (ii) the *computational/energy limits* of space-segment processing, and (iii) the *timeliness* requirements of both the near-real-time control loop and the learning/update loop. To this end, we define a set of metrics and inequalities that support an implementable decision logic for choosing between S1 (ground-centric), S2 (Ground–LEO split), and S3 (GEO–LEO multi-layer).

Lifecycle decomposition and key variables

We model ML policy implementation over an observation horizon in terms of three recurring lifecycle components:

- **(i) Data transfer for learning**: exchange of training data (or selected samples/statistics) needed for model refresh.
- **(ii) Model dissemination**: delivery of model artifacts and policy descriptors to the execution domain.
- **(iii) Repeated inference**: execution of near-real-time inference at a given rate to produce control actions.

The following variables capture the main drivers:

- R_{feed}: effective feeder-link throughput available to the ML workflow (accounting for scheduling and overheads).
- T_{wait}: waiting time/intermittency factor affecting when data/model exchanges can occur (e.g., ground-pass gaps, access-window delays).
- Δ_{train}: size of the training dataset (or the amount of information transferred for training per refresh).
- Δ_{model}: size of the disseminated model artifact (weights, parameters, policies).

- δ_{in}: size of the inference input (raw telemetry or features) per decision.
- f_{inf}: inference rate (decisions per second) required by the Near-RT control loop.
- C_{inf}: computational cost per inference (e.g., operations or CPU cycles).
- P_{comp}: available on-board compute power/energy budget for ML processing.

While the precise numeric values depend on the selected xApp/rApp and service model, these variables provide a compact abstraction that can be instantiated for specific use cases (e.g., slicing, mobility optimization, anomaly detection).

1) Communication feasibility: feeder-link budget constraints

A first feasibility criterion is whether the required information exchange fits within the feeder-link budget over the relevant time window. For a given scenario, the communication load can be approximated as:

- **Scenario S1 (ground-centric)**: continuous telemetry export dominates, because inference is performed on the ground. The feeder load scales with the near-real-time observation stream:

$$L_{\text{S1}} \approx f_{\text{inf}}\,\delta_{\text{in}} + \frac{\Delta_{\text{train}}}{T_{\text{refresh}}} + \frac{\Delta_{\text{model}}}{T_{\text{refresh}}}$$

  where $T_{\text{refresh}}$is the model refresh period. S1 is feasible only if:

$$L_{\text{S1}} \leq R_{\text{feed}}$$

  In NTN, this condition is often violated when $\delta_{\text{in}}$is large (raw telemetry) and $f_{\text{inf}}$is high (tight Near-RT loop).

- **Scenario S2 (Ground–LEO split)**: near-real-time telemetry is processed locally and does not need to traverse the feeder. The feeder load is then dominated by model dissemination and (possibly) training data transfer:

$$L_{\text{S2}} \approx \frac{\Delta_{\text{train}}}{T_{\text{refresh}}} + \frac{\Delta_{\text{model}}}{T_{\text{refresh}}}$$

  with feasibility condition $L_{\text{S2}} \leq R_{\text{feed}}$. Because Δ_{model} is typically far smaller than streaming telemetry, S2 often relaxes the feeder bottleneck substantially.

- **Scenario S3 (GEO–LEO multi-layer)**: the dominant exchange may shift to ISL transport and GEO aggregation, while the ground feeder is reduced. The feeder feasibility condition is evaluated for the ground–GEO exchange (if applicable) and for the GEO–LEO ISL capacity; the key point is that learning/update exchanges are no longer governed by ground-pass opportunities alone.

These relations motivate a first, practical decision rule: *if the near-real-time telemetry export term $f_{inf}\delta_{in}$ threatens to saturate the feeder link, S1 becomes inefficient or infeasible and Split-RIC (S2/S3) should be considered.*

2) Compute and energy feasibility: on-board execution constraints

When inference is moved on-board (S2/S3), feasibility requires that the payload compute can sustain the inference workload:

$$P_{\text{req}} \approx f_{\text{inf}}\, C_{\text{inf}}$$

and the on-board feasibility condition is:

$$P_{\text{req}} \leq P_{\text{comp}}$$

This criterion captures the "hardware ceiling": if the required inference rate and model complexity exceed available compute/energy, on-board inference cannot be sustained, and the system must revert to a more centralized design (or use lighter models / lower inference rates / stronger feature compression). In practice, this condition can be monitored at runtime (resource usage, thermal constraints, battery margin) to trigger adaptive fallback policies (e.g., reduce inference cadence, switch to a simpler model, or temporarily offload inference).

3) Timeliness feasibility: control-loop and learning-loop latency

Two latency constraints matter in NTN:

- **Near-real-time control timeliness**: the end-to-end delay between observation and actuation must be within the allowed Near-RT bound T_{NR}. When inference is ground-based (S1), the delay includes propagation and scheduling across the feeder:

  $$T_{\text{S1}}^{\text{NR}} \approx T_{\text{uplink}} + T_{\text{proc,ground}} + T_{\text{downlink}}$$

  and must satisfy $T_{\text{S1}}^{\text{NR}} \leq T_{\text{NR}}$. If feeder RTT is large or variable, this condition may fail.

  When inference is local (S2/S3), the Near-RT delay is dominated by local processing and local control signaling:

  $$T_{\text{S2/S3}}^{\text{NR}} \approx T_{\text{proc,local}} + T_{\text{local\_ctrl}}$$

  which is typically much smaller and more stable, improving Near-RT feasibility.

- **Learning/update timeliness**: the model refresh cycle must complete within a maximum staleness bound T_{stale}driven by environment dynamics and acceptable control degradation. In NTN, waiting time/intermittency can dominate this cycle. A simple approximation for the refresh-cycle time is:

  $$T_{\text{refresh\_cycle}} \approx T_{\text{wait}} + \frac{\Delta_{\text{train}}}{R_{\text{path}}} + T_{\text{train}} + \frac{\Delta_{\text{model}}}{R_{\text{path}}}$$

  where $R_{\text{path}}$is the effective throughput on the relevant exchange path (feeder or ISL), and $T_{\text{train}}$is training time in the selected domain. Feasibility requires:

  $$T_{\text{refresh\_cycle}} \leq T_{\text{stale}}$$

  This criterion is the primary driver for Scenario S3: if T_{wait}(ground-pass gaps) makes $T_{\text{refresh\_cycle}}$ exceed the acceptable staleness bound, then a multi-layer approach that reduces waiting time (e.g., via GEO hub + ISL continuity) becomes necessary.

4) Combined decision logic (operator-friendly rules)

The above constraints can be combined into a pragmatic selection logic:

- **Choose S1 (ground-centric)** if (a) feeder capacity is sufficient for telemetry export ($f_{\mathrm{inf}}\delta_{\mathrm{in}} \ll R_{\mathrm{feed}}$), and (b) Near-RT timeliness can be met from ground ($T_{\mathrm{S1}}^{\mathrm{NR}} \leq T_{\mathrm{NR}}$), and (c) learning refresh meets staleness bounds under intermittency.
- **Choose S2 (Ground–LEO split)** if S1 violates feeder or Near-RT latency constraints, **and** on-board compute can sustain inference ($f_{\mathrm{inf}}C_{\mathrm{inf}} \leq P_{\mathrm{comp}}$) S2 is particularly effective when inference inputs are high-volume and compressible into local features, while model artifacts remain compact.
- **Choose S3 (GEO–LEO multi-layer)** if on-board inference is feasible (as in S2) but the learning/update loop is not timely due to waiting time ($T_{\mathrm{refresh\_cycle}} > T_{\mathrm{stale}}$) when training is ground-coupled. Introducing a higher-availability training/aggregation hub reduces the effective T_{wait} and restores refresh timeliness.

Overall, these criteria enable the ML policy implementation to be expressed as a constraint-aware placement and cadence decision: telemetry volume and RTT push inference toward the space segment (S2/S3), compute/energy limits cap model complexity and inference cadence, and waiting time/intermittency determines whether a two-layer (S2) or multi-layer (S3) learning loop is required to keep the policy sufficiently up to date. The resulting trade-offs are visualised through the energy and latency feasibility maps in Figure 2-54, which provide an operator-oriented view of when each Split-RIC scenario (S1–S3) is feasible and cost-effective.

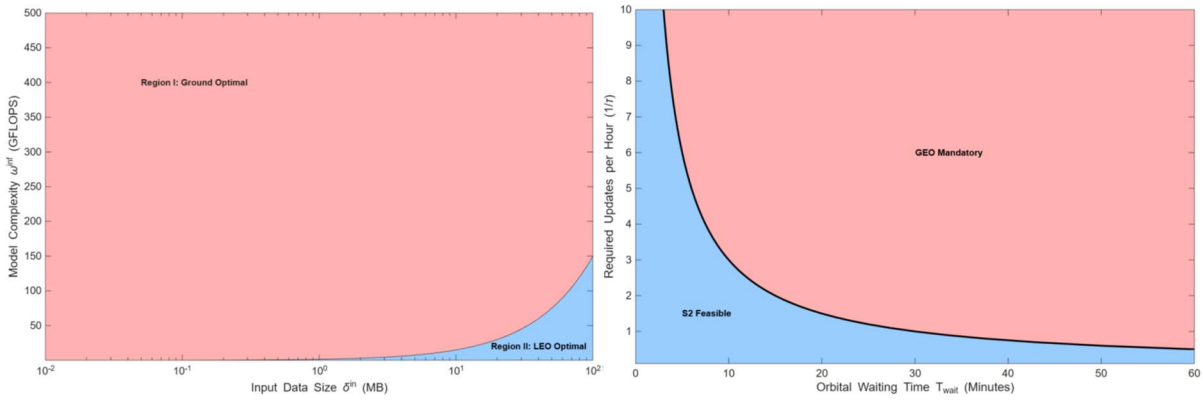


Figure 2-54: Energy and latency feasibility maps for Split-RIC

2.5.4 Domain NWDAFs and Network Orchestrators

The main input to the domain orchestrator comes from the E2E orchestrator since it has the information regarding the overall data flow. Although not explicitly shown in Figure 2-52, NWDAF also provides input to domain orchestrators. The orchestrators can either use the data from AnLF to train local models or use the already trained models from MTLF depending on the available resources for Machine Learning purposes. The fact that NWDAF can share also predictive information reduces the resource demand on the domain orchestrators for implementing a Machine Learning Algorithm.

3GPP [1] has defined in the standards that different instances of NWDAF can be present, which can be specialized in certain aspects of the network. NWDAFs can contain metrics for specific analytics which vary across different instances of NWDAF or different instances of NWDAF can contain the same metric, but for different type/geographic location of the network. This paves the way for different deployment options which can enhance the efficiency of

NWDAF, thereby increasing the efficiency of the domain orchestrators. Here, we are suggesting an approach in which each domain has a dedicated NWDAF for the analytics of each domain. The NWDAF of each domain assists the corresponding domain orchestrators in optimizing the network. The proposed architecture has been shown in Figure 2-55. The scenario envisaged here is an integrated TN and NTN connectivity where the UE is always connected by both the networks (if both are in range). The CN has only one NWDAF while the transport network and RAN have two NWDAFs each. This is due to the fact that the network has one CN while there are two transport networks and RANs between the UE and the CN viz., terrestrial and non-terrestrial. The characteristics of both networks are different, which is better to be handled by different NWDAFs. The NWDAF assists the domain orchestrators to efficiently handle the resources in the corresponding domain. The MTLF may contain already trained model, so that the orchestrators can use the readily available model for load prediction.

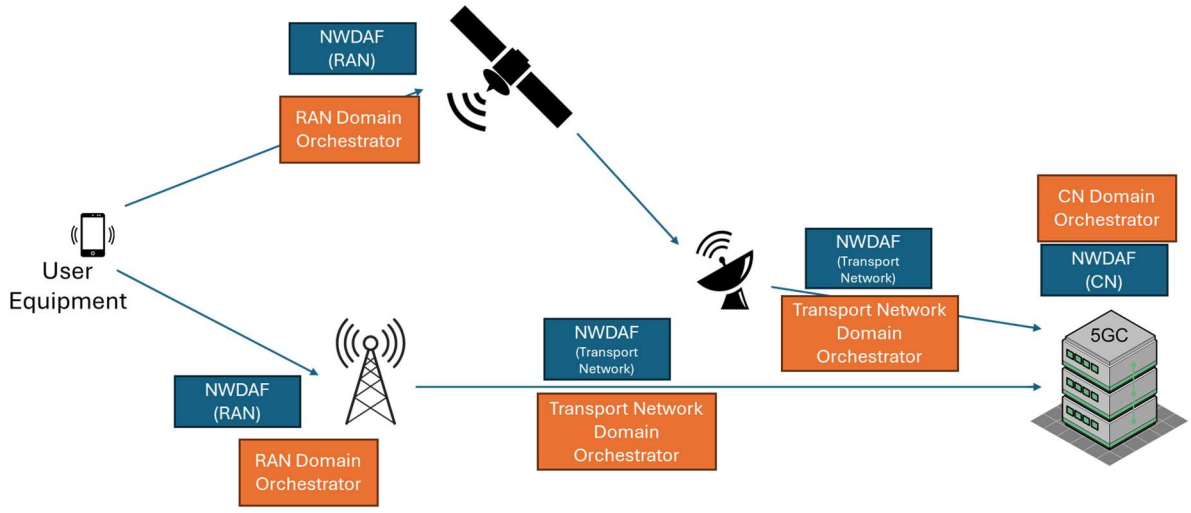


Figure 2-55: NWDAF Deployment strategy

In the following sections, each domain will be discussed in detail.

2.5.4.1 Core Network

The CN is composed of many Network Functions (NFs) with each entity playing a predefined role in the network. The CN will have its NWDAF which collects data from each of the NF in the core network. The role of this NWDAF is to maintain the analytics and train the models that are needed for the functioning of the CN. NWDAF collects data from all the NFs in order to come up with the analytics and statistics pertaining to different network parameters. NWDAF handling CN statistics can be centrally placed, which helps to aggregate the data in a centralised fashion. This will result in low signalling and easier coordination among the different NFs to send the metrics to NWDAF. This is a good approach also due to the fact the CN NFs are usually placed at a central location of the network.

The slicing in the core network mainly consists of handling the VNFs for the core functionalities. The optimization process happens at regular intervals to cope with the changing traffic demands. The information that can be used from NWDAF by the domain orchestrator is PDU session traffic analytics, slice load level related network data analytics, and NF load analytics. The current statistics can be used for training already existing AI models, or the predicted values available in NWDAF can be used for the optimization process if the models perform with satisfactory results.

The main aspects involved in slicing optimization are:

- Traffic flow re-routing based on the placement of the VNFs

  The data traffic has to be rerouted so that the packets reach the required VNFs. Hence, the placement of the VNFs should not result in unrealistic number of hops to be carried out by the data packets. A core network slice can be modelled as a set of virtual links or Service Function Chains (SFC) that share a common SLA [1]. This will help create a slice that can be easily traversed by the data packets. The placement of the VNFs should not be modified frequently since it will affect the stability of the system. The creation and deletion processes of the VNFs on the underlying hardware consume a lot of resources. It involves control and management overhead, rerouting overhead and retransmission overhead due to data loss.

- Scaling of the VNFs to accommodate the changing resource demands

  The network resources have to be scaled based on the changing traffic. An efficient prediction of the traffic demand can help in proactively reconfigure the slices. This reconfiguration should be carried out in a way that the number of reconfigurations is minimized. The optimisation of the number of reconfigurations can be done using Deep Q Network (DQN). But it has the drawback that the number of action spaces makes it difficult to explore efficiently. A Branch Dueling Q-network which incorporates the action branching architecture into DQN is a good approach to mitigate this issue [14]. Hence, a smart network slicing reconfiguration algorithm can be developed using BDQ.

For both the above-mentioned CN slicing aspects, efficient forecasting of data traffic is essential. Long Short-Term Memory (LSTM) is a commonly used algorithm to predict incoming traffic, as it can handle spatial-temporal changes in real time. But recent studies have shown that KAN performs better in predicting the variations in data traffic. The use of Federated-KANs which increases data privacy and decreases training data communication overhead has proved to be a promising approach [15]. The choice of the AI algorithm to predict data traffic should also take into consideration the computational complexity and the time to converge.

The CN domain orchestrator is a good candidate to be deployed centrally. The CN NWDAF is also deployed centrally which helps the domain orchestrator to optimally allocate the resources. This is also due to the fact that normally the CN NFs are not distributed across the network but rather implemented in a central location. There are very few exceptions, like the option of deploying UPF in space to reduce the latency in integrated TN-NTN systems. However, the installation of core NFs in a distributed manner is not a commonly adopted option. The other important aspect that supports the assumption to centrally control the CN domain orchestrator is that the expected optimization and reconfiguration of the slices should take place at larger time intervals in the order of hours/days, thereby avoiding real time interaction with the network resources. Hence, a central deployment will help in easily handling the AI models and the optimization process.

2.5.4.2 Transport Network

The transport network acts as a link between CN and RAN. This is referred to as the backhaul link. The backhaul link implementations can be fibre-based, wireless or copper-based wireline. With the introduction of VLEOs, satellite-based backhaul links are also considered for 5G communication systems. In the current RAN split options, transport network also comprises fronthaul and midhaul links connecting RU to DU and DU to CU respectively. The transport network infrastructure provider divides the resources into tenants which are shared with service providers. This assignment of resources to service providers is on a demand basis. Unlike in traditional approach where the SLAs have to be satisfied only during the assignment, network slicing mandates the SLA adherence throughout the life cycle of the transport slice. This plays a major role in meeting the QoS demands as the transport network contributes

significantly to the E2E latency. The transport network is distributed throughout the network, and a central approach can pose optimization issues due to the heterogeneous nature of the constituting components. De-centralization of NWDAF for transport network is the ideal solution. Terrestrial and NTN can take advantage of this approach by having dedicated NWDAFs for both networks.

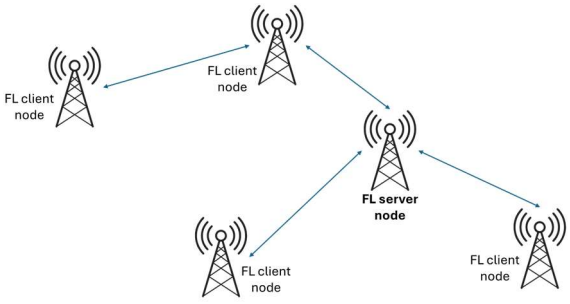
Since the costs incurred on the service providers are based on the resources consumed, it is important that the over provisioning of the resources should be minimized. At the same time, underprovisioning can result in SLA violations. The use of AI for estimating the resource demands can help in tailoring the network slices. The infrastructure provider and the service providers should work closely to minimize this gap. DRL methods can be used to optimize the costs while meeting the required SLAs [16][17].

The deployment option that could be effective for a transport network domain orchestrator is a distributed approach. Since the transport network is distributed across the network, a centralized control can result in delays and control signal overhead. The dynamicity of the network can also jeopardize the stability of the optimization algorithm since the status updates to a centralized orchestrator could be stale.

2.5.4.3 RAN

The NWDAF that handles RAN has two components: Terrestrial and Non-terrestrial, since the dynamics of the network are different. A common NWDAF increases the complexity and also the latency of operation. The optimization in RAN happens at two different time scales: Real Time and non-Real Time. A Federated Learning (FL) approach is an ideal candidate for RAN optimization.

The FL approach for RAN can be approached in a centralized or decentralized manner as shown in Figure 2-56. In a centralized approach, the local models from each FL client are sent to a centralized entity or a server. The global model is created by the server, which is updated back to the clients. In the example below, the FL clients send the local models to the FL server where the global model is created. The developed global model is then sent to all the clients so that the updated model is available to all the clients in the network. While in a decentralized approach, the local models are sent to only nearby clients and the accumulated model is derived only from the local models of the neighbouring nodes. In the example given below, sat1 collects the global model from the neighbouring nodes sat2, sat3, sat4, sat5. The model used by sat1 is developed using these local models from the neighbouring nodes. Hence, the cumulated model used by each satellite will be different.



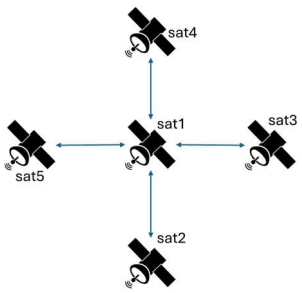


Figure 2-56: FL for TN: Centralised *FL for NTN: De-Centralised*

The centralized approach is effective for terrestrial RAN, since the network will be dealing with similar users throughout the operational timeframe. The geographical area over which the service is provided by the RAN is the same. In the case of a non-terrestrial RAN including

satellites, the de-centralized approach is a promising candidate. The users serviced by each satellite change throughout the orbital plane, and the traffic variations are not the same. A satellite servicing a rural area at a particular time instant can be over an urban area during another interval of time. The same satellite can also be servicing maritime users while it is above the oceans. During this timeframe, a satellite might not have any ground users but only serve as a relay satellite or an edge computing node based on the demand. Hence, a de-centralized approach would help the satellite NWDAFs to vary the analytics or the developed ML model as per the changing traffic conditions in the nearby nodes.

The RAN resources comprise bandwidth, CPU, transmit power, etc. Both RL and supervised learning approaches can be utilized for RAN resource allocation. The selection of an AI learning approach depends on the dataset availability. In dynamic systems where the models have to train ad hoc, RL is the way forward. But the chosen algorithm should not be complex because it consumes critical satellite resources which are limited.

The resource allocation scheme can be coarse grained or fine grained. In a coarse-grained approach, optimization is performed by considering the average values in the optimized time window. In a fine-grained approach, optimization is considering the end user experience which has to be monitored continuously. Both approaches are applied for RAN slicing based on the requirement.

Traffic variations at slice level are usually observed over time intervals of hours/days [18]. Hence, a coarse-gained approach at slice level would suffice the optimization process at slice level. The user traffic variations are evident at time intervals in the order of minutes/seconds. Hence, a fine-grained approach is necessary for user level resource allocation. This points to the need for a two-level resource allocation scheme for efficiently handling the resources in RAN. The resource optimization in RAN can be efficiently handled by using DQN. The spectral efficiency and the utility of the slices can be increased by optimizing the bandwidth allocation using a DQN approach [19]. It is also possible to optimize transmit power and CPU allocation using the same approach [20]. To ensure the privacy of the end users, it is essential to perform the learning process in a federated approach.

Coarse-level optimization can be handled centrally by the orchestrator, as optimizations are not frequent. Fine-grained optimization can be carried out in a distributed manner. This will prevent the delay in the steps taken for the reconfiguration of the resources. Hence, a hybrid deployment approach suits the needs of a RAN domain orchestrator.

Feasibility-driven placement in NTN RAN: In NTN RAN, the hybrid deployment approach must be complemented with feasibility-driven placement rules that jointly account for (i) Near-RT control-loop latency/continuity requirements and (ii) the timeliness of model refresh under orbital intermittency. In practice, this implies that Near-RT decision functions (e.g., fine-grained resource reconfiguration) should be placed as close as possible to the controlled RAN resources when feeder-link round-trip delays or availability would otherwise prevent timely actuation, whereas training/refresh functions should be placed to minimize model staleness when update opportunities are intermittent. This feasibility-driven hybrid placement is consistent with the NTN trend toward decentralized intelligence highlighted in Figure 2-56.

Operational selection guidelines:

- **Ground-centric feasible:** keep Near-RT inference/control on the ground when feeder-link latency and availability support the required fine-grained control timescale, and the telemetry/feature export does not overload the feeder. This option is appropriate for coarse-grained optimisation and slowly varying conditions.

- **Split-RIC preferred (LEO inference):** move Near-RT inference closer to the NTN RAN (e.g., on-board/LEO-proximal execution) when fine-grained optimisation would be delayed by feeder RTT or when exporting high-volume inputs is inefficient. In this case, training and long-term optimisation can remain centralized, while model/policy artifacts are periodically disseminated to the execution point.

- **Multi-layer preferred (GEO–LEO hierarchy):** introduce an intermediate hub for aggregation/training and model distribution when orbital waiting time between update opportunities makes the learning/update loop too slow, leading to unacceptable model staleness. This preserves LEO-local Near-RT responsiveness while improving the timeliness and continuity of model refresh.

2.5.5 ML Deployment Conclusions

In this section, we have proposed a framework for AI-driven resource management within the Radio Access Network (RAN), tailored for the integrated terrestrial and non-terrestrial 5G-STARDUST architecture. By adopting the principles of the O-RAN alliance, we have proposed a two-level control and optimization scheme that effectively addresses the different timescales of traffic variation.

A comparison of different domains has been shown in Table 2-2. The above discussion reveals that a common approach for the optimization of the slices in different domains might not be efficient, and a domain orchestrator tailored to the requirements of each domain is necessary for the optimization of the resources.

Table 2-2: Comparison of different domains

	Core Network	**Transport Network**	**RAN**
Optimisation interval	Coarse grained	Fine grained	Coarse grained / Fine grained
Deployment strategy	Central	Decentralised	Hybrid
Signalling overhead	Low	High	Medium
Standardisation body	3GPP	IETF	3GPP

The proposed architecture leverages a coarse-grained approach, managed by the Non-Real-Time RIC, to handle slice-level resource allocation over longer time intervals, and a fine-grained approach, managed by the Near-Real-Time RIC, for dynamic user-level resource allocation in near real-time. We have identified Deep Q-Networks (DQN) as a powerful and suitable technique to drive the optimization process, aiming to enhance the utility of network slices by intelligently allocating communication and computational resources.

In integrated TN–NTN deployments, this domain-specific orchestration must also be feasibility-driven. In particular, the placement of AI functions across the O-RAN control hierarchy

(Non--RT RIC vs Near-RT RIC) must account for feeder-link capacity and round-trip delay, intermittent connectivity (waiting time between update opportunities), and the compute/energy constraints of the space segment. These constraints directly impact the AI lifecycle (training data transfer, model dissemination, and repeated inference) and therefore determine whether intelligence can remain ground-centric or must be split, with near-real-time inference moved closer to the controlled RAN functions and model refresh handled through compact policy/model updates.

Furthermore, we have outlined a practical hybrid deployment model, combining centralized control for non-frequent optimizations with distributed control for time-sensitive reconfigurations. This hybrid model can be instantiated through feasibility-based placement rules, selecting ground-centric control when feeder conditions allow, Split-RIC with LEO-proximal inference when Near-RT latency or telemetry volume constrain ground operation, and multi-layer updates when orbital intermittency would otherwise lead to stale models.

2.6 CONCLUSIONS

This section has consolidated the final outcomes of Task 5.2 on AI data-driven network management and slicing in the integrated TN-NTN 5G-STARDUST architecture, using the O-RAN framework as the common control and orchestration foundation. Starting from the motivation for openness, disaggregation, and programmability, the section detailed how data acquisition, analytics, and AI-driven decision-making can be combined to optimize heterogeneous resources spanning terrestrial and non-terrestrial domains.

The first key takeaway is the role of end-to-end monitoring as an enabler of AI-driven control, while explicitly acknowledging the NTN constraint that telemetry collection must balance granularity against the bandwidth and energy cost of transporting data over feeder links. The monitoring discussion motivates adaptive strategies, ranging from aggregated KPI reporting to event-triggered high-rate measurements and onboard pre-processing, that allow AI engines to remain effective without overburdening NTN connectivity.

Building on this monitoring foundation, the section addressed three central optimization levers aligned with T5.2 objectives: beam hopping, bearer selection, and virtualized function placement / load balancing. Beam hopping and bearer selection were framed as mechanisms to improve capacity utilization and service continuity under time-varying NTN conditions, while the function placement problem formalized the multi-tier trade-off between communication latency/energy and computation latency/energy when distributing processing across satellites, gateways, and cloud resources. The presented evaluation results and benchmark comparisons illustrate how intelligent policies can outperform static placements, particularly when load and topology dynamics become dominant.

From an algorithmic standpoint, this section focused on learning-based and probabilistic AI techniques for prediction and control (e.g., DBN-based forecasting and RL/DRL-based decision making), targeting dynamic TN–NTN conditions where explicit modelling is difficult and adaptability is required. The proposed approaches align with the O-RAN control hierarchy, with longer-timescale analytics, model training and policy preparation supported by the Non-RT RIC, and near-real-time inference and action enforcement performed by the Near-RT RIC.

Finally, the ML policy implementation discussion—grounded in NWDAF-supported analytics and closed-loop orchestration—connected AI decision logic to practical deployment through feasibility-driven Split-RIC options. By comparing ground-centric and distributed/hierarchical placements, the section emphasized that controller placement must be selected according to feeder-link capacity and latency, intermittency (waiting time between update opportunities),

and space-segment compute/energy constraints. Overall, this section demonstrates a coherent approach to maximizing capacity/bandwidth across TN and NTN while respecting constraints on connectivity resiliency, latency, and processing/convergence time, and provides a validated basis for AI-enabled TN–NTN resource management beyond the project timeframe

3 END-TO-END QOS MANAGEMENT

3.1 EXECUTIVE SUMMARY

Quality of Service (QoS) management in satellite–terrestrial integrated networks is challenged by dynamic topologies and variable feeder link conditions, especially due to weather impact, that affect both availability and capacity. To address this, an extended fast re-route mechanism has been designed, adapting Multiprotocol Label Switching (MPLS) to enable label swapping across alternative feeder links and thereby protecting complete gateways rather than individual nodes or inter-satellite links. The mechanism has been implemented in an OMNeT++-based discrete event simulator configured with constellation scenarios and variable feeder link performance. Evaluation results indicate that traffic can be preserved during feeder link disruptions and capacity reductions, with rerouting leading to latency variations associated with the alternative path. Redundancy across multiple paths further improves availability and stabilizes end-to-end performance. These findings provide the first experimental validation of extended fast re-route for QoS management in satellite networks.

Note: This section continues the work reported in Section 3 of D5.2, where the concept of extended fast re-route for feeder links was introduced. The present deliverable section complements that design by providing measurement results and their evaluation. The simulator framework itself was described in detail in D5.4 and is only referenced here as the basis for the evaluation.

3.2 INTRODUCTION

The management of QoS in satellite–terrestrial integrated networks is fundamentally different from that in terrestrial infrastructures. In fixed optical backhaul, capacity is stable and dimensioned to meet predictable demand, allowing QoS mechanisms to rely on static traffic engineering and established recovery techniques. In contrast, non-terrestrial networks are characterized by moving satellites, time-varying topologies, and feeder links whose availability and performance fluctuate due to atmospheric conditions. These factors introduce variability into the end-to-end path that classical transport mechanisms were never designed to handle.

The feeder link represents the most critical element in this chain, as it concentrates large volumes of aggregated traffic into comparatively few gateways. When weather-related impairments reduce the available capacity of a feeder link, or when a gateway becomes unavailable, the degradation propagates across all services depending on that path. Inter-satellite connectivity can offer redundancy within the constellation, yet the bottleneck at the feeder link persists as the decisive point where continuity of end-to-end QoS is either preserved or lost. The inability to address this bottleneck effectively risks undermining the integration of satellite constellations into the broader service fabric of 5G or 6G.

To overcome this limitation, the principle of fast re-route has been extended to operate at the level of complete feeder links. By enabling a label swap in MPLS, traffic can be redirected towards an alternative gateway path whenever impairments occur. This mechanism does not aim at masking isolated node or link failures but at addressing the aggregated impact of feeder link variability. In doing so, it adapts a proven concept from terrestrial transport to the particular constraints of NTNs, providing a method to sustain continuity where existing QoS frameworks would otherwise fail.

The introduction of such an extended fast re-route mechanism is of particular value in enabling satellite–terrestrial integration to meet the service requirements associated with future mobile systems. Enhanced Mobile Broadband, mission-critical communication, and emerging low-latency applications all depend on stable end-to-end performance even under adverse atmospheric conditions. By ensuring that traffic can be redirected with minimal disruption when feeder link performance degrades, the mechanism creates a basis for delivering reliable QoS in an environment that has historically been prone to variability.

The remainder of this section is structured as follows. Section 3.3 presents the implementation of the extended fast re-route mechanism within the discrete event simulation environment. Section 3.4 introduces the evaluation setup and methodology. Section 3.5 discusses the obtained results, while Section 3.6 outlines lessons learned and limitations. Section 3.7 concludes with a summary of findings and directions for further development.

3.3 IMPLEMENTATION

Note: The implementation represents an extension of the DES Omnet++ based simulator described in D5.3.

The extended fast re-route mechanism has been realized as an adaptation of MPLS within a discrete event simulation framework. The central idea is to protect not individual nodes or inter-satellite links but entire feeder links, which are subject to sudden capacity variations and outages. To make this possible, MPLS labels are pre-computed for alternative feeder link paths so that an ongoing flow can be redirected with minimal control overhead when impairments occur.

Two approaches for generating alternative paths were considered. The first constructs a complete set of alternative paths for all feeder links in the constellation, ensuring full protection but at the cost of high configuration complexity. The second constructs alternatives only for feeder links within a configurable proximity, thereby reducing the number of redundant paths but requiring additional signaling. In both cases, datasets are generated using shortest-path algorithms and installed into the network in advance.

A key aspect of the implementation is that these label sets are installed for all topology configurations generated by satellite movement. As satellites orbit, a label that leads to a given satellite may remain valid in terms of reachability, but the role of that label changes if the satellite no longer provides feeder link access. In practice, this means that the MPLS infrastructure maintains continuity across topology updates, while the effective meaning of the labels adapts with the availability of feeder connectivity. This distinction is essential for ensuring that rerouting actions remain consistent with the current constellation state.

At runtime, feeder link probing is performed locally at the nodes. When congestion or degradation is detected beyond a configurable threshold, the node initiates a late redirect by swapping the active MPLS label with the pre-configured label of an alternative feeder path. This operation does not require a new global routes computation or state dissemination, which allows reaction times consistent with fast re-route principles. The mechanism supports recovery both from complete outages and from partial capacity reductions, enabling traffic to be shifted away from impaired gateways without interruption of end-to-end service.

3.4 EVALUATION

The evaluation of the extended fast re-route mechanism has been conducted in an OMNeT++-based discrete event simulator, which served as the execution platform for the user plane experiments. The simulated environment models a satellite constellation with dynamic topologies and variable feeder link availability, capturing both the regular orbital movement and weather-induced impairments. Feeder link disruptions were introduced either as complete outages or as capacity reductions, thereby emulating realistic operating conditions.

Traffic generation focused exclusively on the user plane. Measurement streams were established to capture end-to-end latency and throughput evolution during rerouting events. Periodic topology changes were applied in cycles of 50 seconds, representing the dynamics of the moving constellation. A key assumption in this setup is that inter-satellite links provide very large capacity compared to feeder and user links, consistent with 5G/6G NTN architectures. As a result, redirecting traffic across ISLs towards an alternative feeder does not create congestion within the constellation but instead manifests as changes in latency depending on the new path length. All tests were executed in an uncongested environment, ensuring that the observed performance variations are attributable solely to feeder link impairments and rerouting actions.

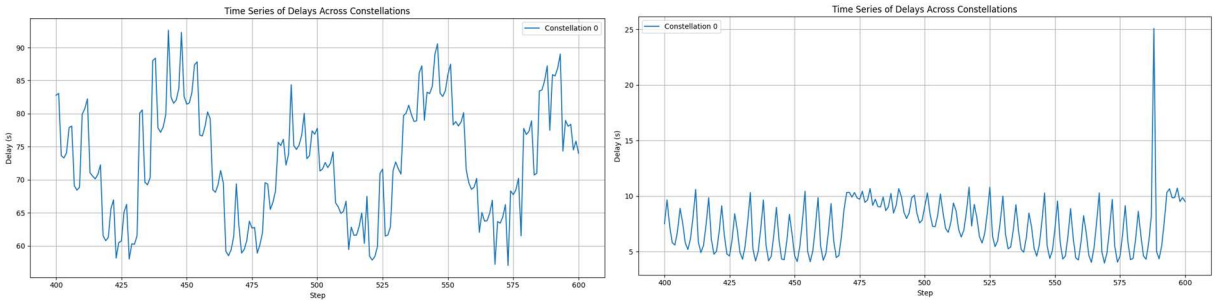


Figure 3-1: End-to-end Simulated data path example (a) between Tokyo and Berlin (b) between Gülpe and Berlin (only between access satellite and feeder satellite, only optical link delay

Figure 3-1 provides a baseline measurement of end-to-end delay in the absence of feeder link failures. Under these conditions, delay remains stable with only minor fluctuations caused by orbital dynamics. This measurement establishes the reference against which impairment scenarios can be assessed. The stability of the baseline confirms that the simulator environment reproduces expected performance when feeder links remain available, and it highlights that the subsequent variations observed in other cases are directly attributable to rerouting actions rather than background effects.

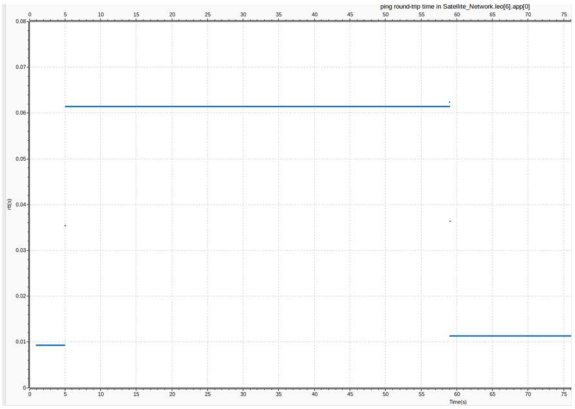


Figure 3-2: RTT in case of a feeder link impair and late redirect

Figure 3-2 illustrates the evolution of end-to-end delay when a feeder link becomes impaired and a late redirect is triggered. During the initial phase, delay is consistent with the baseline as traffic follows the primary feeder. Once congestion is detected, the extended fast re-route mechanism swaps the active MPLS label to an alternative feeder path. This redirection manifests as an increase in the measured delay, corresponding to the longer propagation distance of the backup feeder route. Because inter-satellite links are assumed to provide very large capacity compared to feeder and user links, this redirection does not cause congestion in the constellation, and the impact of the event is confined to delay variation. The continuity of traffic is preserved throughout, with no loss observed.

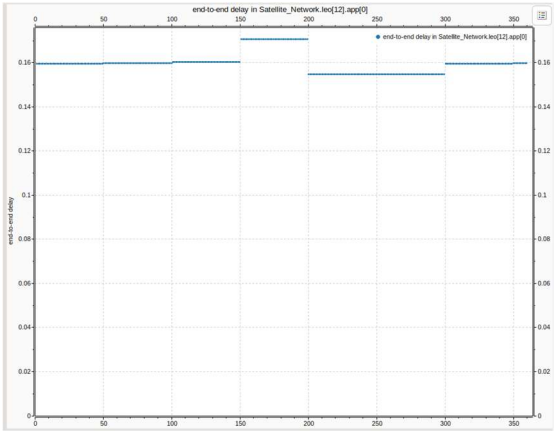


Figure 3-3: Effects on TCP connections of the feeder link change

Figure 3-3 shows the impact of periodic topology changes on end-to-end transport performance. As satellites move and feeder link roles shift, traffic continuity is maintained, though delay distributions broaden with each topology update. These variations reflect the redirection of flows to new gateways as the constellation evolves. Again, due to the uncongested environment and the high-capacity assumption for inter-satellite links, the effects appear as delay variation rather than throughput degradation. The flows adapt almost seamlessly to these changes, demonstrating that extended fast re-route can support user-plane stability even in dynamic orbital conditions.

3.5 CONCLUSION AND NEXT STEPS

This section has presented the first implementation and evaluation of extended fast re-route for feeder link protection in satellite–terrestrial integrated networks. By adapting MPLS to

enable label swapping across alternative gateways, the mechanism extends the protection scope from individual nodes and inter-satellite links to the feeder links that represent the main bottleneck of end-to-end connectivity. The evaluation, performed in a discrete event simulation framework, confirmed that service continuity can be preserved during both feeder link outages and capacity reductions. The results demonstrate that traffic redirection is feasible under dynamic constellation conditions and that the associated performance impact manifests primarily as delay variations due to longer alternative paths.

The findings underline the value of feeder-link-aware QoS mechanisms for future 5G and 6G non-terrestrial networks. Ensuring that services remain available despite adverse atmospheric conditions is critical for the support of broadband, mission-critical, and low-latency applications in satellite systems. At the same time, the evaluation highlights that the late nature of redirect results in non-negligible delay increases, which may pose challenges for applications with strict latency bounds. Furthermore, the current results are limited to small-scale scenarios and uncongested environments and therefore serve as proof of concept rather than as a full characterization of performance.

Future work should focus on several directions. Larger constellations and more diverse traffic patterns should be studied to assess scalability and robustness under realistic load conditions. In addition, the integration of extended fast re-route with higher-layer resource management and scheduling strategies should be investigated to mitigate latency variations associated with redirect events. These steps will allow the concept validated here to evolve into a comprehensive approach for end-to-end QoS management in satellite–terrestrial integrated systems.

4 ON-BOARD NETWORKING CAPABILITIES ANALYSIS

In this section, an analysis is provided of the applicability of the 3GPP 5G Core Network functions, including both control plane and data plane components, as well as the MEC capabilities, when deployed on top of satellite infrastructures. The focus is placed on the instantiation of such functionality as software components executed directly on satellite payloads, thereby transforming spaceborne assets into integral network nodes rather than passive relays. The assessment underlines the feasibility of such deployments in near-term NTN, while simultaneously pointing towards the long-term evolution of 6G architectures where terrestrial and non-terrestrial integration is realized in a unified design.

It has to be noted that the core network functions logically follow the base station layer. As such, the applicability of the following considerations presupposes the presence of on-board gNBs/6G base station, since only under this assumption do the control plane and user plane entities become relevant as part of the payload. Without integrated access functions, satellites remain transparent relays, and the discussion of core instantiation is limited to Earth-based deployments. The analysis therefore explicitly addresses scenarios in which the gNB functionality is already integrated into the space segment, thereby enabling the co-location or distribution of core components in orbit.

Although the terminology and function names used in this analysis follow the established 5G system descriptions as defined by the 3GPP, the conclusions derived here apply directly to 6G deployments. In fact, the space-based realization of 5G network functions provides early insight into the mechanisms required to define a unified NTN–TN architecture, where the separation between terrestrial infrastructures and non-terrestrial extensions becomes increasingly blurred. The implications of this analysis thus go beyond immediate deployment considerations and extend to architectural principles that will determine the shape of future mobile networks.

4.1 SHORT 3GPP CORE NETWORK DESCRIPTION

As illustrated in Figure 4-1, the 5G Core Network adopts a service-based architecture where control and user plane functions interact through standardized interfaces. This architecture was first introduced in 3GPP TS 23.501 and TS 23.502, and it defines a flexible and modular system where functions are no longer bound to fixed nodes but are instead instantiated as software-driven entities discoverable through service registration. The resulting design provides extensibility and network slicing support, yet at the same time creates strong interdependencies among the functions that must be considered carefully in non-terrestrial deployments.

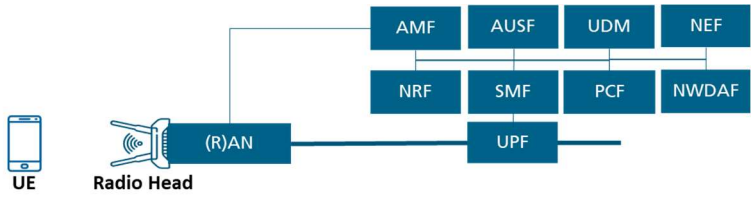


Figure 4-1: Simplified 3GPP Core Network Architecture

At the point of access, the RAN provides the connection between the UE and the core. Control plane signalling flows initially towards the Access and Mobility Management Function (AMF), which is responsible for registration, connection handling, and mobility anchoring. The AMF

communicates with the Authentication Server Function (AUSF) to validate subscriber credentials and with the Unified Data Management (UDM) to retrieve subscription data. Through these interactions, the network ensures that only authenticated and authorized devices obtain access.

The Session Management Function (SMF) handles the lifecycle of PDU sessions, which includes session establishment, modification, and termination. It instructs the UPF on how to route traffic, and it coordinates with the Policy Control Function (PCF) to apply quality-of-service rules and charging policies. The Network Repository Function (NRF) maintains a dynamic catalogue of available network functions, enabling service discovery and selection. The Network Exposure Function (NEF) offers an interface towards external applications and facilitates controlled service exposure, while the NWDAF collects operational data and provides analytics to optimize network behaviour.

The UPF is the termination point for these control interactions. It acts as the anchor between the access network and external data networks, and it implements the actual forwarding of user traffic according to the rules and parameters dictated by the SMF and PCF. In this sense, the UPF is not an isolated forwarding entity but rather the endpoint where control plane instructions are enforced as user plane behaviour. The fact that every session establishment, policy change, or mobility event requires an updated instruction to the UPF underlines its role as the final element in the control chain.

This arrangement has significant consequences when mapping the 5G Core to satellite infrastructures. If control plane functions such as the AMF or SMF are deployed on-board satellites while the UPF remains located on the ground, almost every procedure will result in repeated signaling exchanges across the feeder link. Registration, session establishment, mobility updates, and even periodic policy modifications will traverse the feeder link, adding delay and consuming scarce capacity. Conversely, if the control plane remains on the ground while gNBs are in orbit, the dependency is preserved, as every UE interaction still involves back-and-forth control signaling over the feeder link. The service-based architecture, while modular, was defined under terrestrial latency assumptions, and splitting functions across the feeder link therefore creates inefficiencies and fragility that must be explicitly addressed in non-terrestrial network designs.

4.2 CONTROL PLANE NETWORKING CAPABILITIES

The control plane of the 3GPP Packet Core is responsible for access authorization, session establishment, mobility management, and policy enforcement, and its placement fundamentally determines both service performance and architectural resilience. When transposed to satellite payloads acting as spaceborne network nodes, the deployment of such functionality introduces new design considerations due to orbital dynamics, constrained onboard resources, and inter-satellite connectivity requirements.

In accordance with 3GPP TS 23.501, the analysis begins by retaining the well-established network function terminology, including the AMF, the SMF, and associated service-based interfaces. However, as will be argued throughout this subsection, the rigid notion of function-specific instantiation is progressively being replaced in 6G visions by higher-level service abstractions, such as “mobility management service” or “session management service.” This change reflects a conceptual shift away from static function boxes towards a service-oriented control plane in which the required functionality can be instantiated flexibly and co-located as dictated by operational constraints.

4.2.1 Few Split Models Make Sense Demonstration

When assessing the placement of control plane functions in space, the most immediate question is whether it is advantageous to distribute them between satellite payloads and ground infrastructure. At first glance, such a split may appear to reduce onboard resource requirements while still allowing limited autonomy in orbit. However, when examining the actual control plane procedures defined in 3GPP TS 23.502, it becomes evident that almost every procedure ultimately requires interaction with the UDM, which for reasons of scale, security, and integration is expected to remain located in terrestrial facilities.

During registration, the AMF exchanges authentication vectors with the AUSF and retrieves subscription profiles from the UDM. Even if the AMF were instantiated on the satellite, the procedure could not be completed locally; the signaling would still traverse the feeder link to reach the ground-based UDM. Similarly, in PDU session establishment, the SMF requests subscription data and policy rules, which again involve UDM queries and often PCF interactions. The feeder link thus becomes a mandatory leg of the procedure, irrespective of where the SMF is placed.

A further consideration is that the AMF, the SMF, and the PCF (if deployed) are present in almost all user-related procedures once a device is registered. Registration, service request, session management, mobility update, handover, and policy reconfiguration all invoke one or more of these functions. They form the operational triad of the control plane: the AMF anchors access and mobility, the SMF governs sessions and user plane bindings, and the PCF dictates policies and charging rules. Consequently, placing any of these functions on a satellite while leaving the others on the ground results in constant cross-feeder signaling, since each procedure spans multiple functions that are interdependent by design.

The same pattern repeats in mobility updates and handover procedures. Mobility context and session continuity information are validated against subscription and policy data, and these data sets reside in the UDM. Even optimizations such as local co-location of AMF and SMF in orbit do not remove dependency, as the functions require authoritative data and authentication credentials from the terrestrial domain. Finally, policy and charging control depends on PCF interaction, which in turn queries subscriber data from the UDM. The result is that almost every stateful operation in the control plane follows a signaling path that extends all the way to the ground.

This observation demonstrates that splitting network functions between space and ground provides little gain but introduces multiple disadvantages. Each feeder link traversal adds delay, consumes scarce spectrum resources, and increases the likelihood of disruption due to link variability. Furthermore, the fragmentation of control plane logic complicates fault management and resilience, since failures in either domain can stall end-to-end procedures. As such, rather than separating control plane functions across the feeder link, architectural effort should concentrate on co-locating tightly coupled functions in orbit and minimizing the volume of mandatory interactions with the UDM through caching, replication, or future distributed data layer designs. Without such adaptation, a space–ground split of control plane functions does not deliver meaningful benefits and risks undermining the reliability of the overall system.

4.2.2 Split Models

Because of the requirement to co-locate functions, or at least to run them in the same plane, as demonstrated in the previous section, it is evident that only a few split models remain feasible. The interdependencies between AMF, SMF, PCF, and the UDM ensure that nearly all registered user procedures inevitably extend towards the ground-based UDM. As a result, distributing functions across the feeder link does not eliminate signalling dependency but

instead increases latency, complexity, and orchestration risk. Figure 4-2 illustrates four representative configurations.

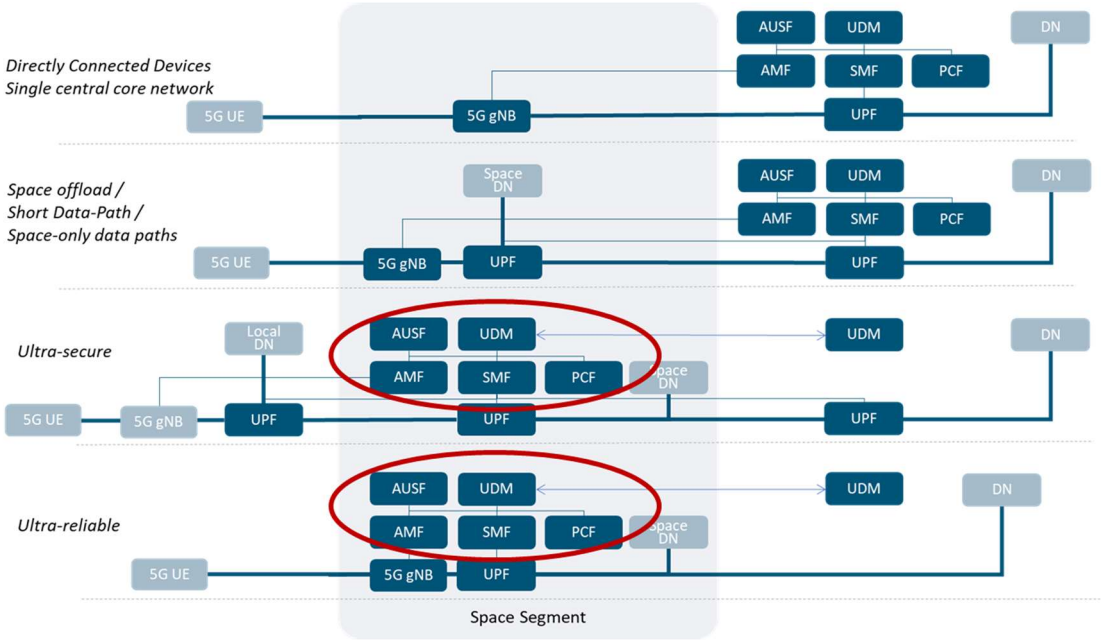


Figure 4-2: Functionality Split Models (with integrated space control plane)

Ground Core model. The gNB is on the satellite while all core functions remain on Earth, which is equivalent to a regenerative payload with an on-board gNB and a ground core network. This minimizes spaceborne complexity but forces every control and user procedure to traverse the feeder link, resulting in high latency and signaling overhead.

Space User Plane model. The gNB is co-located with a UPF in orbit while the control functions stay on the ground. This improves user-plane latency for local traffic, enabling low-delay services at the edge of the constellation. Control-plane dependence on the ground remains, but the model is attractive because of its simplicity: it requires no redesign of core functions, only an additional UPF instance near the access. Interoperability with terrestrial operators is straightforward, as the control plane is left unmodified.

Split Core model. AMF, SMF, and PCF are instantiated on-board together with a UPF, while UDM and AUSF remain on Earth. In the baseline form, every subscription retrieval and authentication still requires feeder link signaling. However, if the UDM is logically split, with the active state of registered users maintained in orbit while the full subscriber profile remains on the ground, many procedures can be executed locally without involving the feeder link. This makes the model more efficient for mobility and session continuity, while still relying on the ground UDM for provisioning and subscriber lifecycle management. It retains good interoperability with terrestrial operators, though reconciliation mechanisms for the distributed UDM state must be carefully defined.

Space Core model with dual UDM. AMF, SMF, PCF, AUSF, and UPF are moved into the space segment, and a UDM is deployed on-board that is correlated with a UDM centralized on Earth. The ground UDM is used for subscriber management operations such as onboarding and lifecycle updates, while the space UDM supports all operational procedures. This reduces inter-satellite signalling and enables high autonomy, but the price is a more complex synchronization framework and a more difficult integration with terrestrial operators. This

model is best suited for ultra-robust, special-purpose networks where independence from the ground is a priority.

The comparison underlines that different models align with different objectives. The Space User Plane model offers immediate latency reduction without architectural redesign. The Split Core model, especially with a distributed UDM, achieves partial autonomy in orbit and can handle frequent mobility with reduced feeder link load, while remaining compatible with terrestrial networks. The Space Core model, although powerful, is mainly justified for special ultra-robust deployments where ground dependency must be minimized. No single model can be selected unconditionally, but the choice reflects the balance between simplicity, autonomy, and interoperability.

4.2.3 Space-Core Optimizations

For scenarios in which the core network is deployed in the space segment, the placement of individual functions on different satellites introduces unnecessary signalling exchanges, elongated decision paths, and increased interface complexity. The service-based architecture of the 5G core was designed under terrestrial latency assumptions, where the overhead of additional interfaces is acceptable. In orbit, however, each interface crossing becomes more costly, both in terms of delay and reliability.

Co-locating tightly coupled functions on the same satellite mitigates these challenges. When AMF, SMF, PCF, and related components operate in close proximity, control decisions can be executed with minimal signalling, reducing the number of inter-satellite transactions and shortening the time between a triggering event and its resolution. This consolidation also simplifies failure management, since the dependencies between functions are resolved locally rather than across the constellation.

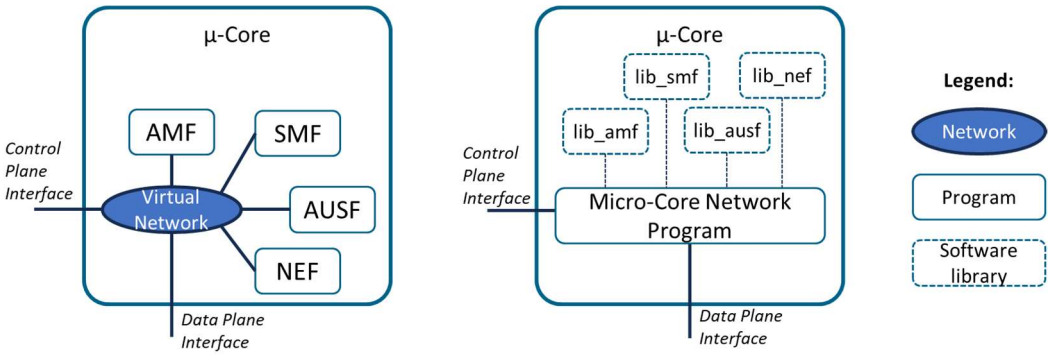


Figure 4-3: Micro-cores concept as introduced in: Corici, M., Zope, H., Magedanz, T., Kurata, M., & Suzuki, M. (2024, October). Micro Core Networks Assessment in the Cloud-Native Beyond 5G/6G Mobile Network Operator Deployments Context. In 2024 3rd International Conference on 6G Networking (6GNet) (pp. 196-200). IEEE.

As illustrated in Figure 4-3 , this principle can be realized through the deployment of micro-cores, which combine the essential control plane functions into a single logical entity per satellite. In this sense, it is preferable to instantiate complete micro-cores on multiple satellites rather than to distribute individual functions such as AMF and SMF across different nodes. Each micro-core can serve the users in its coverage area with full local decision capability, while higher-level coordination between satellites is limited to synchronization and state distribution. This reduces the signalling burden on inter-satellite links, preserves procedural integrity, and makes the system more robust against partial failures.

The figure shows two implementation approaches. On the left-hand side, each function is instantiated as an individual software component connected through a virtual network. This design mirrors the service-based architecture, preserving modularity and alignment with existing 3GPP frameworks, though it still incurs internal messaging overhead. On the right-hand side, the micro-core is consolidated into a single program in which the functional logic is provided as libraries. In this variant, service calls become direct library invocations, significantly reducing messaging overhead and resource consumption while accelerating decision-making. Both variants offer the same functional scope but represent different trade-offs between modularity and efficiency.

The co-location principle thus emerges as a guiding design rule: distributing complete micro-cores across multiple satellites provides the balance between local autonomy and constellation-wide coordination, while the choice of implementation determines whether modular transparency or computational compactness is prioritized.

While co-location of functions into micro-cores removes unnecessary signaling between satellites, it does not eliminate the need for shared subscriber state across the constellation. Without state synchronization, each micro-core would operate in isolation, which would prevent seamless mobility and session continuity. The challenge therefore shifts from functional splitting to state distribution.

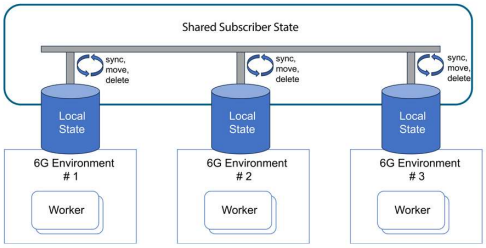


Figure 4-4: Subscriber State Sharing as implemented in Fraunhofer Open6GCore

As illustrated in Figure 4-4, each 6G environment hosts a local state repository tightly coupled with its micro-core. Subscriber context, session information, and mobility anchors are maintained locally to allow low-latency decision-making. At the same time, these local states are connected to a shared subscriber state layer, where synchronization operations propagate changes across the constellation. Typical operations include synchronization of updated session parameters, migration of active state during handovers, and deletion of obsolete records to free resources.

This design ensures that while each micro-core is capable of operating autonomously, the constellation as a whole maintains a consistent view of subscriber data. Workers in each environment can execute requests immediately against the local state, while background processes handle the synchronization, movement, or deletion of records. The effect is a balance between responsiveness and consistency: user procedures remain fast and local, while the shared state layer preserves service continuity as devices move between coverage areas.

The implication is that co-location of functions and distribution of micro-cores across satellites is only viable when supported by robust state synchronization. Without such a mechanism, the benefit of local decision-making is undermined by the inability to guarantee global consistency. State distribution therefore becomes the enabling foundation for ultra-reliable spaceborne networks, defining how local autonomy integrates into a coherent multi-satellite architecture.

4.2.4 Handover Configurations

For mega-constellations where the RAN is integrated directly on-board the satellites, the handover problem takes a distinct form. Unlike terrestrial deployments, where mobility is driven primarily by user movement, here the satellites themselves are mobile, and the handover sequence follows the predictable dynamics of orbital motion. This property makes it possible to replace complex multi-phase procedures with self-organizing mechanisms that exploit trajectory predictability.

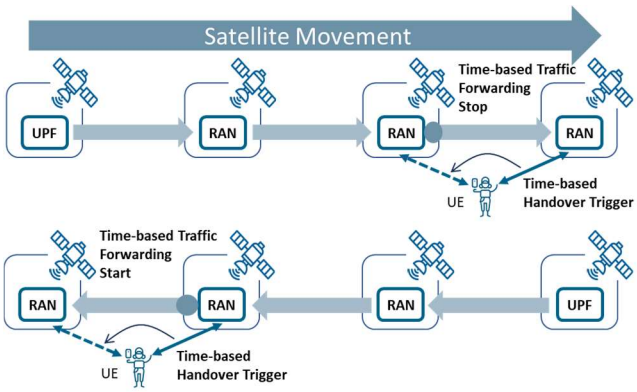


Figure 4-5: Data Path Strip Concept

As illustrated in Figure 4-5, satellites in a given orbital plane pass one another in a deterministic sequence, allowing handovers to be scheduled in advance. The UE can therefore reselect the next serving RAN autonomously according to network policy, using time-based conditional handovers. This eliminates the need for explicit handover commands from the network and removes the preparation phase defined in conventional 3GPP procedures. Instead of tunnelling buffered data from the source to the target RAN, buffering can take place directly at the target RAN in anticipation of the transition.

To support this approach, the concept of a data path strip is introduced. This is a continuous data path along the same orbital plane that links the UPF with the currently serving RAN. During a handover, the redirection of the downlink data within the strip can be triggered automatically at the scheduled time, ensuring continuity without additional signalling. Two situations are depicted in Figure 4-5. In situation A, where the handover is towards the UPF, the target RAN begins buffering data at the appropriate time and stops forwarding to the source RAN. In situation B, where the handover is away from the UPF, the source RAN extends the data path to include the next satellite, delivering the buffered data directly to the target RAN. In both cases, the preparation and execution phases collapsed into a forwarding adjustment rather than a signalling exchange.

Upon completion of the handover, a minimal “Handover Complete” message is still required to update the AMF with the new UE location. This preserves network consistency without reintroducing the complexity of the traditional procedure. The scheme prioritizes reselection within the same orbital plane, which is typically sufficient, though inter-plane handovers remain possible. In such cases, the data path strip must be reconfigured rapidly to anchor traffic to the correct orbital plane.

If the UPF is fixed to a ground station or to a specific satellite, the same mechanism applies, with the strip reconfigured at each transition to reflect the change in the RAN endpoint. This self-organizing reselection of both RAN and data path reduces signalling overhead and node processing load, while fully exploiting the deterministic nature of satellite mobility. Should an

unexpected condition prevent the automated transition, the system reverts to the standard 3GPP handover procedure as a fallback, ensuring robustness.

4.3 USER PLANE OPTIONS AND EDGE COMPUTING

4.3.1 UPF Placement

A UPF placed in space is itself a form of shortcut, since it prevents every packet from traversing the feeder link to a terrestrial anchor. However, a full UPF in orbit brings extensive functionality, including charging, lawful interception, and policy enforcement, which are resource-intensive and may exceed the capabilities of satellite payloads. To address this, the UPF can be split into two components: a lightweight forwarding element in space that provides the shortcut capability, and a ground-based anchor UPF that maintains the full set of administrative and regulatory functions. This separation confines complexity to the ground while still reaping the benefits of reduced data path length in the constellation. A persistent challenge in non-terrestrial networks is the elongation of the data path caused by the traversal of multiple relay satellites and distant UPFs anchored in terrestrial sites. Even when user UEs are located within the same coverage domain, the default forwarding model forces their packets to traverse a fixed anchor UPF, which introduces substantial delay and duplicates processing functions such as charging, accounting, and lawful interception. The shortcut concept, originally introduced for beyond-5G terrestrial systems, can be extended to NTN to reduce these delays while preserving control-plane consistency.

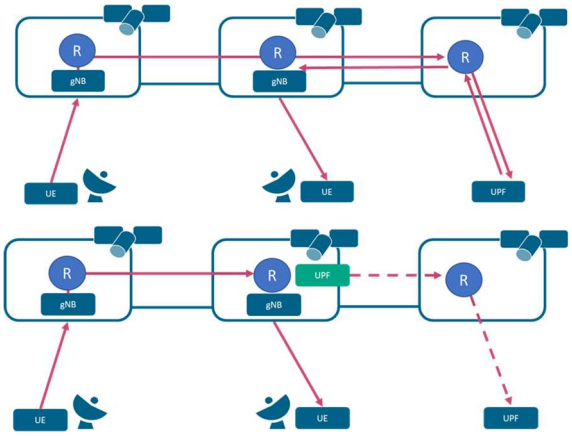


Figure 4-6: Data Path Shortcut Concept applied to NTN

As illustrated in Figure 4-6, shortcuts allow a more direct path between satellites and UPFs bypassing unnecessary anchor traversals. The principle is simple: when uplink and downlink traffic between two UEs or between a UE and an application service can be resolved through a shorter route, an intermediary UPF inserts the packets directly into the downlink path, while a single asynchronous copy of the data is sent to the anchor UPF to fulfill administrative and lawful obligations. This avoids double traversal of the anchor without losing the ability to account for and intercept traffic.

Two cases are shown. In the upper scenario, UE traffic follows the conventional route, with each satellite forwarding towards a fixed UPF anchor, adding multiple relay hops and significant latency. In the lower scenario, a shortcut is established at an intermediary UPF located closer to the UE, enabling direct redirection of traffic along the minimal path. The result is a shortened end-to-end delay, fewer packet encodings and decodings, and a reduced load

on inter-satellite links. Importantly, the original anchor UPF still receives a mirrored stream asynchronously, ensuring consistency with standard core network obligations.

In NTN deployments, shortcuts are particularly beneficial because predictable satellite orbits and coverage patterns make the determination of shortcut-eligible paths straightforward. When two UEs are within the same orbital plane or served by satellites with established inter-satellite links, shortcut establishment can be triggered by the SMF without complex signaling. Should the topology change, the shortcut can be transformed into a longer two-UPF configuration or reverted to the standard anchor-based path, preserving session continuity.

The main gain of shortcuts in NTN is robustness, by avoiding repeated use of the feeder link, and predictability, by containing latency within the orbital segment. When UPFs are anchored on Earth, shortcuts mitigate the delay but cannot fully prevent feeder link dependence. When UPFs are instantiated in orbit, shortcuts confine the data path to the constellation, ensuring that even under mobility events the delay remains bounded. The trade-off lies in complexity: maintaining asynchronous copies at the anchor UPF requires additional signaling and buffering, while a split deployment of UPFs (space and ground) raises questions of consistency, lawful compliance, and resource overhead. Shortcuts therefore shift the design balance from purely minimizing delays to achieving a controlled, robust latency envelope while reducing unnecessary feeder link usage.

4.3.2 Multi-Access Edge Computing for NTN

Enabled by the placement of an UPF at the edge as part of space payload, MEC services can be provided creating a Local Area Data Network (LADN). The main purpose of is to provide low latency and ultra reliable services. For this processing and storage capabilities at the edge are used to provide these services close to the user decreasing the transmission time. Typically, MEC services are enabled by an edge platform addressable by web-services.

Given the NTN architecture from D3.2, there are different possibilities to host edge services. For bigger vehicles, such as airplanes, part of the edge could be hosted on-board as it is already done today e.g. for entertainment systems. Or they are hosted at the ground network, with the benefit is that more powerful hardware can be deployed, or ground nodes are deployed close to processing centres providing almost unlimited processing power. Last option is to deploy MEC functions on-board the satellites. As mentioned, the resources here are limited, but it is expected that they increase in future. The main benefit of hosting it on-board satellite is that the transmission delay is at least half of the transmission time needed since a hop to the ground network can be spared. Given that MEC is mainly about low latency, and in NTN this is mainly determined by the orbit, LEOs are expected to be used and provide MEC services. Still even for GEOs half of the transmission time can be saved which can be around half a second. Figure 4-7 shows the options on MEC placement for an NTN connection.

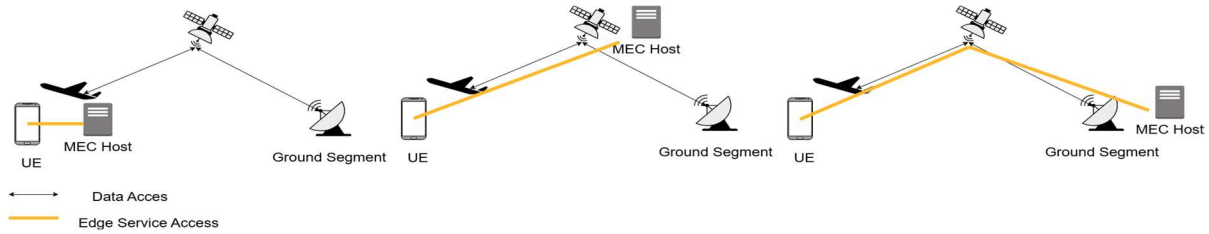


Figure 4-7: MEC placement in an NTN link

However, for LEO constellations the problem of context migration or state transfer arises. If MEC services store user data or MEC platform have a dedicated set of services and configurations this state needs to be transferred. For NTN two possibilities are defined: (I) fixed

cells, where a cell grid on ground is defined and the serving satellite is changing, and (II) moving cell, where each satellite has its own cell, basically overlapping with its footprint, and this cell is moving with the satellite movement over ground. In the first case, the state corresponds to a fixed cell on ground, if there is a hand over to another satellite the state must be transferred. For the moving cells there are two options, the satellite keeps its own state, i.e. state data is moving with the satellite, or it is also transferred to a second satellite. In this case the process can be complex, since state data might be linked to a region or a specific user. It might be that both do not necessarily overlap again, and might be connected by different satellites. The transfer process should follow the hand-over to ensure that the same satellite serving the user has also the state data.

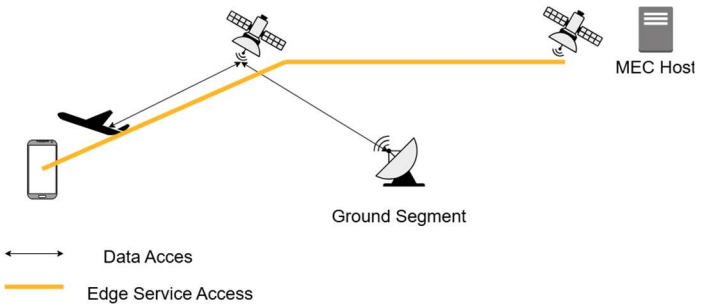


Figure 4-8: MEC placement in constellation

Figure 4-9 shows the high-level architecture for a MEC platform on-board a satellite. The UE connects to the satellite which is linked to the ground network. On-board the satellite the full gNB needs to be located together with an UPF to make the MEC platform accessible via webservices. The UPF is connected to an anchor UPF at the core which is located on ground after the NTN ground segment. This also means that the feeder link is connecting the UPF on-board with the UPF at the core and with this is part of the core connection and is IP based. The satellite is hosting a RAN+MEC node.

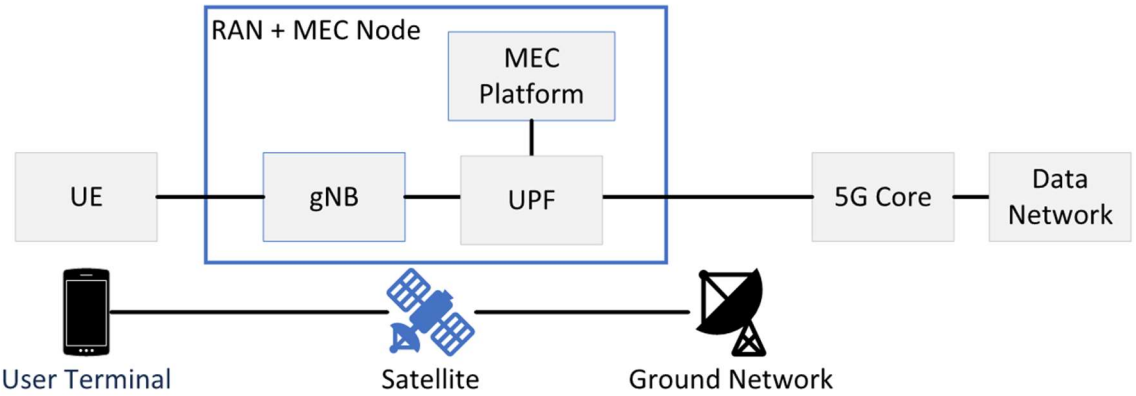


Figure 4-9: MEC platform on-board satellite [21]

4.3.2.1 MEC Standardization Review

As noted in [21], the edge computing concept has been standardized by 3GPP and by ETSI. 3GPP specified a MEC enabled architecture for 3GPP systems, while also looking into edge computing for NTN. ETSI started with MEC for mobile networks but took a more general way

specifying interfaces for MEC apps, MEC app developers and system operators of not mobile systems but also fixed and WiFi. But NTN remains an area to be addressed.

3GPP TS 23.558 provides a comprehensive framework for MEC in 5G networks, outlining the architecture, requirements, and use cases for MEC-enabled services. The architecture for enabling edge application is shown in Figure 4-10. Key functional entities are:

- Edge Enabler Client (EEC): Resides in the UE; handles client application (AC) profiles, requests services, and supports service continuity.
- Edge Enabler Server (EES): Provides support for Edge Application Servers (EAS) and EECs; includes configuration provisioning, API exposure, 3GPP Core Network interaction, and Application Context Relocation (ACR) support.
- Edge Application Server (EAS): Hosts edge applications and serves Application Clients (ACs).
- Edge Configuration Server (ECS): Manages service provisioning, provides edge configuration data to EECs, and handles EES registrations.
- Cloud Application Server (CAS): An application server in a cloud data network, outside the Edge Data Network (EDN).
- Cloud Enabler Server (CES): Supports CASs and facilitates interworking for service continuity.
- ECS with Repository function (ECS-ER): An ECS capable of storing edge deployment information from other ECSs and common EAS information.

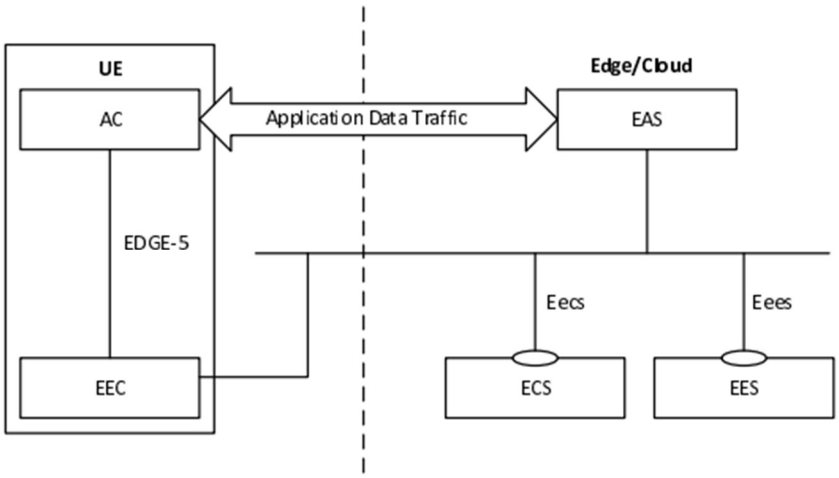


Figure 4-10: 3GPP architecture for enabling edge applications [22]

On the other hand, ETSI defined the architecture presented in Figure 4-11 [23]. The federated architecture is shown. Federation is used to connect the MEC system of provider A with a MEC system of provider B which is very interesting for NTN-MEC since it can be used to connect a MEC system of an SNO with this of an MNO. ETSI specified dedicated APIs for this.

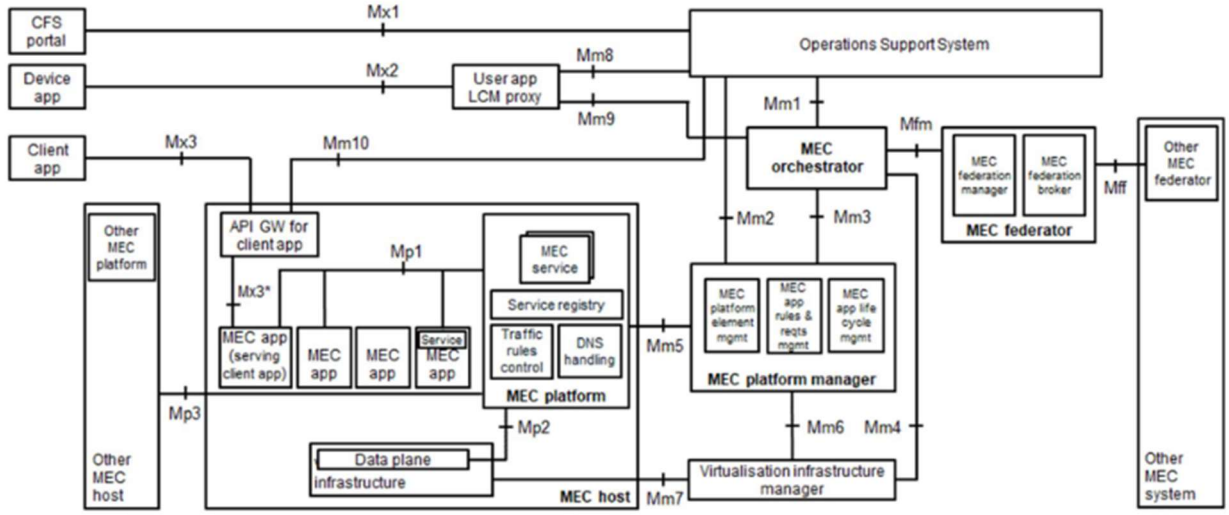


Figure 4-11: ETSI MEC Federated Architecture [23]

The most important elements of the MEC architecture are:

- The MEC host provides the resources to host for the MEC-Platform and MEC apps and has a connection to the data plane, e.g. for radio network information service (RNIS)
- The MEC platform is the central component of the MEC host. It provides services registry and discovery features, has an interface for the data plane and the MEC apps. Furthermore, it can connect to other MEC hosts, e.g. for state transfer or exchanging user information
- MEC platform Manager and Orchestrator allows us to manage the MEC system from operator side, i.e. perform life cycle management of the platform itself, but also the apps and generally to interface the virtual infrastructure.
- Client apps on UE side can access the MEC apps i.e. the API Gateway (GW) via webservices.

For the other components, it should be referred to the standard document. ETSI harmonized their work with 3GPP and provided a synergized architecture presented in Figure 4-12 [30]. In this, the Radio System part is mainly taken from 3GPP and the management and orchestration from ETSI. For the overlapping components that are on MEC host side, a dual implementation is considered which needs an alignment of the EDGE-3 and Mp1 interfaces.

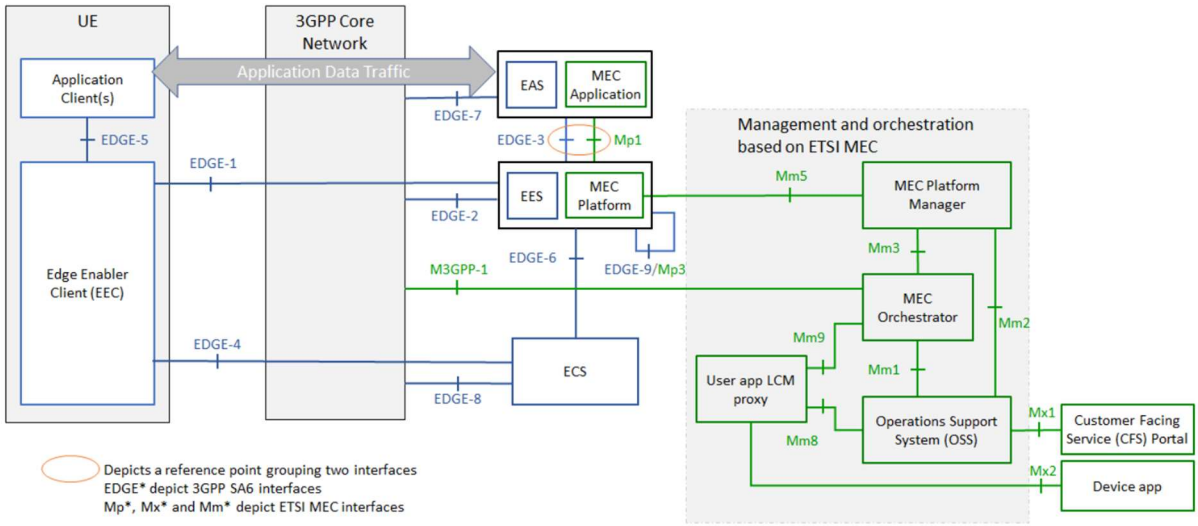


Figure 4-12: ETSI MEC ISG harmonized architecture [30]

3GPP studied in Release 19 [31] on extending 5G MEC capabilities to NTN. The study on MEC investigates deploying edge computing components, including EES and EAS, onboard satellites (both GEO and NGSO), while keeping the ECS on the ground. The existing EDGEAPP architecture is reused without architectural impacts, with enhancements focused on optimizing EAS discovery and service provisioning for mobile satellites. To support satellite mobility, EAS and EES profiles are extended with parameters such as “Trajectory ID” and “Satellite Assistant Information,” which aid in selecting suitable EES/EAS instances and improving service continuity by aligning user movement with satellite orbit paths.

4.3.3 MEC for NTN: Platform and MEC Services

In order to provide a MEC Platform for NTN services, two existing system could be used that fit the needs of a research project. First is light edge (available at [32]) using srsRAN [45]. However, this tool is not supported anymore by the developers and is still based on 4G services. The second option is the OpenAirInterface Orchestration Suite (available at [33]) that includes a MEC platform and an RNIS service for data plane feedback. RNIS monitors the current data plane performance and makes it accessible via web-service to MEC-apps. It is a core service for MEC-apps dealing with Quality of Service, especially in the low-latency use cases MEC typically is used for. The MEC platform offers MEC-app registry services as specified by ETSI.

Figure 4-13 shows the open box perspective of a RAN and MEC node based on the OpenAirInterface setup. The UE connects to the gNB which connects to the UPF on-board. As mentioned, the UPF on-board is connected to an anchor UPF at the core. A REST gateway makes the services accessible and interconnects them. The MEC services include the RNIS services, the MEC platform services and potential other services in the MEC App block. The RNIS services, as described, needs access to the data plane, which is implemented by the means of the RIC. A dedicated RNIS xApp is designed to read the necessary user information from the RIC. On the other hand, the RIC gets this information from the gNB via the E2 interface. In order to fully support the RNIS services, some information from the 5GC is required which is provided via a configuration manager. The configuration manager subscribes to AMF and SMF information at the core. Information from both, the xApp and configuration manager, is matched at the RNIS web services to provide lower layer connection information for each connected user.

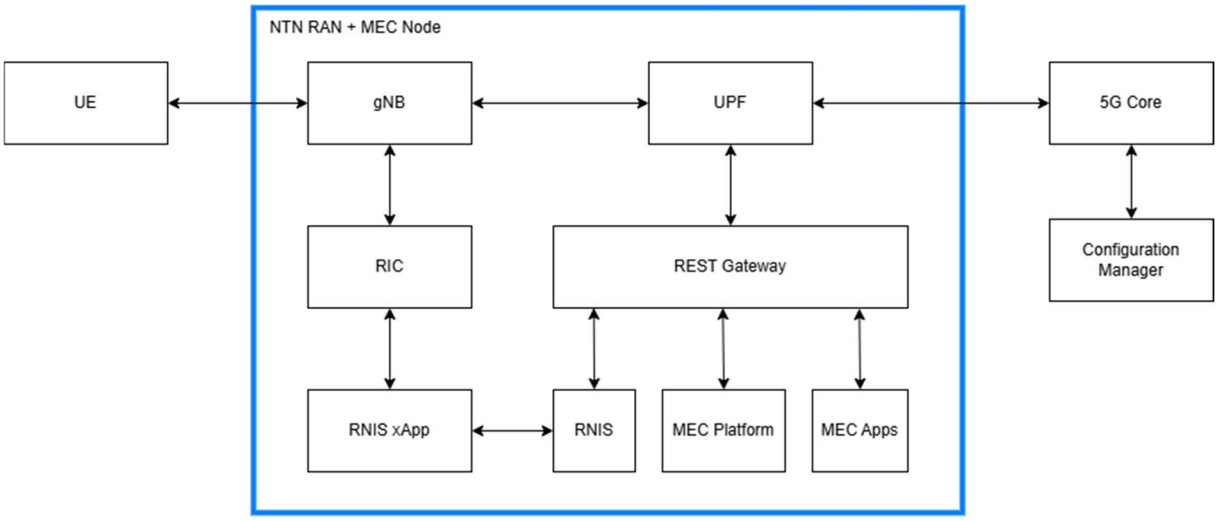


Figure 4-13: open box diagram RAN + MEC node based on OpenAirInterface

The information read from the xApp on the E2 interface is presented in Table 4-1.

Table 4-1: RIC RNIS parameters on E2 Interface

Parameter	**Description**
RSRP	Reference Signal Received Power
MCS_ul	Modulation and Coding Scheme for Uplink
MCS_dl	Modulation and Coding Scheme for Downlink
BLER_ul	Block Error Rate for Uplink
BLER_dl	Block Error Rate for Downlink
ul_aggr_bytes_sdus	Uplink Aggregate Bytes for Service Data Units
dl_aggr_bytes_sdus	Downlink Aggregate Bytes for Service Data Units
PHR	Physical Hybrid Automatic Repeat Request Feedback
SNR	Signal to Noise Ratio

Summarizing the shortly the inputs from WP4 on the possible hardware configuration at today's satellite, state of the art is:

- 4 CPU (Multithread)
- 128-254 GB RAM
- 1-8TB storage (no SSD)

For comparison and assessing the feasibility of such an MEC platform we summary some requirements found in the following:

- OpenAirInterface – RNIS [34]
    - Minimum 4 Core CPU (better 8)
    - 16 GB RAM
    - Docker version > 22
- Intel Smart Edge [35]
    - 2 Core CPU
    - 8 GB RAM
    - 50 GB Storage (SSD)
    - OS: Ubuntu* 20.04 LTS
- OpenNetwork Foundation Cord
    - 4 Core CPU
    - 32 GB RAM
    - 100GB Storage
    - Recommended: 2x Intel E5-2630 v4 10C 2.2GHz 85W, 64GB of RAM 2133MHz DDR4, 2x 500GB HDD

On a first intuitive comparison, storage and memory are supported, besides that SSD storage is not available which slows down the speed needed for read and write access. The bottleneck might be the number of processors needed. Given that the resources have to be shared between the MEC system and communication system, i.e. the gNB, the processing capabilities might be limited. In cooperation with WP4 the suggested solution is to have a flexible payload that either uses the available resources in the following way:

- RAN Satellite (100% resources radio link)
- RAN + MEC Satellite (50/50 resources radio link/MEC)
- MEC Satellite (100% resources MEC)

This could be orchestrated depending on the demands over a certain region. As mentioned, idle satellites (e.g. above ocean area) could be connected as MEC satellites via inter satellite links to benefit from these unused resources. As mentioned, in the mid to long term it is expected that more processing power will be available and that both services could be offered simultaneously without limitations. Alternatively, as looked in other European Projects addressing 6G systems, e.g. 6GNTN [36], different layers of satellites could be combined with different purpose satellites, e.g. a layer for communication and a layer for MEC.

But generally, the requirements of the system depend on the number of supported users, and requirements of the MEC apps that need to be hosted. Some classification on the MEC-Apps helps to estimate the technical challenges:

- Stateful / State-less: Whether a MEC-app is stateful or state-less defines if this state needs to be transferred. If it is needed in best case, inter-satellite links are used or the state is transferred via the ground segment to the next satellite, the time for this transfer needs to be considered. For mobile users it needs to be considered that a user can switch the cell, as in terrestrial systems.

- RNIS / non-RNIS: if a MEC-platform shall support RNIS services a link to the data plane is needed as illustrated in Figure 4-13. This means that in addition to the requirements from MEC platform and the gNB, the requirements of the RIC need to be fulfilled.

Early MEC platforms for NTN might first support state-less services without RNIS. With increased capabilities in mid- and long term, more demanding services could be provided, e.g. adding states or RNIS. The more capabilities are offered, the more services, and for a higher number of users can be offered. Some examples of use cases that are either more related to storage or processing are:

- Data access: caching, e.g. management of on-board entertainment systems, or file sharing for disaster management, localization services

- Processing power: video surveillance/hazard detection, image processing, autonomous/remote controlled systems, IoT check relevance of data, AR/VR, Connected vehicles

4.3.4 AI-Driven Onboard MEC Services

As part of the 5G-STARDUST project's Task 5.5, this analysis addresses the challenge of defining optimal on-board networking capabilities for the Non-Terrestrial Network (NTN) segment. The core methodology of the task is to execute a comprehensive trade-off analysis by decomposing the project's reference architecture from WP3 into a set of chained micro-services for both the control and data planes [24]. These different "split models", which determine where each micro-service is placed, are then evaluated against the service requirements from WP2 and the inherent constraints of limited satellite payload resources.

While architectural splits of core Control Plane (CP) functions can often be evaluated with static models, the optimal placement of User Plane (UP) and, particularly, Multi-access Edge Computing (MEC) functions is highly dynamic. This dynamism is driven by real-time factors such as fluctuating user demand, the constant movement of the LEO satellite constellation, and changing network conditions. The 5G-STARDUST architecture's use of regenerative payloads, which allows network functions to be hosted directly on satellites, transforms the constellation into a space-based edge cloud, creating a powerful but complex resource management challenge [31].

Therefore, this section focuses specifically on solving the MEC task placement problem by applying a Deep Reinforcement Learning (DRL) framework. We propose an intelligent system capable of determining the optimal dynamic split model for data-plane and MEC services in real-time. This approach directly addresses the T5.5 goal of evaluating functional splits by providing a mechanism to find the best placement adaptively, rather than relying on a fixed configuration. This analysis evaluates how the decomposition of 5G network functions into fine-grained micro-services can be exploited to select the optimal on-board placement of tasks, thereby meeting the diverse service requirements and Key Performance Indicators (KPIs) identified in D2.1.

To solve this complex placement problem, the DRL agent's objective is to learn a policy that minimizes a cost function representing a weighted combination of end-to-end latency and satellite energy consumption. The agent is trained and evaluated in a simulated environment that models the reference LEO constellation defined in D3.2. The performance of this AI-driven policy is then compared against benchmark strategies, including a fully ground-based (non-split) processing model and a fixed full-onboard processing (static split) model, to quantify its benefits.

4.3.4.1 System Model

The model is based on a heterogeneous network architecture that integrates a Low Earth Orbit (LEO) satellite constellation with a terrestrial Cloud Computing facility. The system's behavior is analyzed in a discrete-time framework, with network parameters assumed to remain constant over each time interval, τ_i. This discrete-time approach allows us to model the dynamic nature of the satellite constellation and the time-varying user demands. The core components of the system are as follows:

- **User Population:** We consider a set of users, $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$, randomly distributed within a specified rectangular area. Each user u is considered active in every time interval with a defined probability. An active user generates a computation task request $\rho_m(\tau_i)$, which is formally defined by a tuple $\langle D_{\rho_m}, D^r_{\rho_m}, \Omega_{\rho_m}, T_{\rho_m} \rangle$. This tuple specifies a task with an input data size of D_{ρ_m} Bytes, an expected output data size of $D^r_{\rho_m}$ Bytes, requiring Ω_{ρ_m} CPU execution cycles, and having a maximum permitted execution latency of T_{ρ_m}

- **Satellite Constellation:** The system model assumes a LEO constellation with regenerative payloads, consistent with the reference architecture in D3.2. The on-board processing capabilities are modeled to be representative of a satellite hosting a full gNB or at least a gNB-DU, enabling the execution of MEC services as analyzed in D3.1. The space segment is composed of a set of LEO satellite nodes, $\mathcal{S} = \{s_1, \dots, s_j, \dots, s_N\}$, all positioned within the same orbital shell. This constellation is a key element of the end-to-end architecture defined in D3.2. The constellation is structured into P_S orbital planes, with satellites uniformly distributed within each plane at an inter-satellite separation denoted as d_{IS}. Neighboring orbital planes are spaced by an interplane distance d_{IP}. Each satellite node is characterized by a finite and specific computational capacity, defined by a processor with a CPU operating frequency of f_j and L_j cores, each capable of $c_j$floating-point operations per second (FLOPS) per CPU cycle. The inter-satellite communication links, crucial for data routing within the constellation, are defined by a bandwidth of B_j. To access EC services in this NTN, each device establishes a connection to a specific satellite in the LEO constellation. This connected satellite is referred to as a source satellite or input satellite, denoted by l_s^{in}. This notation signifies that the s-th satellite acts as the initial point of contact for user data, receiving it from devices and potentially routing it to other satellite nodes for distributed processing. This dynamic selection of a source satellite is a key aspect of managing the network's on-

demand resource allocation. The set of all such source satellites is defined as $\mathcal{L}^{in} = \{ l_s \in \mathcal{L} \mid l_s \text{ serves as a source node} \}$.In contrast, satellites that are not currently involved in any communication or processing tasks are considered idle. These idle satellites represent a critical pool of unused resources that can be dynamically recruited as needed for data relaying or parallel processing to handle sudden load spikes or complex tasks. For any satellite $l_s \in \mathcal{L}$, the set of neighboring idle satellites is defined as $I_{l_s} = \{ l_j \in \mathcal{L} \setminus \mathcal{L}^{in} \mid l_j \text{ is idle and directly connected to } l_s \}$. Additionally, we define the global set of idle satellites as $\mathcal{L}^{idle} = \{ l_s \in \mathcal{L} \setminus \mathcal{L}^{in} \mid l_s \text{ is idle} \}$. The remaining satellites in the constellation may be actively participating in inter-satellite relaying or task execution and are grouped under $\mathcal{L}^{active} = \mathcal{L} \setminus (\mathcal{L}^{in} \cup \mathcal{L}^{idle})$.

- **Terrestrial Ground Segment:** A terrestrial ground station is integrated into the system at a predetermined location (x_C, y_C). This station also serves as the location for a Cloud Computing facility. The cloud facility provides a centralized processing capability characterized by a processing capacity of c_c FLOPS per CPU cycle, a CPU frequency of f_c, and L_c cores. This facility is connected to satellite constellation via high-capacity feeder links, maintaining a total bandwidth of B_c for connectivity with any of the LEO satellites within its coverage.

Task Computation and Communication Models

The execution of a task is defined by the time and energy consumed for its processing on a specific device, which can be either a satellite node or a cloud computing facility. For the generic k-th device (where $k \in \mathcal{S} \cup C$), the temporal and energetic consumption for executing the ρ_m-th task are expressed as:

$$T_{c,k}^{\rho_m} = \frac{\Omega_{\rho_m}}{c_k^{\rho_m} f_k^{\rho_m}}, \qquad E_{c,k}^{\rho_m} = \frac{\Omega_{\rho_m}}{c_k^{\rho_m} f_k^{\rho_m}} P_{c,k}$$

In these equations, $T_{c,k}^{\rho_m}$ represents computational time and $E_{c,k}^{\rho_m}$ represents computational energy. Ω_{ρ_m} is the number of CPU cycles required by the task. The parameters $c_k^{\rho_m}$ and $f_k^{\rho_m}$ denote the number of floating-point operations per second and the frequency per CPU cycle, respectively, specifically allocated for the ρ_m-th task on the k-th device. $P_{c,k}$ is the computational power consumption of the k-th device's processor.

The communication model accounts for the temporal and energetic costs of transmitting and receiving data. The total duration of data exchange between any two arbitrary nodes, k and l, for a processing task ρ_m is the sum of the transmission duration and the signal propagation delay. This is given by:

$$T_{tx,kl}^{\rho_m}(\tau_i) = \frac{D_{\rho_m}}{r_{kl}(\tau_i)} + \frac{d_{kl}}{c}$$

In this equation, d_{kl} represents the physical separation between nodes k and l, while c denotes the speed of light. The energy expenditure associated with this transmission is given by:

$$E_{tx,kl}^{\rho_m}(\tau_i) = \frac{D_{\rho_m}}{r_{kl}(\tau_i)} Pt_k$$

Here, $r_{kl}(\tau_i)$ signifies the data rate of the communication link between the nodes at time τ_i, and Pt_k is the transmission power of node k. Similarly, the time and energy required at node l to receive the task data of size D_{ρ_m} from node k are described by:

$$T_{rx,kl}^{\rho_m}(\tau_i) = \frac{D_{\rho_m}}{r_{kl}(\tau_i)} + \frac{d_{kl}}{c}, \quad E_{rx,kl}^{\rho_m}(\tau_i) = \frac{D_{\rho_m}}{r_{kl}(\tau_i)} Pr_l$$

where Pr_l is the power consumed by node l during data reception. The communication channel between any two nodes is presumed to exhibit symmetry in its characteristics. We model the channel characteristics between the k-th and l-th nodes at the i-th time interval using a path loss model. The link gain is represented by:

$$h_{k,l}(\tau_i) = \beta_0 \cdot d_{k,l}^{\theta^l}(\tau_i)$$

where $d_{k,l}(\tau_i)$ denotes the time-varying separation between node k and l. Here, β_0 indicates the power gain of a reference channel at a distance of 1 meter, and θ^l indicates the path loss exponent for the specific communication link between nodes k and l. The achievable data rate is formulated from Shannon's capacity theorem and is given by:

$$r_{kl}(\tau_i) = b_k^{\rho_m}(\tau_i) \cdot \log_2\left(1 + \frac{Pt_k \cdot h_{k,l}(\tau_i)}{N_0}\right), \quad \forall k, l$$

where Pt_k is the transmission power of node k, $b_k^{\rho_m}(\tau_i)$ represents the allocated bandwidth for the communication, and N_0 defines the power of thermal noise, calculated from the noise power spectral density N_T as $N_0 = N_T b_k^{\rho_m}(\tau_i)$.

Task Offloading Process

The system is designed with a seamless task offloading capability, allowing a task initiated by a user to be processed by any capable node within the network, including satellite or cloud computing nodes. When the m-th user offloads its task to a chosen destination node, referred to as the generic k-th node, the total associated time and energy expenses for this entire process (offloading and receiving the results) can be expressed as follows:

$$T_{m,k}^{\text{off}}(\tau_i^{\text{off}}) = \sum\nolimits_{l \in \mathcal{P}_{m,k}} T_{tx,l}^{\rho_m}(\tau_i^{\text{off}}) + T_{c,k}^{\rho_m}(\tau_i^{\text{off}}) + \sum\nolimits_{l \in \mathcal{P}_{k,m}} T_{rx,l}^{r,\rho_m}(\tau_i^{\text{off}})$$

$$E_{m,k}^{\text{off}}(\tau_i^{\text{off}}) = \sum\nolimits_{l \in \mathcal{P}_{m,k}} E_{tx,l}^{\rho_m}(\tau_i^{\text{off}}) + \sum\nolimits_{l \in \mathcal{P}_{k,m}} E_{rx,l}^{\rho_m}(\tau_i^{\text{off}}) + E_{c,k}^{\rho_m}(\tau_i^{\text{off}})$$

In these formulations, the set $\mathcal{P}_{m,k}$ represents the ordered sequence of all intermediary nodes on the path from the source user u_m to the chosen edge computing node k. Similarly, $\mathcal{P}_{k,m}$ is the reverse path for the result to be returned. The summation indicates the aggregation of terms related to transmission, reception, and computation processes across all participating nodes in the offloading sequence. It is assumed that the energy consumed by nodes when they are idle is negligible or is accounted for in a separate baseline budget. Thus, the energy evaluated specifically pertains to the incremental usage resulting from the offloading process itself. The selection strategy for these paths is a key output of our AI-driven placement models, which seek to optimize this end-to-end offloading process. A core assumption in this model is that we are analyzing a single time interval in isolation and thus do not account for the complexity of handovers between satellites.

4.3.4.2 Problem Formulation and Methodology

The challenge of our investigation is the joint optimization of task placement and offloading paths within a highly dynamic and resource-constrained LEO satellite constellation. Building upon the architectural decomposition from WP3, we model the 5G Core Network (5GC) and

Radio Access Network (RAN) functionality as a set of chained micro-services. These are presented to the system as a continuous stream of computational requests, originating from a multitude of users, which must be collectively processed. The problem is thus formulated as a multi-objective optimization task, which must solve two core, joint sub-problems: (1) assigning each request to an optimal computing node (satellite or cloud), and (2) determining the most efficient communication path for each task from its origin to its assigned node and back.

In this analysis, we explore scenarios in which distinct users simultaneously generate multiple requests. Each request may only be processed by one satellite at any given moment, indicating that each task is an indivisible entity and cannot be partitioned between different nodes. Each request is encapsulated within a single task and, consequently, assigned to one node. The allocation strategy, influenced by multiple requests, is dictated not only by the system's current state and the requisite conditions of each request but also by how different requests are collectively assigned. To achieve this, the assignment of tasks to each edge computing satellite node entails solving a multidimensional problem, requiring tasks to be assigned collectively while adhering to each task's specific requirements.

The objective of the analyzed system is to accurately identify the node capable of hosting the processing task. This requires a combined effort to minimize both the E2E delay and total energy consumption, allowing the chosen node to balance between energy use and processing time. The problem can be formally expressed as:

$$\min_{k,\mathcal{P}} \sum_{m} \left(\alpha_1 \cdot T_{m,k}^{\text{off}}\left(\tau_i^{\text{off}}\right) + \alpha_2 E_{m,k}^{\text{off}}\left(\tau_i^{\text{off}}\right) \right)$$

Here, k refers to the index of the computing node (including both satellite and cloud nodes) selected for task offloading, and $\mathcal{P}$ denotes the offloading path from the source user to node k and back. The optimal selection of this node involves not only selecting the node itself but also determining the most efficient path. The weighting coefficients, α_1 and α_2, are introduced to balance the trade-off between minimizing offloading delay and energy consumption. A value of $\alpha_1 > \alpha_2$ prioritizes latency-sensitive applications, while $\alpha_2 > \alpha_1$ is more suitable for energy-constrained scenarios, such as battery-powered IoT devices. The energy term $E_{m,k}$ is a key proxy for resource utilization on the satellite, as it directly impacts the limited onboard power budget. The method for selecting k will be elaborated later, in the context of benchmarks and our proposed load balancing approach. The strategy for selecting paths relies on the Dijkstra algorithm, which is employed to calculate the optimal routes between satellite nodes. It is important to note that while this algorithm can find optimal static paths, it is executed for each time interval τ_i to account for the dynamic changes in the constellation topology.

Given the constraints in link capacity and processing power, the former minimization problem is defined by two conditions:

$$\begin{cases} \sum_{m \in \mathcal{M}_k} r_{m,k} \leq R_k \\ \sum_{\rho_m \in \mathcal{P}_k} c_k^{\rho_m} \leq C_k \end{cases}$$

Here, $\mathcal{M}_k$ denotes the set of users assigned to node k and $\mathcal{P}_k$ is the set of tasks processed at node k. R_k denotes the upper limit of data throughput for the node k, and C_k represents the upper limit of the processing capacity available for node k. These constraints are applied to each computing node independently and ensure that the total data rate and processing demand of all offloaded tasks do not exceed the node's physical capabilities.

The joint optimization of task assignment and routing, subject to these constraints, belongs to a class of NP-hard problems, making it computationally intractable to solve optimally in a dynamic, real-time environment. This complexity is heightened by the time-varying nature of the LEO constellation topology and user demands. The problem can be viewed as a combinatorial optimization task where the number of possible task-to-node assignments and routing paths grows exponentially with the number of users and satellites. Therefore, traditional optimization algorithms are unsuitable, justifying our proposed AI-driven methodologies as a means to find near-optimal, adaptive solutions [25]. The specific strategies for selecting the path and the computing node are elaborated later, within the context of our proposed load-balancing approaches.

4.3.4.3 MEC Placement and Load Balancing Strategies

In this section, we present the solutions developed to address the complex, multi-objective optimization problem of MEC placement and task offloading in our dynamic LEO satellite network. While existing scientific research has explored the transition from centralized cloud computing to distributed edge computing, a significant gap remains in solutions tailored for the unique constraints of satellite-based systems. Specifically, prior work has often overlooked joint optimization of radio resource allocation and offloading processes, as well as the critical consideration of energy consumption on the satellite itself. Our proposed strategies are designed to directly address these research gaps by providing adaptive, intelligence-driven methods for task distribution.

To provide a comprehensive performance analysis, we first establish two benchmark solutions that represent conventional data processing scenarios. These fixed-allocation methods serve as baselines for evaluating the performance of our more sophisticated heuristic and AI-driven approaches.

Benchmark Strategies

- Cloud-based Processing Approach: In this approach, all user data is transmitted through the satellite network to a centralized terrestrial cloud facility for processing. Due to their extensive resources, cloud computing infrastructures offer virtually limitless computational capacity ($C_k \to \infty$). However, this approach introduces substantial data transfer delays and link energy consumption, as tasks must traverse multiple satellite hops and a long feeder link to the ground. This strategy is essential for establishing a baseline for maximum latency and energy cost, highlighting the benefits of localized edge processing.

- Nearest Satellite-based Processing Approach: This strategy represents a rigid edge computing model where each user's task is offloaded to the nearest satellite node for processing. This approach is designed to minimize initial transmission delays, as it reduces the propagation path between the user and the computing node. While it excels at minimizing latency for single-user scenarios, it is highly susceptible to congestion. When multiple users are in the coverage area of a single satellite, the node can quickly reach its capacity limits, leading to task rejections or significant queuing delays that increase the overall end-to-end latency and computational costs.

Heuristic Load Balancing Strategies

Expanding upon the benchmarks, we have devised two heuristic algorithms that are efficient at distributing processing tasks across a pre-defined path of satellite nodes. These strategies, while effective at distributing load, are still reactive and lack the predictive, holistic optimization capabilities of a truly intelligent system. Both approaches utilize a shortest-path routing

mechanism based on the Dijkstra algorithm, which is re-executed for each time interval τ_i to account for the dynamic LEO constellation topology.

- Random Allocation (RA): The Random Allocation strategy serves as a baseline for a decentralized, non-optimizing approach. In this model, a processing task is randomly assigned to any of the available satellite nodes located along the shortest path from the user's nearest satellite (ingress node) to the cloud facility. The primary advantage of this strategy is its simplicity and its ability to distribute the workload without requiring complex state information. Unlike fixed approaches that can lead to rapid congestion, RA spreads the load, mitigating the risk of a single node becoming a bottleneck under heavy user demand. It provides a clear reference for the benefits of basic load-spreading over a rigid assignment policy.

- Maximum Capacity Allocation (MCA): The MCA algorithm is a greedy, capacity-aware approach that aims to efficiently distribute the workload between nodes along the pathway that connects the ingress satellite to the cloud. The core concept is to prioritize the nearest nodes to minimize latency, but only until they reach a pre-established capacity threshold (C_k). Once a node is at full capacity, it is bypassed, and the task is re-assigned to the next available node along the path towards the cloud. The procedure is applied uniformly to all processing tasks. As a fail-safe, should all nodes along the path be at full capacity, the cloud facility acts as a fallback option, drawing upon its presumed limitless processing capabilities. The algorithm for this approach is best represented as a flow diagram to show the decision-making process.

AI-Driven Load Balancing

The highly dynamic nature of LEO satellite networks—characterized by rapid topology changes, fluctuating link conditions, and heterogeneous task demands—poses significant challenges for deterministic and heuristic algorithms. These reactive strategies, such as RA and MCA, respond to network conditions after they occur, which can lead to suboptimal performance in a high-speed, dynamic environment. Deep Reinforcement Learning (DRL) offers a data-driven alternative that adaptively and proactively optimizes task allocation by learning from historical and real-time network states. Recent studies have also investigated DRL/MARL-based offloading and resource allocation specifically for LEO satellite edge computing networks [29]. Unlike heuristic methods, a DRL agent can balance the complex latency-energy trade-off while accounting for complex interdependencies between satellite resources, task requirements, and channel dynamics.

We model the task allocation process as a Markov Decision Process (MDP) to be solved by the DRL agent:

- State Space ($\mathcal{S}$): The state is a representation of the network at any given moment. It is composed of a vector of discrete and continuous variables including:
  - **Satellite states:** Current CPU utilization (%), available memory (GB), remaining power budget (%), and queue length (number of tasks).
  - **Task states:** For each active task, its size (MB), computational requirements (CPU cycles), and latency deadline (s).
  - **Network states:** Real-time Inter-Satellite Link (ISL) quality (Signal-to-Interference-plus-Noise Ratio or SINR), queuing delays on each link, and the availability of paths to the cloud.

- **Global context:** The total number of active users and tasks on the network.

- Action Space ($\mathcal{A}$): The action space is discrete, corresponding to the decision of where to offload the current task. For a given task, the agent can select one of the following actions:
  - Assign the task to a specific satellite s_j along path $\mathcal{P}$.
  - Assign the task to the cloud facility. The total number of possible actions is equal to the number of candidate satellites plus one (for the cloud), which keeps the action space manageable. All actions respect the satellite's capacity and link throughput constraints as defined in the problem formulation.

- Reward Function ($\mathcal{R}$): The agent's reward is designed to incentivize the behaviors that align with our optimization goals. It is a weighted function of the negative offloading latency and energy consumption.

$$r = -\left(\alpha_1 T_{m,k}^{off} + \alpha_2 E_{m,k}^{off}\right)$$

  Penalties are applied for missed deadlines ($T_{m,k}^{off} > T_{\rho_m}$) and for actions that violate a satellite's capacity. The reward is normalized to prevent magnitude imbalances between the latency and energy terms, and the coefficients α_1 and α_2 allow us to tune the agent's behavior to prioritize either latency or energy efficiency.

DRL Architecture and Training

We employ the Proximal Policy Optimization (PPO) algorithm for training [26]. PPO is selected for its stability and data efficiency, which are crucial for training in a dynamic simulation environment where data collection can be computationally expensive. It allows the agent to make small, controlled updates to its policy, preventing it from straying into unstable states.

The agent's intelligence is powered by two neural networks.

- *Actor Network*: This network takes the current state vector as input and outputs a probability distribution over the available actions. It uses a Long Short-Term Memory (LSTM) layer to process the temporal sequence of states, allowing the agent to capture the dynamic dependencies of the satellite network [27]. The output of the LSTM is fed to fully connected (FC) layers, which finally produce the Softmax action probabilities.

- *Critic Network*: This network, with a similar input structure, outputs a single value estimate ($V(s)$) of the expected future reward from the current state. This value is used by the actor to guide its learning process.

The training of the DRL agent is conducted within a simulated environment that accurately models the orbital dynamics, user mobility, and network characteristics of the LEO constellation. The training is performed in a Python-based simulator that emulates the LEO constellation, including the mobility of satellites, the dynamic nature of ISLs, and user traffic generation. The agent uses prioritized experience replay to focus on learning from states that have the most significant impact on performance, such as when satellites become overloaded [28].

Deployment and Inference Strategy

Once the DRL agent has been thoroughly trained in the simulation environment, the final, optimized neural network model can be deployed directly onto the on-board computers of the satellite nodes. During real-world operation, this model will function as the on-board task placement and offloading policy. At each time interval τ_i , the satellite's on-board agent will:

1. Collect real-time state information (e.g., current load, queue lengths, link quality).
2. Feed this state vector as input to the trained neural network.
3. The network will perform a rapid inference (a single forward pass) to output the optimal offloading action.
4. The satellite will then execute this decision, either processing the task locally or forwarding it to the assigned node.

This strategy ensures that the solution is not only intelligent but also practical and executable in a real-time, distributed system.

4.3.4.4 Performance Evaluation

In this section, we present the numerical results obtained through computer simulations, which were performed using a custom Python-based simulator developed on the Google Colab platform. This collaborative environment enables seamless development and provides free access to computing resources, including GPUs and TPUs, to accelerate the analysis.

For our simulations, we used an activity factor of 0.1, meaning that for a given number of active nodes, the total number of IoT nodes in the system is approximately 10 times higher. This factor models the realistic scenario where only a fraction of devices is active at any given moment, despite a much higher total user population. The numerical analysis was performed with a hypothetical satellite constellation designed for regional coverage over Europe, as utilizing satellites on the Earth's antipodal side for edge computing would be impractical due to excessive communication delays.

The simulation parameters for the constellation and tasks are defined as follows:

- **Satellite Constellation:** 3 orbital planes, each with 5 satellites.
- **Orbital Parameters:** Satellites maintain an orbital radius of 1000 km, with inter-satellite and inter-plane separations of 500 km and 200 km, respectively.
- **Power Consumption:**
  - Transmission: 1 W per satellite.
  - Reception: 0.8 W per satellite.
  - Computation: 1 W per satellite.
- **IoT Nodes:** A variable range of 50 to 300 ground IoT nodes, homogeneously dispersed over a 25000 km^2 region.
- **Communication Bandwidth:**
  - Ground-to-satellite: 100 MHz, distributed among all users connected to a satellite.

- Inter-satellite links: 100 MHz
- Feeder link to cloud: 20 MHz per IoT message.

- **Computational Requirements:**
  - Each node generates tasks of size 2 MB
  - Task computational requirement: 1000 FLOPS per bit
  - Satellite computational capacity: 30 GFLOPS.
  - Cloud computational capacity: 10 GFLOPS per IoT task.

Simulation Results

As shown in Figure 4-14, communication energy consumption varies significantly across strategies. The cloud processing approach incurs the highest energy cost due to the long-distance data transmission to terrestrial servers, which necessitates multiple satellite hops and a long, high-power feeder link. This makes it an unsustainable solution for energy-constrained on-board systems. While the nearest satellite strategy minimizes initial transmission distance, the random allocation (RA) method can select distant nodes, leading to inefficient paths that are not optimized for energy. Both the Maximum Capacity Allocation (MCA) and the DRL strategies perform well by prioritizing in-network processing and thereby avoiding the high-cost cloud link entirely. The DRL's ability to learn from the dynamic network state allows it to make nuanced routing decisions, subtly outperforming the static rules of MCA by identifying and avoiding congested inter-satellite links that would otherwise increase energy consumption.

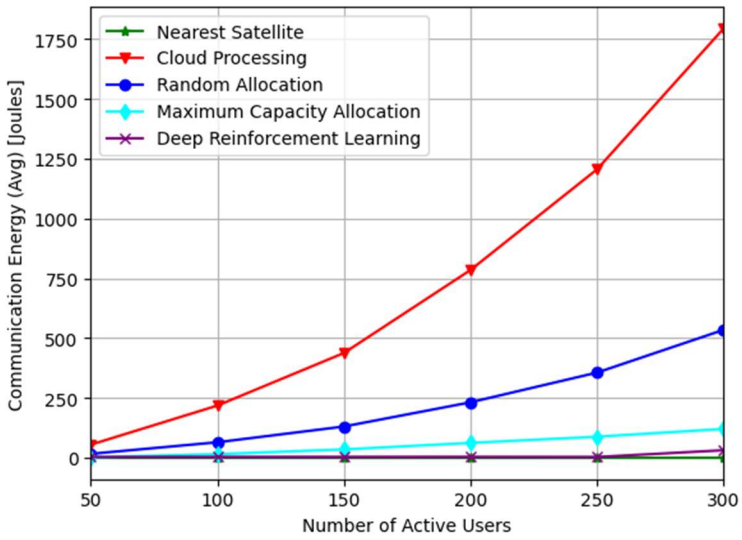


Figure 4-14: Average Communication Energy

Figure 4-15 reveals the critical impact of load distribution on computational energy. The nearest satellite and random allocation strategies perform poorly due to their inherent lack of load-balancing, leading to severe congestion on individual satellites. This congestion forces tasks to wait in queues, causing processors to draw power for extended durations and skyrocketing the total computation energy. While cloud processing offloads this computational cost from the satellites, it simply moves the problem to a different part of the network while adding massive communication overhead. The MCA heuristic effectively addresses satellite-side congestion

by distributing tasks along a path based on predefined capacity thresholds, preventing any single node from becoming a computational bottleneck. However, the DRL strategy demonstrates a superior, dynamic approach. It learns to optimize the entire system's computational load in real-time, resulting in a far more efficient distribution of tasks that minimizes the collective computation energy and significantly outperforms all other methods, including MCA, by proactively balancing the load before congestion occurs.

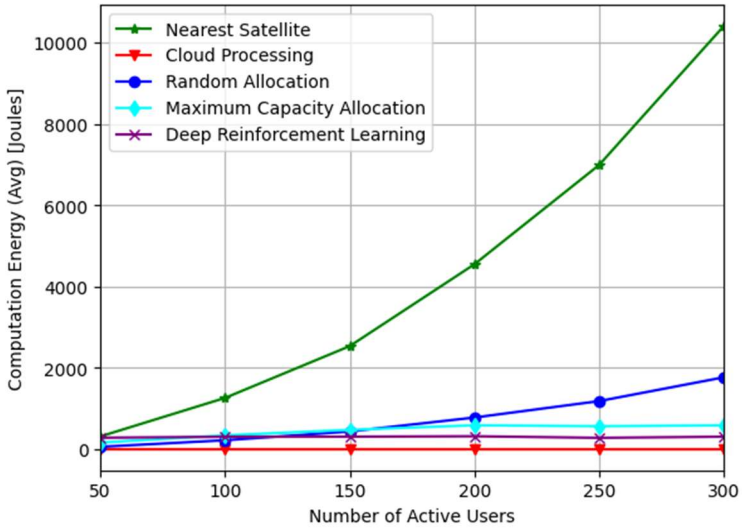


Figure 4-15: Average Computation Energy

The total energy consumption, a sum of communication and computation costs, is shown in Figure 4-16. The cloud processing and random allocation strategies represent the extremes of failure: the former due to its immense communication energy and the latter due to its computational inefficiency. The nearest satellite approach is similarly inadequate, as its minimal communication cost is utterly negated by its high computation energy from congestion. The MCA heuristic emerges as a robust solution, successfully balancing the two energy components through its logical, rule-based distribution of tasks, proving its value as a non-AI benchmark. Ultimately, the DRL strategy stands apart, achieving the lowest total energy consumption. Its ability to learn and simultaneously optimize both transmission paths and computational load distribution represents a fundamental advancement over heuristic methods, solidifying its role as the optimal strategy for energy-constrained environments.

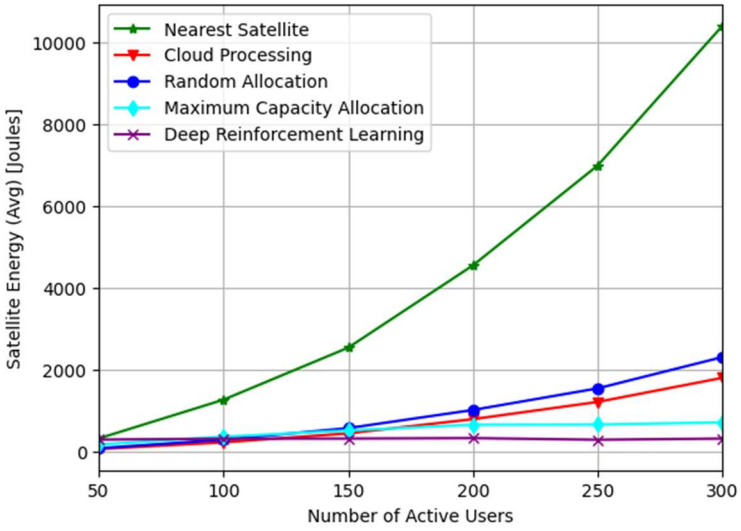


Figure 4-16: Average Satellite Total Energy

Analysis of communication latency (Figure 4-17) highlights the impact of data travel distance. The cloud processing strategy incurs the highest delay due to the inevitable propagation latency of the long round trip to a ground station. The random allocation method introduces unpredictable and often high latency by routing data through sub-optimal paths within the network. Conversely, the nearest satellite strategy achieves the lowest possible communication latency for the initial hop. Both the MCA and DRL strategies effectively minimize this metric by keeping data within the satellite network and using intelligent routing. Their performance is strong and stable, with DRL's adaptive learning allowing it to match or marginally improve upon the excellent performance of the MCA heuristic by continuously identifying the most efficient paths.

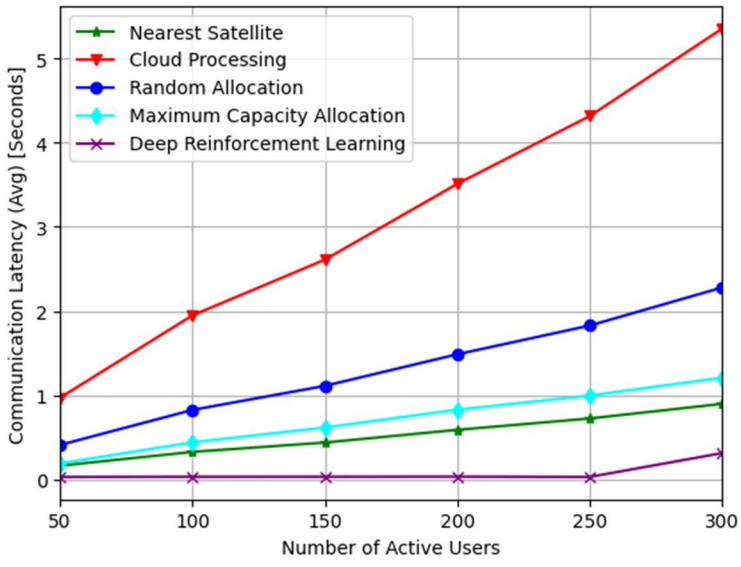


Figure 4-17: Average Communication Latency

As illustrated in Figure 4-18, computation latency is dominated by the effectiveness of load distribution. The nearest satellite and random allocation strategies fail here, as they cause severe processing bottlenecks on individual satellites, leading to exorbitant queueing delays.

Cloud processing leverages its vast ground-based resources to deliver the lowest computation latency, representing its primary strength. The MCA heuristic successfully mitigates satellite-side congestion by offloading tasks once a node's capacity is reached, providing a substantial improvement over the simpler methods. However, the DRL strategy surpasses MCA by proactively learning network conditions and dynamically distributing the load to prevent bottlenecks before they occur, resulting in significantly lower computation latency and closely approaching the performance of the cloud.

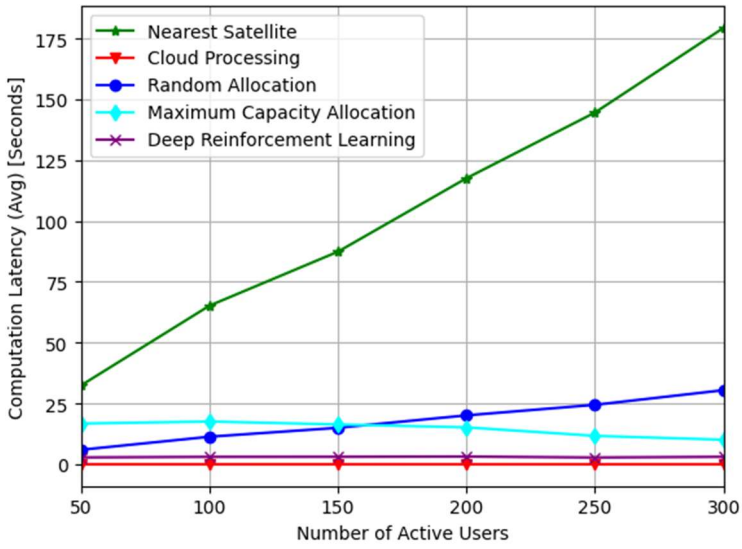


Figure 4-18: Average Computation Latency

The total latency (communication plus computation), defining the user experience, synthesizes all these effects, shown in Figure 4-19. The nearest satellite and random allocation strategies yield the worst user experience due to their crippling computation latency. Cloud processing is strong at low user loads, where its computational power outweighs its communication delay, but its performance degrades as user count increases and transmission paths become congested. The MCA heuristic provides a consistent and reliable low-latency service by effectively balancing both latency components, making it a viable and practical solution. The DRL-based strategy delivers the best overall quality of service, particularly on a scale. By jointly adaptively optimizing communication and computation delays, it outperforms MCA and even surpasses Cloud Processing at higher loads, demonstrating that an intelligently managed satellite network can provide superior responsiveness without relying on terrestrial infrastructure. The superior total latency achieved by the DRL agent, particularly under high user loads, demonstrates its ability to meet the stringent performance targets for latency-sensitive services.

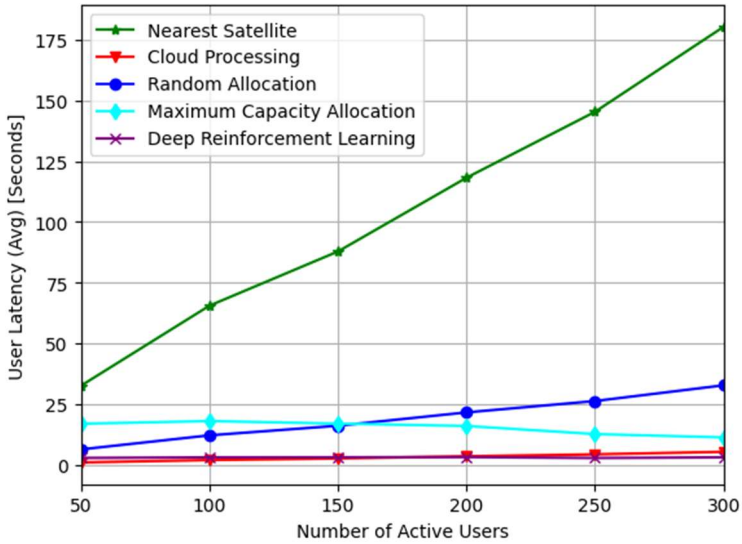


Figure 4-19: Average node latency

4.3.4.5 Conclusions

In conclusion, this analysis demonstrates that an AI-driven approach provides a powerful and practical solution to the MEC placement problem outlined in Task 5.5. Rather than relying on a fixed split model, the DRL agent learns a dynamic policy that continuously optimizes the placement of micro-services across the space and ground segments. This work provides a quantitative framework for trading off latency and on-board resource consumption, offering a clear path towards an organic, self-organizing core network for the data plane.

4.4 TN-NTN INTEROPERABILITY

The deployment of mega-constellation non-terrestrial networks introduces a layered architecture where control, user, and application functions must be placed and managed in response to constantly changing topologies. Decisions regarding link establishment, integration with terrestrial domains through feeder links, routing within the constellation, and the positioning of RAN, core, and application layers are central to ensuring that the control and data planes operate without congestion and within the targeted delay budgets.

Terrestrial networks, by contrast, are limited in their ability to adapt dynamically. Their operational models are largely confined to handling failure scenarios by means of redundancy and hot-standby configurations. For this reason, the integration of terrestrial and non-terrestrial segments must be arranged in such a way that the terrestrial components remain minimally affected, while the constellation handles the majority of operational adaptation internally.

Different interoperability models arise due to the existence of distinct administrative domains, even though all adhere to the 3GPP standards that guarantee interface-level interoperability. As illustrated in Figure 4-20, three representative approaches can be identified.

Over-the-Top Backhauling. In this model, the terrestrial network operator uses the non-terrestrial network as a backhaul provider. End-to-end connectivity is established between selected terrestrial locations under specific service level agreements, similar to today's satellite-based transport services. Operational control remains entirely separate, with no need for shared authentication or signalling exposure between the networks. This preserves clear

separation but prevents advanced features of the non-terrestrial infrastructure from influencing the terrestrial domain.

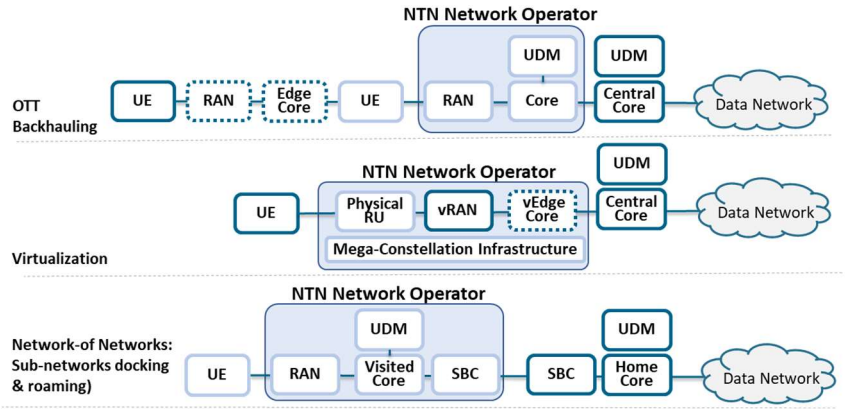


Figure 4-20: High Level view of TN-NTN Interoperability Models

Virtual Infrastructures. In this model, terrestrial network functions are deployed directly on the satellite infrastructure, following a paradigm similar to Infrastructure-as-a-Service in cloud networks. Multiple terrestrial operators may coexist virtually on the same constellation, sharing satellite antennas under the management of the non-terrestrial operator. This arrangement makes the placement of RAN and core functions critical, and routing strategies, such as semantic routing, require careful boundary control. To prevent unpredictable propagation of routing effects into the terrestrial systems, semantic routing should be confined to the constellation gateways. While this restriction limits optimization opportunities, it avoids destabilizing terrestrial routing. Resource planning at the gateways therefore becomes decisive, especially under adverse conditions such as weather-related attenuation, where the redirection of traffic must remain contained within the non-terrestrial domain.

Network of Networks. In this model, the non-terrestrial operator offers a roaming-like service to terrestrial operators, which may take the form of classical roaming or more advanced sub-network interworking foreseen in 6G. To support this, a Service Border Control (SBC) function is introduced. The SBC consolidates multiple functions already standardized in 5G and enables negotiation of service agreements, secure discovery, and controlled interconnection between the domains. The use of SBCs allows each network to continue its independent optimization, with only the interconnection points acting as binding constraints. As long as these points remain stable, each operator can evolve its own network without cross-domain propagation of operational changes. The main challenge lies in the limited computational resources of satellite payloads, which restrict the degree of isolation and independence achievable in practice.

Outlook between 5G-Advanced and 6G. These interoperability models illustrate the gradual evolution from today's static interconnection arrangements towards a more flexible integration of terrestrial and non-terrestrial infrastructures. In 5G-Advanced, adaptation remains confined to gateway-level adjustments and static agreements, while in 6G the ambition is to support dynamic interworking, relocation of functions, and unified state handling across domains. The long-term vision is a seamless TN–NTN continuum where interoperability no longer requires manual agreements but is embedded into the architecture itself.

5 CONCLUSION

This deliverable has presented the final results of Work Package 5, combining AI-driven networking, end-to-end QoS mechanisms, on-board networking capabilities, and TN–NTN interoperability models. Across these domains, the work has consolidated both algorithmic advancements and conceptual frameworks tailored to the unique constraints of non-terrestrial networks.

A number of key findings should be underlined:

- **Algorithms:** Dedicated AI methods were designed and evaluated for O-RAN virtual function placement, beam hopping optimization, and bearer selection under NTN conditions, together with reinforcement learning–based strategies for dynamic MEC placement.
- **Implementations and validations:** Proof-of-concept prototypes, most notably the UPF shortcut mechanism implemented in Open5GCore, were validated in distributed testbeds. Simulations confirmed the feasibility of feeder link rerouting and delay-aware QoS maintenance.
- **Concepts:** Several architectural principles were introduced, including the deployment of micro-cores in space, active state synchronization across satellites, and the definition of TN–NTN interoperability through backhauling, virtualization, and network-of-networks models.

Together, these contributions demonstrate that integrating NTN capabilities into beyond-5G and 6G infrastructures requires a careful balance between validated implementations, AI algorithms, and forward-looking concepts. While the algorithms provide immediate performance improvements, the implementations prove feasibility in realistic environments, and the concepts outline promising technologies that must be further matured to achieve robust and scalable NTN integration.

REFERENCES

[1] O-RAN ALLIANCE, "O-RAN Architecture Description," Version 14.0, 2025. [Online]. Available: https://www.o-ran.org/specifications

[2] O-RAN Software Community (O-RAN SC), "O-RAN Architecture Overview," documentation page. [Online]. Available: https://docs.o-ran-sc.org/en/latest/architecture/architecture.html

[3] M. Polese, L. Bonati, S. D'Oro, S. Basagni and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," in IEEE Communications Surveys & Tutorials, vol. 25, no. 2, pp. 1376-1411, Second quarter 2023, doi: 10.1109/COMST.2023.3239220.

[4] ETSI (3GPP), ETSI TS 123 288 V19.5.0 (2026-01), "5G; Architecture enhancements for 5G System (5GS) to support network data analytics services (3GPP TS 23.288 Release 19)." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123200_123299/123288/19.05.00_60/ts_123288v190500p.pdf

[5] ETSI (3GPP), ETSI TS 123 501 V17.12.0 (2024-04), "5G; System architecture for the 5G System (5GS) (3GPP TS 23.501 Release 17)." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/17.12.00_60/ts_123501v171200p.pdf

[6] ETSI (3GPP), ETSI TS 124 193 V16.2.0, "5G; 5G System; Access Traffic Steering, Switching and Splitting (ATSSS); Stage 3 (3GPP TS 24.193)." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/124100_124199/124193/16.02.00_60/ts_124193v160200p.pdf

[7] 3GPP, "Non-Terrestrial Networks (NTN) Overview," 3GPP technologies page. [Online]. Available: https://www.3gpp.org/technologies/ntn-overview

[8] X. Lin, S. Rommer, S. Euler, E. A. Yavuz and R. S. Karlsson, "5G from Space: An Overview of 3GPP Non-Terrestrial Networks," in IEEE Communications Standards Magazine, vol. 5, no. 4, pp. 147-153, December 2021, doi: 10.1109/MCOMSTD.011.2100038

[9] K. Chen, X. Liu, and W. Li, "A Distributed Multi-Agent Deep Reinforcement Learning Approach for Dynamic Beam Hopping Optimization in LEO Mega-Constellations," Swarm and Evolutionary Computation, vol. 97, Aug. 2025, Art. no. 102039, doi: 10.1016/j.swevo.2025.102039

[10] X. Xie, K. Fan, W. Deng, N. Pappas, and Q. Zhang, "Multi-Satellite Beam Hopping and Power Allocation Using Deep Reinforcement Learning," arXiv preprint arXiv:2501.02309, 2025. [Online]. Available: https://arxiv.org/abs/2501.02309

[11] X. Gao, R. Liu, and A. Kaushik, "A Distributed Virtual Network Function Placement Approach in Satellite Edge and Cloud Computing," arXiv preprint arXiv:2104.02421, 2021. [Online]. Available: https://arxiv.org/abs/2104.02421

[12] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual Network Function Placement and Resource Optimization in NFV and Edge Computing Enabled Networks," Computer Networks, vol. 152, pp. 12–24, Apr. 2019, doi: 10.1016/j.comnet.2019.01.026.

[13] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Machine-Learning Based Prediction and Optimization of Mobile Metro-Core Networks," in Proc. IEEE Photon. Soc. Summer Topicals Meeting Ser. (SUM), no. 1, Jul. 2018, pp. 155–156.

[14] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y. C. Liang, "Network slice reconfiguration

by exploiting deep reinforcement learning with large action space," IEEE Trans. Netw. Service Manage., vol. 17, no. 4, pp. 2197–2211, Dec. 2020.

[15] E. Zeydan, C. J. Vaca-Rubio, L. Blanco, R. Pereira, M. Caus and A. Aydeger, "F-KANs: Federated Kolmogorov-Arnold Networks," 2025 IEEE 22nd Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2025, pp. 1-6, doi: 10.1109/CCNC54725.2025.10976205.

[16] W. Lu, H. Fang, and Z. Zhu, "AI-assisted resource advertising and pricing to realize distributed tenant-driven virtual network slicing in interDC optical networks," in Proc. 22nd Conf. Opt. Netw. Design Model. (ONDM), 2018, pp. 130–135.

[17] X. Zhang, W. Lu, B. Li, and Z. Zhu, "DRL-based network orchestration to realize cooperative, distributed and tenant-driven virtual network slicing," in Proc. Asia Commun. Photon. Conf. (ACP), 2019, pp. 2019–2021.

[18] H. Zhang and V. W. S. Wong, "A two-timescale approach for network slicing in C-RAN," IEEE Trans. Veh. Technol., vol. 69, no. 6, pp. 6656–6669, Jun. 2020.

[19] S. Meng, Z. Wang, H. Ding, S. Wu, X. Li, P. Zhao, C. Zhu, and X. Wang, "RAN slice strategy based on deep reinforcement learning for smart grid," in Proc. Comput., Commun. IoT Appl. (ComComAp), 2019, pp. 106–111.

[20] X. Chen, Z. Zhao, C. Wu, M. Bennis, H. Liu, Y. Ji, and H. Zhang, "Multitenant cross-slice resource orchestration: A deep reinforcement learning approach," IEEE J. Sel. Areas Commun., vol. 37, no. 10, pp. 2377–2392, Oct. 2019

[21] B. Barth, S. Baradie, C. Keuker, R. Zaitsev, G. Guruvayoorappan and T. De Cola, "MEC in 3D-Networks: Supporting Application Controlled QoS," in *IEEE Access*, vol. 13, pp. 120960-120972, 2025, doi: 10.1109/ACCESS.2025.3586811.

[22] 3GPP TS 23.558, "Architecture for enabling Edge Applications", 19.6.0, 2025-06

[23] ETSI GS MEC 003 V4.1.1, "Framework and Reference Architecture", 2025-05

[24] S. Robitzsch, M. Centenaro, N. di Pietro, L. Cordeiro, A. S. Gomes, P. Sanders, and A. Ishaq, "Prospects on the adoption of a microservice-based architecture in 5G systems and beyond," Computer Networks, vol. 237, Art. no. 110058, 2023, doi: 10.1016/j.comnet.2023.110058

[25] D. Hortelano, I. de Miguel, R. J. Durán Barroso, J. C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R. M. Lorenzo, and E. J. Abril, "A comprehensive survey on reinforcement-learning-based computation offloading techniques in Edge Computing Systems," Journal of Network and Computer Applications, vol. 216, Art. no. 103669, Jul. 2023, doi: 10.1016/j.jnca.2023.103669.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347, Jul. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[27] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," arXiv:1511.05952, Nov. 2015. [Online]. Available: https://arxiv.org/abs/1511.05952

[29] H. Li, J. Yu, L. Cao, Q. Zhang, Z. Song, and S. Hou, "Multi-agent reinforcement learning based computation offloading and resource allocation for LEO Satellite edge computing networks," Computer Communications, vol. 222, pp. 268–276, Jun. 2024, doi: 10.1016/j.comcom.2024.05.008

[30] ETSI MEC ISG, White Paper No #36, "Harmonizing standards for edge computing - A

synergized architecture leveraging", 1st edition, July 2020 ISBN No. 979-10-92620-35-5

[31] 3GPP TR 23.700-01, "Study on application enablement for Satellite access enabled 5G Services", 19.0.0, 2024-09-30

[32] https://github.com/lightedge

[33] https://gitlab.eurecom.fr/oai/orchestration

[34] OpenAirInterface RNIS readme, available at https://gitlab.eurecom.fr/oai/orchestration/oai-mec/oai-rnis, last accessed: 29/09/2025

[35] Intel® Smart Edge : Controller Software, "Product Specification Sheet", available at [https://cdrdv2-public.intel.com/631044/Intel%20Smart%20Edge%20Controller%20Software%20Spec%20Sheet.pdf], last accessed: 29/09/2025

[36] 6GNTN Deliverable 3.1. "D3.1 REPORT ON 3D MULTI LAYERED NTN ARCHITECTURE", Version 1.0, 10-2023

[37] Polese, M., et al. (2023). "An O-RAN Perspective on 6G." IEEE Communications Magazine, 61(3), 89-95.

[38] Giordani, M., et al. (2020). "A Tutorial on Non-Terrestrial Networks for 6G: Vision, Challenges, and Enabling Technologies." IEEE Communications Surveys & Tutorials, 22(4), 2451-2487.

[39] Sun, H., et al. (2020). "Deep Reinforcement Learning for Radio Resource Management in 5G and Beyond." IEEE Communications Magazine, 58(8), 56-62.

[40] Boutiba, M., et al. (2022). "AI-Powered Zero-Touch Network and Service Management in 5G and Beyond: A Survey." IEEE Communications Surveys & Tutorials, 24(2), 1109-1153.

[41] Li, R., et al. (2018). "Deep Reinforcement Learning for Resource Management in Network Slicing." IEEE Access, 6, 74429-74441.

[42] SNS JU 5G-Stardust "D3.2: End-to-end ground and space integrated architecture", February 2024;

[43] SNS JU 5G-Stardust "D5.3: Preliminary report on AI-data driven networking and QoS management", June 2024

[44] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., *et al.* (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529–533. https://doi.org/10.1038/nature14236

[45] Software Radio Systems, "srsRAN: Open Source 4G and 5G Software Radio Access Network," 2024. [Online]. Available: https://github.com/srsran/srsRAN_Project

