6Green

Green Technologies for 5/6G Service-Based Architecture

Grant Agreement No	101096925
Full Title	Green Technologies for 5/6G Service-Based Architectures
Start date	01 Jan 2023
End date	30 Apr 2026
Duration	40
Project URL	https://www.6green.eu
Coordinator	CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI (CNIT)

Deliverable D3.3

The 6Green 5/6G Service Based Architecture

Contractual due date	31/12/2025 (M36)	Actual submission date	04/02/2026
Nature	R - Report	Dissemination Level	PU - Public
Lead Beneficiary	ATOS		
Responsible Author	Jesus Benedicto (ATOS)		
Contributions from	Daniele Ronzani (HPE); Marius Iordache, Catalin Brezeanu, Daniel Tichiu, Carmen Patrascu (ORO); Orazio Toscano (TEI); Luis M. Contreras, Guillermo Sanchez (TID); Alexandros Valantasis (UBITECH); Anastasios Zafeiropoulos, Nikos Fryganiotis (ICCS); Chiara Lombardo (CNIT); Claudio Cicconetti, Raffaele Bruno (CNR); Armand Divband (EURECOM)		

Revision history

Version	Issue Date	Changes	Contributor(s)
v0.1	14/05/2025	TOC Initial version.	Jesús Benedicto (ATOS)
v0.2	15/09/2025	Contributions in Section 3 and 4.	Daniele Ronzani (HPE); Marius Iordache, Catalin Brezeanu, Daniel Tichiu, Carmen Patrascu (ORO); Orazio Toscano (TEI); Luis M. Contreras, Guillermo Sanchez (TID); Alexandros Valantasis (UBITECH); Anastasios Zafeiropoulos, Nikos Fryganiotis (ICCS); Chiara Lombardo (CNIT); Claudio Cicconetti, Raffaele Bruno (CNR), Armand Divband (EURECOM)
v0.3	28/01/2026	Internally reviewed.	Chiara Lombardo (CNIT); Rudolf Susnik (ININ)
v0.4	30/01/2026	Version ready for submission	Jesús Benedicto (ATOS)
v1.0	04/02/2026	Final review of content and format.	Riccardo Rapuzzi (CNIT)

Disclaimer

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the 6Green Consortium make no warranty of any kind with regard to this material includeing, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the 6Green Consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the 6Green Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
5G	5th Generation of cellular technologies
5GC	5G Core
6G	6th Generation of cellular technologies
AF	Application Function
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
BESS	Berkeley Extensible Software Switch
BSSF	Business Support System Function
CCMF	Cloud-Continuum Management Function
CN	Core Network
E2E	End to End
EC	Energy Consumption
EdgeMF	Edge-cloud Management Function
EDN	Edge Data Network
EE	Energy Efficiency
EGMF	Exposure Governance Management Function
ENI	Experiential Networked Intelligence
ENIF	ENI Function
ES	Energy Saving
GRE	Generic Routing Encapsulation
HTTP	Hyper-Text Transfer Protocol
HW	Hardware
ICT	Information and Communications Technology
IDAF	Infrastructure Data Analytics Function
INT	In-band Network Telemetry
IT	Information Technologies
K8s	Kubernetes
KPI	Key Performance Indicator
LSTM	Long short-term memory

Abbreviation / Term	Description
MDAF	Management Data Analytics Function
MILP	Mixed-Integer Linear Programs
ML	Machine Learning
MLI	Machine Learning Inference
NEF	Network Exposure Function
NF	Network Function
NFMF	Network Function Management Function
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NS	Network Service
NSDAF	Network Slice Data Analytics Function
NSENIF	Network Slice ENIF
NSI	Network Slice Instance
NSMF	Network Slice Management Function
NSPCF	Network Slice Policy Control Function
NST	Network Slice Template
NWDAF	Network Data Analytics Function
P4	Programming Protocol-independent Packet Processors
PCF	Policy Control Function
PV	Packet Voting
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RU	Radio Unit
S-NSSAI	Single Network Slice Selection Assistance Information
SBA	Service-Based Architecture
SBI	Service Based Interface
SLA	Service Level Agreement
SLS	Service Level Specifications
SMF	Session Management Function
TN	Transport Network
TS	Technical Specification
UE	User Equipment

Abbreviation / Term	Description
UPF	User Plane Function
VAO	Vertical Application Orchestrator
vApp	Application-layer virtualized functions
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNE	Virtual Network Embedding
VNF	Virtual Network Function
WIM	WAN Infrastructure Manager
WV	Winfow Voting

Executive Summary

This deliverable, the D3.3 “The 6Green 5/6G Service Based Architecture”, describes the final version of the 6Green 5/6G Service Based Architecture aiming at updating and complement what was reported on D3.1 “Work-in-Progress: Enabling Green Interoperability in the 5/6G SBA” due on M12.

WP3 aims to define and implement a functional and modular 5/6G SBA architecture that is fully compatible with 6Green's “Edge Agility,” “Green Elasticity,” and “Energy-aware Backpressure” paradigms. To this end, it provides mechanisms that enable the technologies designed in WP2 to manage the lifecycle of software components and resources deployed in the 5/6G continuum and abstract them to VAO. These mechanisms support the green interoperability of stakeholders acting as a link between infrastructure and vertical application domains, which also enables the allocation and coordination of the instantiation and use of resources, data, and services across the continuum to comprehensively reduce the energy and carbon footprint. This is achieved by mapping energy-aware backpressure KPIs across the different layers of the domain, strengthening platform awareness to optimize the location and configuration of microservices across the continuum, as well as developing efficient reaction/proaction mechanisms to trigger rapid migrations of applications and slices in accordance with the edge agility paradigm, and supporting AI-driven zero-touch automation for network segments and edge resources in accordance with the policies of telecom operators and vertical application stakeholders.

Energy awareness enables a more dynamic and intent-based system, allowing the 6Green SBA to achieve fully dynamic interoperability without human intervention between stakeholders to reduce the overall energy and carbon footprint induced in the infrastructure, which is combined with separate cognitive and automated decision-making in the global network and/or within a part of the network, respectively, carried out by the ENIF, a strategic network function that has been developed and conforms to the ETSI ENI framework.

The 6Green 5/6G service-based architecture has evolved considering compliance with the three main 6Green paradigms mentioned above and taking as a starting point the information contained in deliverable D3.1.

The solution provided and described in this document ensures that the evolved and improved 6Green 5/6G service-based architecture guarantees the modularity and integrity of the information transmitted across all the different NFs that make up the final 6Green SBA, covering all the new features added to the SBA.

The document presents five different prototypes aligned with the defined NF Sets, covering the specific features added to the 6Green SBA, demonstrating the integration of the different network functions and the interactions between them.

1 Introduction

The 6Green project analyses and implements features that enable the transition to 6G technology by updating and improving the current 5G architecture, considering the 6Green paradigms of Edge Agility, Green Elasticity and Energy-Aware Backpressure providing mechanisms that meet these new challenges.

In addition to ensuring that energy efficiency is a fundamental part of the technology from the beginning, energy saving, monitoring, and optimization functions have been incorporated directly into the system.

This report presents the final version of the 6Green SBA, covering the different network functions (NF) that make up this new architecture, and it is structured into three main chapters: The first chapter presents an overview of the final architecture of 6Green 5/6G SBA, emphasizing the three fundamental pillars or provided by 6Green, the Edge Agility, Green Elasticity and Energy-Aware Backpressure. The second chapter details the different areas of research that have been carried out in WP3, covering the technologies and support mechanisms for the management and supervision of network slices, covering mechanisms for slice-sensitive data plane supervision, and the cognitive and scalable automation of segment lifecycle operations. The third chapter focuses on detailing a series of prototypes aligned with the 5 NF Sets that have been defined to form part of the final 6Green SBA architecture, where the objective is to demonstrate the integration and functionality of the components that comprise them and how they can be integrated into the overall 6Green architecture. This chapter also covers the validation and testing of these prototypes and the identification of KPIs and performance metrics where possible, allowing us to use them.

Appendix A covers the specification of the Restful API interfaces exposed by each of the NFs, as well as the input/output messages aligned with the data models specified by each of them for the transfer of information.

2 6Green 5/6G SBA Final Architecture Overview

2.1 Service-Based Architecture

As described in more detail in deliverable D3.1, the improvement of the 5/6G ecosystem is based on the deployment of the advanced capabilities of 6Green SBA network functions as a set of closely linked frameworks. These frameworks work by ingesting information from the underlying infrastructure or similar frameworks, refining and interpreting that information, and then making the resulting insights available to the rest of the system.

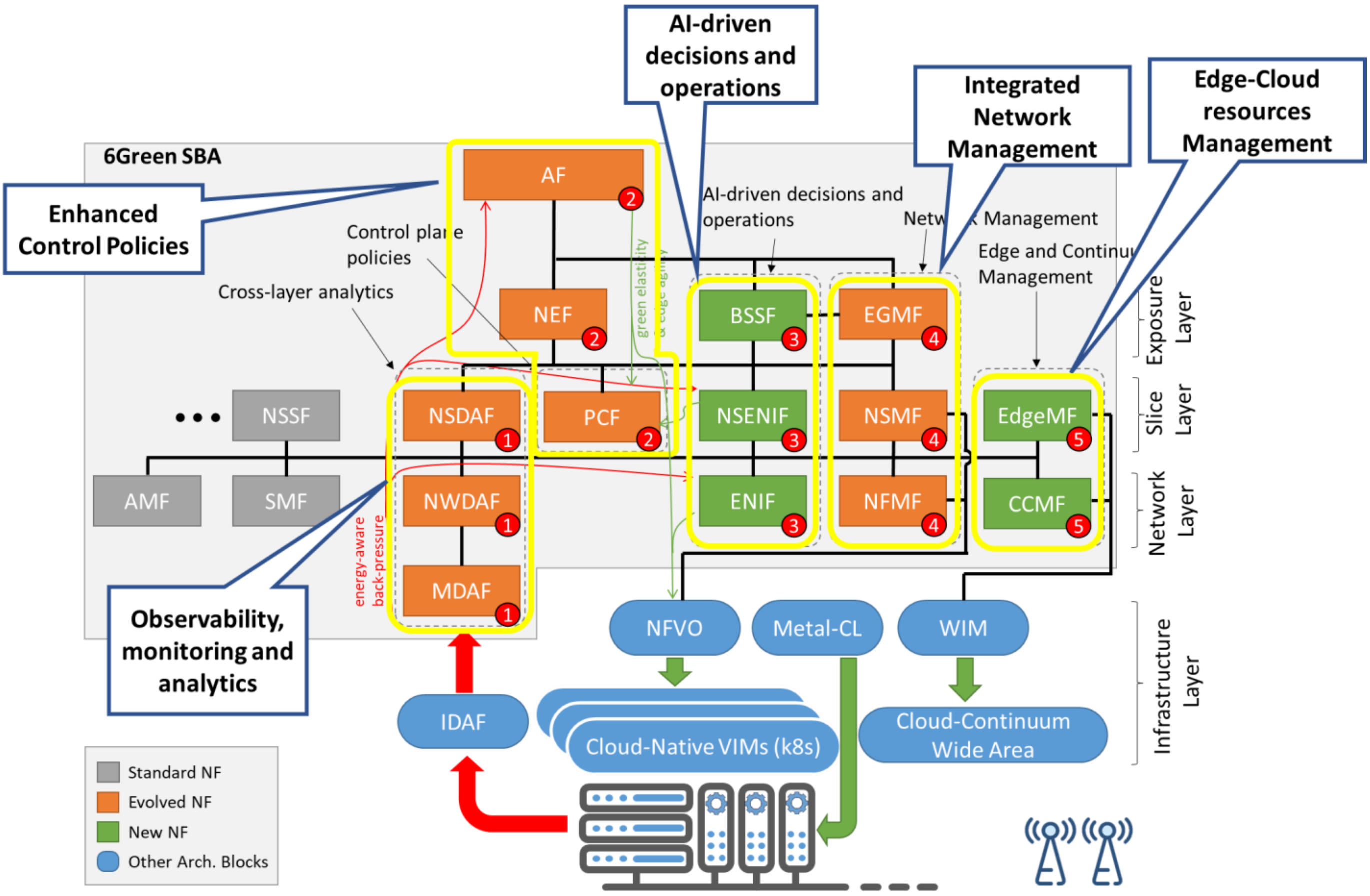


Figure 1: The 6Green SBA architecture and its component (NFs) frameworks

The final Architecture of the 6Green 5G/6G SBA, defines and implements five different frameworks, each of them focused on specific advanced features that will allow to cover the three 6Green paradigms, the Edge Agility, Green Elasticity and Energy-Aware Backpressure:

1. the enhanced observability, monitoring, and analytics framework,
2. the enhanced control policy framework,
3. the AI-driven decision and operations framework,
4. the integrated management subsystem,
5. the management framework for edge-cloud resources.

Figure 1 depicts the five frameworks that comprise the 6Green SBA architecture. One of the key elements is the framework dedicated to advanced observability, monitoring, and analytics. Its primary role is to collect raw measurements and metrics from both the infrastructure layer and other SBA-based network functions and convert this data into key performance indicators (KPIs) that provide clearer operational value and support informed monitoring and analysis.

Within this framework, several analytical functions are combined. The Management Data Analytics Function (MDAF) is responsible for collecting and analysing management-related KPIs associated with network functions. The Network Data Analytics Function (NWDAF) provides analytical capabilities that enable a preliminary correspondence to be derived between the results of the MDAF and the control plane network KPIs. In addition, the Network Slice Data Analytics Function (NSDAF) manages the analysis of data from various sources to obtain information about the performance, behaviour, and utilisation of network slices.

A second framework is focused on policy control and management, and is based on three main functions of the network. Policy Control Function (PCF) is responsible for the rules of the control plane, ensuring consistent quality of service both in individual portions and across the entire platform. Working in collaboration with it, the Network Exposure Function (NEF) serves as a bridge between the SBA and the outside world. It manages the secure exchange of data with third-party systems through its Application Function (AF), while also handling security and data abstraction. These AFs are designed to be integrated into a broader Vertical Application Orchestrator (VAO).

A third framework is responsible for decision-making based on artificial intelligence and operational intelligence. Its role is to train AI agents that can predict and adjust resource usage and traffic demand based on data from the monitoring and analysis tools covered in the first framework. 6Green implements these through ENIF, which manage the optimization of the entire network and slice segments. Finally, the Business Support System Function (BSSF) is responsible for the commercial side. It links network management with business needs, facilitating collaboration between operators and vertical partners within the framework of more flexible and sustainable business models.

The fourth framework is the integrated management subsystem, where we can observe that three main network functions are included in 6Green SBA. Each of them, the NSMF (Network Slice Management Function) and NFMF (Network Function Management Function) was designed from the ground up to manage edge agility and green elasticity. They are developed as cloud-native microservices and communicate directly with the NFVO to instantiate Network Service Instances (NSIs) through the orchestration of 5GC functions. The third part of the puzzle is called EGMF (Exposure Governance Management Function). Its role is to consume the services from other management NFs and make them securely available to untrusted Application Functions.

The fifth framework focuses on resource management at the edge and in the cloud. The goal is to extend the 6Green SBA to allow vertical applications to directly access computing power at the edge of the network. This configuration is, in fact, what makes edge agility and green elasticity possible in practice. It is based on two main functions: the Cloud Continuum Management Function (CCMF), which is responsible for coordination between different cloud domains, and the Edge-cloud Management Function (EdgeMF), which deals specifically with resources at the edge.

For a better understanding of the functionality of each of these five frameworks that make up the architecture of the 6Green 5/6G SBA, Section 3 details and implements a specific prototype for each framework, demonstrating the functionality and new features that each of them provides to the overall 6Green 5/6G architecture.

2.1.1 Edge Agility

Edge Agility represents a means to redistribute the workload across the 5/6G edge-cloud continuum in a fast and automated way that is transparent to the end-user and, of course, endorses sustainability for the whole ecosystem. This innovation is enabled through two mechanisms: scale-to-zero and live migration.

Scale-to-zero refers to the ability to turn off a pod when the application it is hosting is not in use and to resume its operation when needed. Although Kubernetes provides its own built-in scaler, called Horizontal Pod Autoscaling, it presents limits related to handling traffic spikes. The solution developed for 6Green regards the development of a UPF prototype, based on the Berkeley Extensible Software Switch[1] (BESS): when the UPF receives a packet directed towards the application deployed on the "scaled-to-zero" pod, it sends an alert to the pod asking it to scale back up and, in the meantime, it stores the incoming to give it time to get back up without packet losses.

Scale-to-zero represents the simplest case of Edge Agility, in which it is possible to maintain the service/application in the same geographical area. In case this mechanism may not be enough when, for instance, the proximity with the user needs to be revised or a different premise within the infrastructure is served by a green energy source, a migration is triggered. Thanks to the cloud-native approach adopted for the NFs and vApps developed within 6Green, migrations within the same cluster are enabled by Kubernetes itself, while migrations from one cluster to another can still be carried out by combining the scale-to-zero mechanism with a pre-deployment of the images and the sole context migration.

2.1.2 Green Elasticity

Green Elasticity entails exploiting diverse technologies to achieve the best trade-offs between consumption and performance according to the current workload and SLA/DLA. In 6Green, this innovation comes particularly in handy when applied to the UPF: in fact, the possibility of selecting a UPF based on a HW accelerator, or a pure software artifact can contribute to significant energy savings while improving the flexibility of the entire 5GS.

In practice, for the Green Elasticity innovation to work, two functionalities need to be in place: the selection of the UPF and the consequent offloading of data traffic.

The dynamic UPF selection is handled by a decision engine that leverages traffic prediction and UPF profiling to optimize energy efficiency while maintaining service performance. The first step is the UPF Profiling, a thorough characterization of different UPF implementations to understand their power consumption and performance under various traffic loads. Then, Traffic Forecasting is performed by means of a lightweight LSTM model to predict upcoming traffic loads based on historical data. Finally, a Utility-Based Decision Engine uses a utility function that considers both the predicted performance and power efficiency to choose the most suitable UPF.

Traffic offloading in 5G/6G networks refers to the process of dynamically steering data traffic to alternative pathways to optimize performance, reduce congestion, enhance the user experience, or change user plane technology.

[1] https://span.cs.berkeley.edu/bess.html

During the 6Green activities, traffic offloading has been designed and implemented in 6Green the by evolving the NEF to support moving traffic flows from one User Plane Function (UPF) to another, either to follow user mobility or to move between instances with hardware-acceleration and fully virtual implementations. Efficiency is enhanced by incorporating new input parameters in the decision that allow to move multiple UEs at a time from one slice to another.

2.1.3 Energy-Aware Backpressure

Energy-Aware Backpressure represents the very heart of 6Green, as it is the innovation allowing the three domains (e.g., infrastructure, network platform and vertical industry) to come together and cooperate towards joint sustainability goals.

The Observability Framework is in charge of collecting metrics from the infrastructure (thanks to the IDAF) as well as from the network (MDAF and NWDAF) and infer the consumption, both of power and computing resources, on a per-slice basis.

This information is made available to both the network and the vertical, but the latter receives KPIs computed by the Rating Operator. The reason for these KPIs is twofold: on the one hand, they allow to propagate only the information that the infrastructure and the network decide to expose to the outside, effectively creating a safe level of abstraction among separated stakeholders; on the other hand, it provides tailored information of interest for the demands of that specific vertical.

The metrics collected by the Observability Framework are also made available to ENIF to be used for the deployment of the application graph negotiated through the BSSF. This is where the business side meets the 5/6G ecosystem: the DLAs agreed among the stakeholders set the foundation for environmentally-aware decisions on slices initial deployments and later, thanks to the information coming from the Observability framework, to make changes to such deployments.

2.2 Technologies and Support Mechanisms for Network Slice Management and Monitoring

2.2.1 Mechanisms for Slice-Aware Data Plane Monitoring

A typical end-to-end network slice spans multiple segments of the infrastructure, including the Radio Access Network (RAN), the Core Network (CN), and the Transport Network (TN). The slice's characteristics are defined using a Network Slice Template (NST), that specifies its behaviour and resource requirements. Each NST can be linked to one or more Service Level Specifications (SLSs), which define expected Key Performance Indicators (KPIs), such as latency, availability, or monitoring granularity. These SLSs are then formalized into a Service Level Agreement (SLA), which specifies the contractual obligations between the network operator and the tenant, including the penalties for SLA violations.

Continuous and fine-grained monitoring is essential to detect SLA violations in real time and to ensure accountability. Traditional monitoring approaches, based on periodic probing or flow-level statistics, are not well suited for high-speed, low-latency environments like the 5G TN, due to their significant overhead and poor scalability. In recent years, programmable data plane technologies, such as P4[2], and In-band Network Telemetry (INT), have offered new opportunities for fine-grained, real-time monitoring. INT enables programmable switches to embed telemetry metadata directly into the packet headers as the packets traverse the network. Several research efforts have leveraged this capability to monitor network performance and identify anomalies. However, INT-based solutions face several limitations. First, labelling every packet can result in excessive overhead and reduced throughput, especially under high traffic volumes. Additionally, fixed and periodic labelling strategies lack adaptivity since they collect telemetry data regardless of possible SLA violations. Second, the use of port- and flow- based sampling to monitor the performance of network slices may introduce inefficiencies since the monitoring needs of different slices may differ even on the same port. Therefore, there is a need for intelligent, selective monitoring frameworks that can dynamically trigger telemetry collection based on early indications of performance degradation or SLA violations.

To deal with the above limitations, we have proposed SliceMon, an efficient slice-aware SLA monitoring frame- work within the TN segment. SliceMon leverages data plane programmability and an ensemble-based learning architecture to detect potential SLA violations. Rather than labelling every packet, SliceMon embeds lightweight weak learners into switches. These learners are trained using only local data plane observations and make inferences on whether an SLA violation may be occurring. Their predictions are sent to the controller, which aggregates them using ensemble methods and, only, when necessary, triggers the collection of fine-grained telemetry data to confirm the SLA violation. In the following, we describe the architecture, the design principles, and the inference methodology of SliceMon, considering the limitations of P4-capable network devices. We also have also developed an emulated environment based on Mininet[3] and BMv2[4] P4 to build a custom dataset to train and test SliceMon for a delay-based SLA. Then, we compare several controller aggregation strategies to combine individual weak learners' predictions, including majority voting and more sophisticated ML-based models, evaluating the system performance in terms of accuracy, overhead, and responsiveness metrics.

[2] https://p4.org/

[3] https://mininet.org/

[4] http://bmv2.org/

2.2.1.1 SliceMon Architecture and Design Principles

SliceMon has a twofold goal: providing native support for slice-aware monitoring tasks and minimizing monitoring overhead while promptly detecting SLA violations. SliceMon's functional architecture is illustrated in Figure 2. The idea is to combine two operating modes: *i) inference*, where lightweight machine learning inference (MLI) is performed directly in the data plane; and *ii) monitoring*, which enables selective, on-demand, fine-grained telemetry data collection. By default, SliceMon runs in inference mode: each switch executes per-packet MLI tasks to detect potential SLA violations locally. Upon detecting a violation, the switch sends an alert to the controller. The controller aggregates alerts from multiple switches and assesses whether a global SLA violation is likely. If an SLA violation is confirmed, the controller transitions to monitoring mode by issuing a command to network switches to activate detailed telemetry collection. Once the SLA violation is resolved, the system returns to inference mode. The transition logic is dependent on the monitored slice's SLAs.

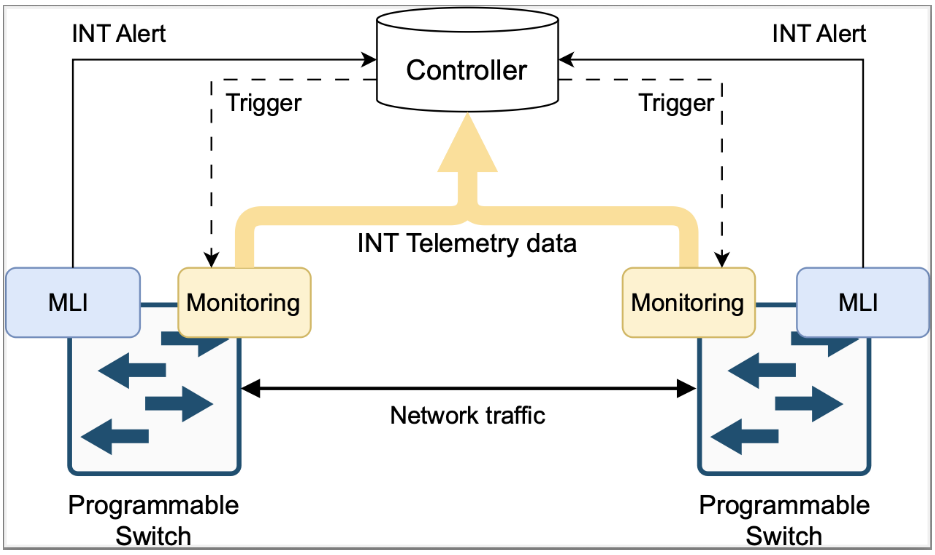


Figure 2: Functional view of the SliceMon architecture

The design of our scheme is guided by the following key principles: *i) Data-plane MLI for selective probing* - To minimise monitoring overhead, MLI is performed at the per-packet level within the data plane of programmable switches. This enables the early identification of potentially anomalous conditions and allows the centralised controller to trigger the monitoring mode on the network nodes only when necessary. i*i) MLI compatible with programmable switches constraints* - The ML models deployed on the switches must conform to the stringent limitations of programmable network devices including limited memory, computational capabilities, and deterministic processing pipelines. Moreover, the limited expressiveness of the P4 language, such as the lack of support for operations like division, imposes additional restrictions on the complexity of models that can be implemented directly within the data plane. *iii) Local-feature MLI design* - To maintain scalability and avoid introducing communication overhead, the base learners running on each switch are trained using only locally available features. No inter-switch or switch- to-controller communication is required during inference to exchange features.

To satisfy these design goals, we adopt an ensemble-based architecture where MLI models, known as weak learners, are deployed on switches, and an aggregation model residing at the controller combines their predictions. We envision the following split of responsibilities between the data plane and the control plane of the TN.

Control plane. It performs three main functions: *i)* it trains the base learners and deploys them as P4 programs onto the programmable switches; *ii)* in inference mode, it receives alerts from switches and, upon determining that a potential SLA violation is occurring for a target slice, it transitions to monitoring mode to initiate detailed telemetry collection; *iii)* in monitoring mode, it receives and analyses telemetry data to confirm the presence of an SLA violation for the target slice, and once it is resolved, it switches back to inference mode.

Data plane. Slice-specific agents are deployed on each switch that forwards traffic belonging to the target slice. Per 3GPP standards, each network slice is identified by an S-NSSAI (Single Network Slice Selection Assistance Information). The UE and the network functions use the S-NSSAI to associate packets with the appropriate network slice. We assume the S-NSSAI is explicitly carried within the packet header using a dedicated field (e.g., MPLS, VXLAN). Upon the arrival of a packet matching a specific slice ID, the switch behaves according to the current operational mode. In inference mode, the MLI task trained for that slice is executed, producing a boolean output that indicates whether a potential SLA violation is detected. If so, the switch appends an alert to the packet, an INT header containing the switch and port identifiers where the SLA violation was detected. As in traditional INT, when a packet containing INT information reaches the egress point of the network, data is forwarded to the controller for further analysis. In monitoring mode, triggered by the controller, the switch appends fine-grained telemetry data as INT headers to every packet associated with the target slice.

2.2.1.2 Inference Methodology

We consider a TN composed of M - enabled switches. For each switch k , we assume a local training dataset $D_k = \{(x_1^k, y_1^k), (x_2^k, y_2^k), \dots, (x_{s_k}^k, y_{s_k}^k)\}$, where s_k is the number of data points, $x_i^k \in \mathbb{R}^p$of data is a p-dimensional feature vector associated with a packet reception, and $y_i^k \in \{0,1\}$ is the label indicating whether the feature vector belongs to the target class. In this work, the target class corresponds to the occurrence of an SLA violation condition for the slice to which the packet belongs. Later in this chapter we detail the specific type of SLA violation considered in the evaluation.

Each local dataset D_k is used to train a weak learner $F_k(x)$. The choice of the base model for these weak learners depends on the use case and the ensemble method employed. For our purposes, we adopt shallow (low-depth) decision trees due to their fast training and inference capabilities, crucial for line-rate execution on programmable switches. Moreover, decision trees implement a simple logical structure easily mapped onto the constrained environment of P4 switches. A decision tree classifier can be expressed as a sequence of `if` statements in P4, requiring only the storage of threshold values at each decision node. A binary decision-tree classifier of depth l (i.e., the maximum number of edges from the root node to any leaf node), requires at most $2^{l+1} - 1$ `if` statements on a P4 switch. To enable per-slice monitoring, each switch must implement a dedicated decision tree for every slice whose traffic it forwards.

It is important to note that the controller performs an additional classification step on the alerts received from the switches to determine whether an actual SLA violation is occurring. This classification can be based on various criteria and may utilize algorithms of varying complexity. In this work, we consider approaches based on voting and ML-based classifiers, as detailed following sections.

2.2.1.3 Experimental Setup and Dataset Creation

We here describe the experimental setup to collect the dataset for training and validation, and the features extracted for model inference.

Experimental setup

To the best of our knowledge, no publicly available dataset provides per-slice traffic measurements within TNs. We thus generated a dataset using the widely adopted network emulation environment Mininet combined with BMv2 P4 software switches[5]. Mininet was deployed on a single machine equipped with dual Intel Xeon Silver 4410T processors (2.7 GHz, 20 cores, 40 threads), 1 TB of RAM, and Ubuntu 22.04 OS.

We adopt a typical Fat-Tree topology composed of four Spine switches, four Leaf switches, four ToR switches, and eight hosts. Traffic is generated by six hosts (senders), while the remaining two act as receivers. Each sender transmits traffic to each receiver. For clarity in result interpretation, we consider a traffic scenario with two network slices: `Slice A` and `Slice B`. `Slice A` models a network slice tailored for real-time applications demanding low latency and high reliability. In our experiments, the SLA for `Slice A` requires that 90% of its traffic experiences an end-to-end delay (from ingress to egress in the TN) below 20 ms. `Slice A` serves as the target network slice for our slice-aware telemetry system, and its SLA violation events are used later to label the dataset accordingly. On the contrary, `Slice B` represents a best-effort network slice providing a baseline service level to bulk traffic, without explicit QoS guarantee. Traffic for `Slice A` is generated by just one of the senders, which produces flows with a constant payload size of 1500 bytes and a fixed packet rate of C_A Kbps. On the contrary, the other five senders generate traffic for `Slice B`, which is best effort traffic characterised by a highly variable, bursty pattern. More precisely, time is divided into slots of random duration d_B, independently sampled for each flow. Within each slot, transmission occurs with probability $p = 0.2$. If transmission is triggered, a mean data rate c_f is drawn uniformly from the interval $[0, C_B]$. During the active slot, packets are generated following an exponential inter-arrival time distribution with mean $1\ /c_f$. The data rate for links between switches is set to 1 Mbps, both to facilitate the generation of SLA violations for `Slice A`, and to prevent packet losses caused by buffer overflows or CPU saturation on the Mininet host machine. Since Mininet's default interface rate limiting can produce non-deterministic or unrealistic results, we configure the BMv2 switch parameter `queue_rate` to control the maximum output rate of queues. A remote controller is implemented as a separate process to enable programmatic, runtime configuration of switch behaviour.

Dataset creation

To demonstrate the general applicability of SliceMon under varying traffic conditions, we defined three distinct traffic scenarios. In the first scenario (Scenario 1), `Slice A`'s flow rate is fixed at $C_A = 100$ Kbps, while the slot duration for `Slice B` traffic follows a uniform distribution $d_B = U(1{,}2)$ seconds, with a rate $C_B = 500$ Kbps. This setup generates short bursts of disturbance traffic from `Slice B`. The second scenario (Scenario 2) differs only in the slot duration, which is increased to $d_B = U(3{,}4)$seconds, resulting in longer disturbance bursts from `Slice B`. This allows evaluation of the system's behaviour under more prolonged congestion. In the third scenario (Scenario 3), the longer slot duration from Scenario 2 is maintained, but the `Slice A`'s flow rate is increased to $C_A = 400$ Kbps. This introduces greater competition for bandwidth between Slice `A` and `Slice B`, providing insight into performance under higher load conditions. In all scenarios, Slice `A` traffic is generated for 350 second. The `Slice B` traffic starts 10 seconds after `Slice A` and continues for 300 seconds, creating an initial interval during which `Slice A` flows do not share resources with the best-effort traffic. Each scenario is executed over ten independent runs to ensure statistical significance.

[5] https://github.com/p4lang/behavioral-model

To train the weak learners of the SLA violation classifier, we extract a set of telemetry features from the Mininet- emulated network prototype on a per-packet basis for all switches traversed by `Slice A` traffic. The selected metrics are designed to capture queuing behaviour and delay-related performance. For each packet, the following telemetry metrics are extracted:

- *Traversal Time*: the total time a packet spends inside a switch, computed as the difference between the egress and ingress timestamps. This value captures per-hop forwarding delay.
- *Enqueue Depth*: the length of the ingress/egress queue when the packet is enqueued. This reflects the level of congestion when the packet enters the queue.
- *Dequeue Depth*: the length of the same queue when the packet is extracted for transmission. Comparing enqueue and dequeue depths can provide insights into dynamic queue occupancy trend.
- *Queueing Delay*: The time spent by the packet waiting in the queue before being transmitted.

To construct the feature vector for SLA violation classification, we compute moving averages over the per-packet telemetry metrics described above. For each metric, we calculate rolling averages over exponentially increasing window sizes defined as $S = 2^n$, with $n \in \{0,1, \dots ,10\}$. This yields 11 moving averages per metric, including the case $S = 1$, which corresponds to the raw metric value. The use of powers-of-two window sizes is motivated by practical implementation constraints in P4. Since the P4 language lacks native division operations, averages overpowers of two window sizes can be efficiently computed via bitwise right-shift operations, thereby enabling low-complexity computations within the data plane. Additionally, maintaining rolling buffer of size 1024 packets, corresponding to the largest window size, is sufficient to support all required moving averages per metric, ensuring an efficient, bounded-memory implementation. For each packet i observed at switch k , the resulting feature vector x_i^kcomprises eleven moving averages for each of the four metrics described earlier, resulting in a total of $p = 44$ features per packet.

The final dataset comprises a feature vector for each Slice A packet at every switch it traverses. As detailed in following sections a local decision tree classifier is independently trained for each switch using only the data collected at that switch. The classifiers are limited to a maximum depth of $l = 3$ to ensure interpretability and suitability for deployment in resource- constrained environments. To generate the binary target label for training, we derive the SLA violation condition from the end-to-end delay experienced by each `Slice A` packet. For each packet received at both slice endpoints, we verify whether its end-to-end delay exceeds a threshold of 20 ms. Subsequently, a moving average over a 1-second window is computed for the fraction of packets exceeding this threshold. If this average surpasses 0.1, the corresponding packet is labelled as an anomaly (target class $y = 1$); otherwise, $y = 0$. It is important to note that although each switch observes a distinct local feature vector for the same packet, the target label y is shared across switches and determined by the global SLA violation condition. For each experimental scenario, the dataset aggregates data from all ten independent runs. Classifier evaluation is performed via 10-fold cross-validation, where in each fold, one run is used as the test set and the remaining nine as the training set. Final performance metrics are computed by averaging results across all ten folds to ensure statistical robustness.

2.2.1.4 Evaluation

We first introduce the benchmark and baseline methods employed for performance comparison, then we present the SliceMon numerical results across various performance metrics.

Baselines and benchmark methods

We evaluate five distinct aggregation mechanisms, with different levels of implementation complexity, which are used to compute the final classification outcomes of the weak learners' inference. Two of them are based on a majority-voting approach. The first, called Packet Voting-α (PV-α), infers an SLA violation if the proportion of INT entries within a single packet, relative to the total number of switches traversed, exceeds the threshold α. The second strategy, termed Window Voting-α (WV-α), is slightly more sophisticated: it calculates the fraction of INT entries across all packets traversing the switches during a rolling time window of 100 ms. An SLA violation is declared if this aggregated fraction exceeds the threshold α. The remaining three strategies employ a meta-classifier to combine the predictions of the base learners. We consider Random Forest (RF), eXtreme Gradient Boosting (GB), and Multi-Layer Perceptron (MLP). To train the meta-classifier, we construct the dataset D_a as follows. Let *n* be the total number of packets collected for the target slice. Each packet i is associated with a training sample $(x_i^a, y_i^a)$where x_i^a is a feature vector of size m_a, corresponding to the total number of switches traversed by `Slice A` traffic. Each feature in takes values from the set $\{-1,0,1\}$, where -1 indicates that the packet did not traverse the corresponding switch, 0 denotes no anomaly detected by that switch for packet i, and 1 indicates that an anomaly was detected. The label $y_i^a \in \{0,1\}$ identifies whether packet i belongs to the target class, i.e., whether it corresponds to an SLA violation, as defined in the following.

The benchmarks for comparison are based on INT-label. The first benchmark, INT-L, performs the classical interval-based labelling of INT-label on all network switches, using a labelling interval T of 50 ms. Prior work has demonstrated that this interval is sufficient to collect telemetry data from all network ports every 100 ms. The second benchmark, called INT-LA, employs the same labelling method as INT-L but restricts its operation to only those switches traversed by the traffic of `Slice A`.

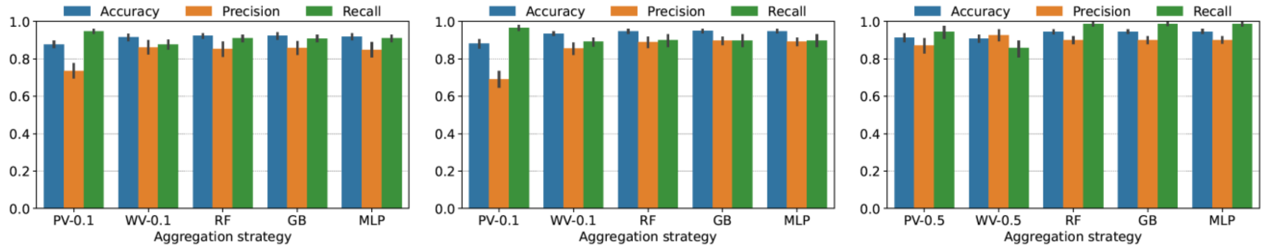
Results

We consider three categories of performance metrics. The first category evaluates the effectiveness of the SLA violation detection model using accuracy (the proportion of correct inferences over all predictions), precision (the proportion of correctly identified positive instances among all predicted positives), and recall (the proportion of positive instances correctly detected). The second category quantifies the over- head introduced by SliceMon relative to INT-L and INT-LA, measured by the total volume of INT metadata bytes transmitted. The third category assesses the responsiveness of the SLA detection system.

1) Evaluation of SLA violation classification performance: To compute the three metrics described above, we consider both operational modes of SliceMon: inference and monitoring. In the monitoring mode, we assume 100% accuracy since all telemetry data is forwarded to the controller. For the same reason, we assume 100% accuracy for INT-L and INT-LA.

Table 1: Evaluation of SliceMon voting strategies

		Scenario 1				Scenario 2				Scenario 3			
Strategy	α	Accuracy	Precision	Recall	Score	Accuracy	Precision	Recall	Score	Accuracy	Precision	Recall	Score
PV	0.1	**0.88**	**0.73**	**0.95**	**1.72**	**0.88**	**0.69**	**0.97**	**1.71**	0.79	0.69	1.00	1.64
	0.3	0.87	0.87	0.71	1.66	0.89	0.87	0.68	1.67	0.85	0.76	0.99	1.73
	0.5	0.84	0.94	0.52	1.57	0.86	0.95	0.46	1.57	**0.91**	**0.87**	**0.94**	**1.82**
	0.7	0.78	0.99	0.31	1.43	0.82	0.99	0.29	1.46	0.89	0.96	0.79	1.77
	0.9	0.72	1.00	0.09	1.27	0.79	1.00	0.16	1.37	0.80	0.99	0.57	1.58
WV	0.1	**0.92**	**0.86**	**0.88**	**1.79**	**0.94**	**0.86**	**0.89**	**1.82**	0.80	0.69	1.00	1.65
	0.3	0.85	0.97	0.55	1.61	0.88	0.97	0.55	1.64	0.88	0.79	0.98	1.77
	0.5	0.79	0.99	0.34	1.45	0.83	1.00	0.34	1.50	**0.91**	**0.93**	**0.86**	**1.81**
	0.7	0.75	1.00	0.19	1.35	0.80	1.00	0.22	1.41	0.81	0.99	0.58	1.60
	0.9	0.70	1.00	0.06	1.23	0.78	1.00	0.12	1.34	0.65	1.00	0.22	1.26



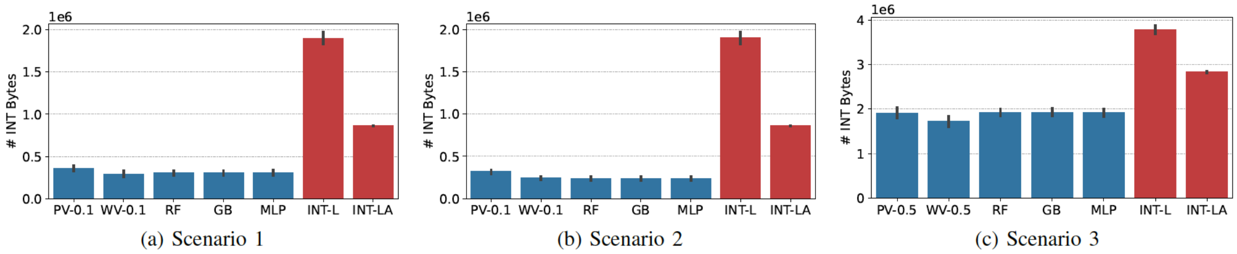
(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 3: Comparison of SliceMon aggregation strategies

First, we perform a sensitivity analysis to determine the voting threshold α that yields the best performance across the three scenarios. We evaluate threshold values $\alpha = \{0.1, 0.3, \dots, 0.9\}$. We highlight that, from an efficiency perspective, a high recall is critical to ensure that most SLA violations are correctly identified. A lower precision (i.e., a higher number of false positives) is less critical for inference accuracy, but it increases overhead by triggering unnecessary INT metadata collection. To balance these considerations, we define a combined score $s = accuracy + \beta\, precision\ + (1-\beta) recall$, which is computed for each threshold and scenario. In the rest of the analysis, we select, for each scenario and voting strategy, the threshold α that maximises this score. We set $\beta = 0.5$ to assign equal weight to precision and recall.

Summary results are presented in Table 1. We first analyse the PV strategy. In the first two scenarios, the model shows good accuracy, ranging approximately from 0.7 to 0.9 across all voting thresholds. As the voting threshold α increases, precision improves, but the recall sharply decreases to zero. This indicates that, while the model is very accurate when predicting SLA violations, it misses most of them. The threshold $\alpha = 0.1$ yields the highest overall score in these two scenarios. The third scenario shows a different behaviour. Here, accuracy is the highest at a voting threshold of $\alpha = 0.5$, which provides a balanced trade-off between precision (0.87) and recall (0.94), resulting in the highest score for this scenario. Similar trends are observed for the WV aggregation strategy, although its performance is generally slightly better than that of PV. These findings suggest that no fixed α threshold performs optimally across all scenarios. Instead, the voting threshold should be adaptively tuned based on traffic intensity.

We now include the three ML-based aggregation strategies in our analysis. For a fair comparison with the voting strategies, we use the α values identified earlier. Figure 3 presents the results of this comparison. First, it must be noted that the three ML approaches obtain very similar performance across all scenarios, generally outperforming the voting strategies. While the voting strategies deliver comparable results and offer the advantage of simpler implementation, which may be preferable for line-rate deployment, they must be carefully tuned to the specific traffic scenario. This may hinder practical applicability in dynamic scenarios.



(a) Scenario 1

(b) Scenario 2

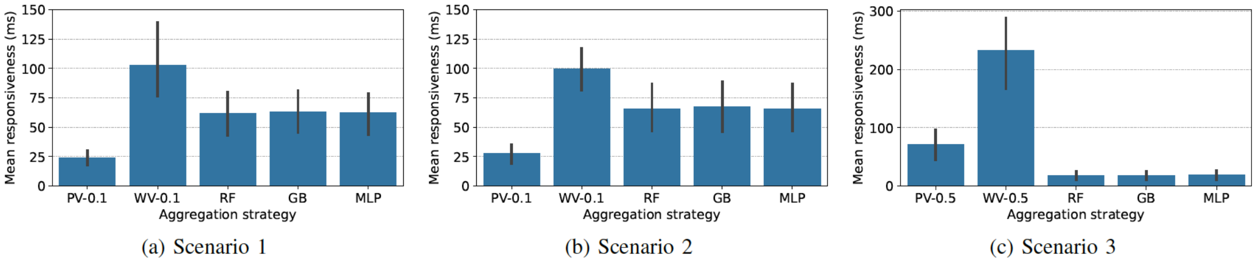
(c) Scenario 3

Figure 4: Comparison between SliceMon aggregation strategies (in blue) and INT-label (in red)

2) Analysis of the overhead: We now evaluate the data plane overhead introduced by SliceMon in terms of bandwidth consumption. To quantify the overhead of the INT-L and INT-LA baselines, we simulate the INT-label mechanism on our experimental dataset. For each packet encapsulating INT metadata, we assume an INT header size of 20B, which includes the Generic Routing Encapsulation (GRE) required to carry INT information between layers 3 (IP) and layer 4 (transport). For every switch traversed by the packet, we account for: 8B to encode the switch identifier and port, 4B for reporting the total number of packets observed during the label interval T, and a list of metric tuples, where each tuple consists of four 4B fields capturing the telemetry associated with each packet observed during T . For instance, if a switch processes three packets during a given interval T , the corresponding metadata includes 8B for the switch ID and port, 4B for the packet count, and $3 \times (4 \times 4B) = 48B$ for the telemetry metrics, yielding a total of 60B of metadata added by that switch. This metadata allows the controller to reconstruct the same level of per-switch telemetry data as provided by SliceMon.

To compute the overhead introduced by SliceMon, we distinguish between the inference and monitoring modes. In inference mode, telemetry processing is performed locally at the switch, and only packets potentially associated with SLA violations carry metadata. For these, we assume a 20B INT header, and 8B per switch for the switch identifier. No additional telemetry metrics are required, as no raw data is exported. In monitoring mode, SliceMon behaves as conventional INT. Thus, each packet includes a 20B INT header, and for each switch along its path, 8B for identification and four 4B fields (16B total) to encode the collected metrics. It is worth noting that our analysis does not account for potential packet fragmentation.

Results are illustrated in Figure 4. In Scenarios 1 and 2, where `Slice A` traffic intensity is lower, the five SliceMon aggregation strategies yield comparable overhead levels. Among them, PV strategy provides the highest overhead, approximately 20–30% greater than the others. Most importantly, all SliceMon strategies introduce substantially lower overhead than the INT-L and INT-LA baselines. The reduction in overheads relative to INT-L ranges from 80% to 87%, while compared to INT-LA, the reduction lies between 60% and 70%. In Scenario 3, characterized by higher Slice A traffic and consequently a greater number of SLA violations, the WV strategy with α = 0.5 achieves the lowest overhead among the SliceMon variants—approximately 10% less than the others. Even under this more demanding traffic condition, SliceMon maintains a clear advantage: its overhead remains 50–55% lower than INT-L and 33–40% lower than INT-LA. The increased overhead observed in this scenario is explained by noting that the higher number of SLA violations induces more frequent activations of the monitoring mode.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 5: Responsiveness of SliceMon aggregation strategies

3) *Responsiveness*: Responsiveness is defined as the delay between the occurrence of an SLA violation and its detection by the system. To compute it, we record the arrival time at the receiver, denoted by t_r, of the first packet for which the ground-truth label is $y = 1$. We then identify the time of detection t_i, defined as the arrival time of the first packet that causes the system to infer an SLA violation. Responsiveness is computed as $r = t_i - t_r$. This metric does not account for processing delays at the controller. For the INT-L and INT-LA baselines, we assume a fixed telemetry collection interval of 100ms, as per the INT label design. Under this assumption, the time of detection is uniformly distributed within this interval, yielding an expected mean responsiveness of 50ms.

Figure 5 presents the mean responsiveness of SliceMon aggregation strategies. In Scenarios 1 and 2, the results are comparable. The PV-0.1 strategy achieves the lowest responsiveness, approximately 25ms, benefiting from per-packet inference. The WV-0.1 strategy exhibits a mean responsiveness of approximately 100ms, which is coherent with the window size used for aggregating the local switches' predictions. The ML-based strategies yield intermediate responsiveness values, averaging around 62ms. Scenario 3 shows very different trends. Both voting-based strategies experience increased responsiveness: approximately 70ms for PV-0.5 and 220ms for WV-0.5. Conversely, the ML-based strategies show improved responsiveness, achieving a mean of approximately 18 ms. This suggests that the higher volume of monitored traffic in Scenario 3, resulting in greater congestion and more informative feature vectors, improves the early detection capability of learning-based methods. It is worth noting that, in each scenario, at least one SliceMon aggregation strategy outperforms the INT-L and INT-LA baselines in terms of responsiveness, although the best-performing strategy varies depending on the scenario.

Final remark: We consider SliceMon as a first step toward demonstrating the potential of intelligent, on-demand telemetry in slice-based networks. While SliceMon shows promising results in accuracy, responsiveness, and overhead reduction, several avenues remain for improvement and extension. Future work will focus first on implementing a P4-based prototype of SliceMon and evaluating it on physical testbeds or large-scale emulations. We plan to investigate adaptive aggregation mechanisms that dynamically select the optimal strategy in response to traffic patterns and SLA profiles. To support real-time model updates, continual learning approaches will be considered. Finally, we intend to extend the SliceMon architecture to support multitenant and multi-domain environments, where SLA enforcement must be coordinated across administrative boundaries.

2.2.2 Cognitive and Scalable Automation of Slice Lifecycle Operations

ENIF plays a central role in the cognitive and scalable automation of slice lifecycle operations within the 6Green SBA architecture. As the main component responsible for integrating intelligence into network management and service orchestration, ENIF ensures that the entire slice lifecycle—ranging from creation and updates to re-optimization—is managed efficiently and autonomically.

In the context of slice lifecycle automation, ENIF takes a slice intent provided by BSSF, which includes detailed requirements for computing, networking, and QoS, as well as any location-specific constraints for each application component. ENIF then leverages the available infrastructure data from EdgeMF/CCMF to formulate and solve a virtual network embedding (VNE) problem, using a mixed-integer linear program (MILP) to determine an optimal deployment plan for the vApp.

This optimization process is key to enabling scalable automation, as ENIF's decision-making engine is able to autonomously allocate resources across computing nodes and network elements based on real-time conditions and predefined goals. It dynamically adjusts to the requirements of each vApp component, producing a deployment plan that specifies resource allocations, node assignments, and network bandwidth, ensuring optimal performance across the slice.

Furthermore, ENIF is not only responsible for initial placement but also for continuous slice lifecycle management. Through its built-in re-optimization mechanism, ENIF can trigger resource scaling or migration when the performance of a vApp component is impacted by infrastructure degradation or changes in component needs. This adaptive capability ensures the scalability and flexibility required for maintaining the quality of service (QoS) throughout the lifecycle of the slice.

The integration of functionalities provided by other components of the 6Green SBA, such as EdgeMF/CCMF for infrastructure data and NWDAF/NSDAF for historical and forecast data, allows ENIF to make informed decisions and to optimize slice operations effectively. By solving the VNE problem, ENIF ensures that resources are allocated efficiently across the infrastructure, enabling cognitive and scalable automation in network slicing.

The detailed formulation of how slice creation and slice updates are expressed as mixed-integer linear programs (MILP) is presented in the following sections, providing the mathematical foundation for ENIF's optimization and decision-making processes.

3 6Green Service-Based Architecture NFs Prototypes

This section focuses on defining prototypes/use cases to demonstrate specific NF Set integrations (a preliminary step prior to the final integration of the NF Sets that will be carried out in the use cases for the validation of the 6Green SBA architecture in WP5). It presents a description of the prototypes, and the NFs involved. To demonstrate the integration of NFs from the NF Sets, some prototypes also integrate NFs from other NF Sets.

These are the NF Sets and their corresponding prototypes:

- NF Set-1: Cross-Layer Analytics Extension [MDAF/NWDAF/NSDAF]
  - Network Slice Energy Consumption Monitoring Prototype
- NF Set-2: Control Plane Policies and Network Capability Exposure NF Set Extension [PCF//NEF/AF]
  - Offloading network slice traffic to nearby edge nodes Prototype
- NF Set-3: AI-Driven Decisions and Operations [BSSF/ENIF]
  - Network Slice Intent Creation Process Prototype
- NF Set-4: Network Management [EGMF/NSMF/NFMF]
  - Network Slice Life Cycle (Creation, modification and deletion) Prototype
- NF Set-5: Management Framework for Edge-Cloud Resources [EdgeMF/CCMF]
  - Network Slice Intent Infrastructure Offering Prototype

3.1 Network Slice Energy Consumption Monitoring (NF Set 1)

Network slice is the central component for energy efficiency implementation in 6Green, allowing creation of multiple virtual networks on shared physical infrastructure (Cloud native or K8s environment). In the 6Green context, energy awareness is integrated into the network slicing paradigm through several innovative mechanisms, consisting of cross-domain observability mechanisms and then applying analytics to evaluate the energy consumption of specific vertical applications. Steps implemented during NF Set 1 phases:

- Collecting and analysing energy consumption metrics from the infrastructure.
- Mapping these metrics to individual network slices and their components.
- Providing feedback about energy usage patterns.
- Enabling slice-specific energy optimization decisions.

The Network Slice Energy Monitoring is supported by the integration and interaction of the following components:

- Network Slice Data Analytics Function (NSDAF), that estimates the energy of network slices, including the contribution of edge-cloud resources.
- Management Data Analytics Function (MDAF), that collects energy KPIs from infrastructure resources, providing a breakdown of energy consumption against various subsystems, including network slices
- Network Data Analytics Function (NWDAF), that acquires and infers control-plane data relevant to drive energy-efficient operations, including workload per slice.

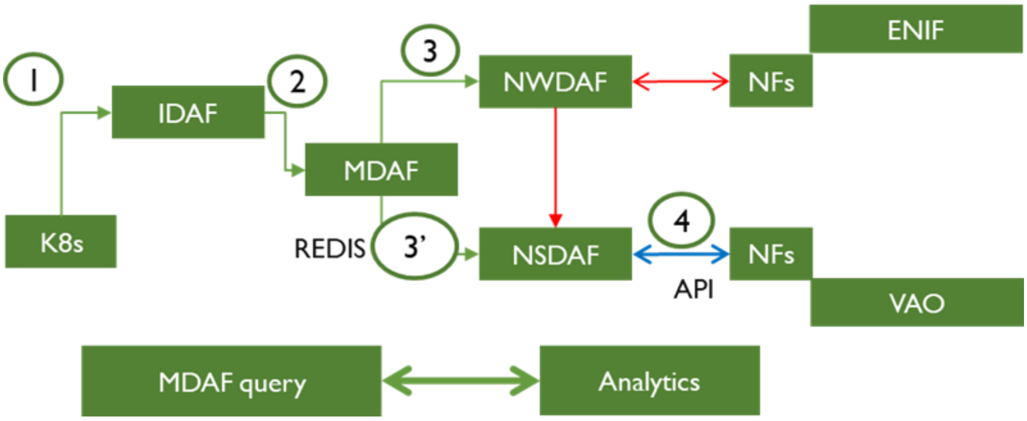


Figure 6: NF SET 1 components and interactions

The system architecture presented in Figure 6 shows the NF interactions within the NF Set 1 but also with other NFs and infrastructures. The Collection and Analysis infer KPIs provided by NSDAF, NWDAF and MDAF and estimates the energy of network slices through the analysis of data collected from infrastructure and metrics of network slices, tracking resource usage and impact associated per slice. By collecting per slice, the energy consumption data may be predicted for the future energy consumption, enabling the system prediction capabilities leveraging on predictions models. The prediction is implemented using a Flask web service with two main endpoints, one for retrieving current historical data and one for obtaining future predictions. This model supports different ML models for flexibility and adaptation to data characteristics.

NF Set 1 delivers integration points with the other's system components, it provides REST API calls for ENIF/VAO, enables both historical insights and future prediction and supports additional analyses like carbon footprint estimation by applying conversion factors to energy data. In depth, there are several steps of development, where the system collects infrastructure and power data from Prometheus and Redis, identifying containers via unique slice_id. ARIMA model (p=5, d=1, q=1) forecasts resource and power consumption by calculating daily energy averages. Forecasting extends up to 28 days ahead, combining autoregressive components, differencing, and moving average elements to model temporal dependencies effectively.

The implementation is crucial for network slicing as it enables energy-aware management by providing slice-level energy metrics and predictions, supporting carbon footprint calculations per slice and integrating with orchestration systems for energy-efficient operations. There is a technical implementation of labelling, extended to the K8s environment, where PODs/containers have a label such as slice-id: <unique-identifier>, allowing the system to filter and group containers by their associated network slice. The slice labelling configuration is an internal mechanism that happens during the deployment of network functions and slice components. The labels are visible to monitoring systems, which collect infrastructure data and which allow metrics to be aggregated and analysed on a per-slice basis.

The NF slice APIs interaction capabilities are:

- Slice ID Discovery:
    - The GET /slice_ids endpoint returns a list of all available slice IDs in the system.
    - This allows clients to discover which slices are currently deployed and monitored.

```
json
{
  "slice_ids": [slice_id_1_string, slice_id_2_string, etc],
  "error": Boolean,
  "error_details": None/String
}
```

- Slice-Specific Queries:
    - The other endpoints (/current_power_microwatts, /predicted_power_microwatts, /cpu_usage) all accept a slice_id parameter.
    - This allows clients to request data for specific slices by their unique identifier.

POST /current_power_microwatts, returns historical energy consumption data:

```
json
// Input
{
  "slice_id": String,
  "nr_of_days": Integer
}

// Output
{
  "data": [(date_string, value_float), .....],
  "timestamp": String,
  "error": Boolean,
  "error_details": None/String
}
```

POST /predicted_power_microwatts, provides forecasting of future energy consumption:

// Input/Output structure similar to *current_power_microwatts*

3.2 Network Exposure and Slice Traffic Offloading to Nearby Edge Nodes (NF Set 2)

Within the context of NF Set 2, the most relevant component currently being prototyped is the NEF. This component was briefly introduced in D3.1 and further detailed in D3.2, with a focus on its interactions with other NFs in the SBA.

This section provides an overview of the NEF functionality and interactions, with a particular focus on recent advancements in the Slice Offloading feature, an enabler described in D2.3. The second part of the section will address the Event Exposure service, which is offered indirectly through the AMF prototype. The final interaction of NEF with the target 5GC is shown in Figure 7.

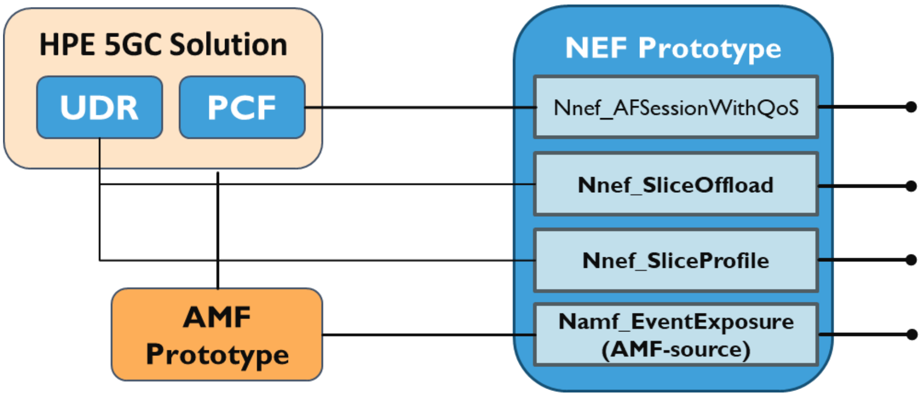


Figure 7: NF SET 2 components and interactions

Slice Offloading and Slice Profile

Within the SBA, the NEF acts as enabler for switching traffic to a different slice for a specific single UE or a set of UEs, based on certain conditions analysed by ENIF. In the 6Green use cases, the Slice Offloading functionality allows traffic to switch through different UPFs. The reasons for triggering a User Plane change fall under broader 6Green operations, such as Edge Agility and Green Elasticity, and may include the following:

- The new UPF is more energy-efficient.
- The new UPF leverages hardware acceleration.
- The new UPF is powered by renewable energy sources.
- The new UPF has lower resources utilization compared to the current one.
- The path with the new UPF is less congested than the current one.
- The new UPF belongs to the new selected slice.

ENIF is the component candidate for directly making the slice offload request to the NEF, by specifying the following body structure:

```
{
  "afServiceId": "string",
  "nfId": "string",
  "gpsi": "string",
  "supi": "string",
  "externalGroupId": "string",
  "dnn": "string",
  "snssai": {
    "sst": "integer",
    "sd": "string",
  }
  "routeSlice": [
     {
       "relatPrecedence": int,
       "dnn": "string",
       "snssai": {
         "sst": int,
         "sd": "string"
       }
     }
  ]
}
```

In particular:

- `gpsi` and `supi` are unique identifiers for a single UE. These two fields are mutually exclusive and cannot be included together in the same request.
- `externalGroupId` identifies a group of UEs. This field can be used, for example, to filter UEs based on MCC and MNC values.
- `dnn` and `snssai` represent the originating slice and DNN. This implies that the traffic of all UEs associated with that slice/DNN must be redirected to a new slice.
- `routeSlice` defines a prioritized list of targets slice/DNN pairs. Multiple targets can be specified, allowing the system to attempt the first option and fall back to the next ones if the preferred slice is unavailable.

To enable ENIF to retrieve the available slices from the 5GC, the *Slice Profile* service has been specifically designed for this purpose.

Event Exposure

The Event Exposure functionality is an additional feature able to notify UE events in real time and is based on the AMF EventExposure service introduced in [1]-[3]. This functionality has been developed by implementing a partial prototype of AMF that is linked with the NEF (AMF is the event producer, NEF is the service offering this event to external parties). The available event types are:

- UE Registration State (REGISTERED / DEREGISTERED)
- UE Connectivity State (CONNECTED / DISCONNECTED)
- Timezone
- Access Type (3GPP / N3GPP)
- Rechability (REACHABLE / UNREACHABLE)
- Location (in terms of Cell ID, gNodeB ID, TAI)

Generally, an event notification is triggered when the AMF becomes aware of an information event change (e.g., an event notification is sent when a UE changes from deregistered status to registered status). Sometimes, the NF consumer can ask for the current status of an event. When making a subscription request for an event or a set of events, the NF consumer can ask for event reporting for One UE, a Group of UEs or Any UE. In the case of 'Any UE' type, typically a location area filter or a random selection is applied.

NFs within the 6Green SBA can contact this service to know specific events, subscribing to the service by providing this body structure:

```
{
  "nfId": "string",
  "supi": "string",
  "gpsi": "string",
  "groupId": "string",
  "anyUe": boolean
  "eventList": [
     {
       "type": "Enumerator",
       "immediateFlag": boolean
       "areaList": EventArea
     }
  ],
  "eventNotifyUri": "string",
  "options": {
    "trigger: "Enumerator",
    "expiry: "DateTime",
    "maxReports": integer,
    "repPeriod": "DurationSec"
  }
}
```

In particular:

- `supi`, `gpsi`, `groupId` indicate the UEs, similar as defined above.
- anyUe is a Boolean variable indicating if the event has to be notified for any UE in the core system.
- `eventList` is the list of events to be checked for notification.
- `options` is a structure including some additional parameters.

3.3 Network Slice Intent Creation Process (NF Set 3)

This section explains the slice creation process focusing on the view from the components in NF Set 3. As described in previous deliverables, components involved are mainly BSSF and ENIF.

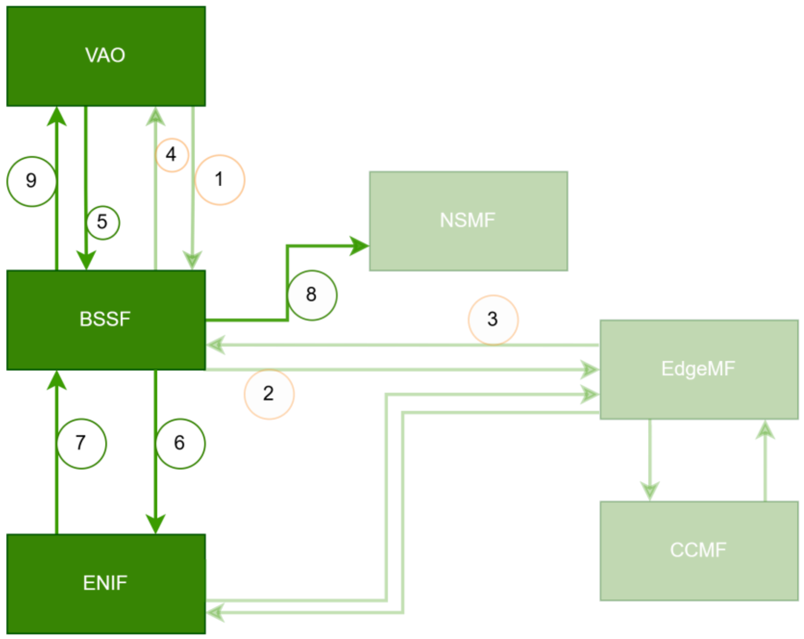


Figure 8: NF Set 3 components and steps for the slice intent creation

In the Figure 8 are represented the steps that traverse the NF Set 3 when a new slice is created. The highlighted ones are the steps that this section will focus on.

3.3.1 BSSF

As previously noted, the BSSF represents the component within the 6Green SBA architecture responsible for interfacing with the customer to receive network slice service requests. The customer's interaction with the 6Green system is facilitated through declarative intent-based interfaces, with syntax aligned to the 3GPP TS 28.312 specification.

From a functional behaviour perspective, the BSSF processes the network slice intent aggregating both SLA and DLA aspects and subsequently translates them into an enriched network slice request for the component in charge of optimizing the deployment slice (ENIF). This enriched request incorporates constraints relevant to the intended slice in accordance with provider-defined policies, which may vary depending on the customer type. In addition, the BSSF is responsible for reporting back to the customer on metrics related to the requested network slice, with respect to both SLA and DLA compliance.

Once the mapping from the requested sites (step 1-4 in Figure 8) is completed (see the Network Slice Intent Infrastructure Offering prototype in section 3.5), the AF/VAO request that slice following an intent format defined by the 3GPP specification [4]. In order to explain the process, the intent will be described by its subdivided parts (expectations). The next interface (Table 2) represents step number 5 in the diagram (Figure 8).

Table 2: Intent expectations

	objectType	contextAttribute	contextValueRange	contextValueRange
objectContexts	Application Component	componentNode InstanceName	IS_EQUAL_TO	name [string]

	contextAttribute	contextCondition	contextValueRange
expectationTargets	ram	IS_EQUAL_TO_OR_GREATER_THAN	size [string]
	cpu	IS_EQUAL_TO_OR_GREATER_THAN	size [string]
	storage	IS_EQUAL_TO_OR_GREATER_THAN	size [string]

	contextAttribute	contextCondition	contextValueRange
expectatonContexts	locationConstraint	IS_EQUAL_TO	id of location [string]

	contextAttribute	contextCondition	contextValueRange
expectationContexts	sliceType	IS_EQUAL_TO	String (name of network slice template)

Each target represents a value that then the optimization component, ENIF, will optimize when deploying the applications. A detailed example of the slice intent is available in the Annex A.

Table 3: Application graph request

	Attribute		Description
enifGraphApp	applicationComponents	computingConstrains	Cpu, memory and storage
		locationConstrains	
	collocationConstraints		Array of constrains
	resourceUnits		Units for each computing constrain

Then, the intent is divided into the network slice (step 8 in Figure 8) and the application slice (step 6 in Figure 8). In the previous Table 3, an overview of the application graph requested to the ENIF component is represented.

Also, from the same intent, network information can be extracted. In this case, the BSSF transforms the high-level requirement, based on the slice type (eMBB, uRLLC, etc.) and its GST attribute value, into network slice profile parameters. Each slice type in the intent maps to a payload type in the slice profile, associated with a specific use case. Furthermore, depending on the slice type, parameters such as the maximum aggregated bit rate and the guaranteed bandwidth determine different uplink and downlink bit-rate configurations in the network slice profile. In this way, the user is abstracted from the technical implementation details and only needs to provide an SLA for the slice.

3.3.2 ENIF

The ENIF receives an input descriptor, the enifGraphApp, from the BSSF, which is in the form of an application graph comprising various components. Each component has associated SLAs specifying CPU, RAM, and storage requirements, as well as colocation and geographical constraints regarding where the component should be deployed. ENIF features an API that listens for placement requests via POST requests at a designated endpoint. This triggers the CCMF/EdgeMF to retrieve the available resources of computing nodes and the geographical areas where those nodes are located. Based on this information, ENIF formulates a Virtual Network Embedding (VNE) problem and invokes its optimization engine to generate an optimal deployment plan. The problem formulation is outlined below (Figure 9).

$$\min \sum_{u \in V_a} \sum_{i \in V_s} \left(c_u^i + r_u^i + s_u^i \right) + QoS(V_a)$$

subject to

$$\sum_{i \in V_s(u)} x_u^i = 1, \quad \forall u \in V_a \tag{1}$$

$$\sum_{i \in V_s \setminus V_s(u)} x_u^i = 0, \quad \forall u \in V_a \tag{2}$$

$$\sum_{u \in V_a} c_u^i \leq C^i, \quad \forall i \in V_s \tag{3}$$

$$\sum_{u \in V_a} r_u^i \leq R^i, \quad \forall i \in V_s \tag{4}$$

$$\sum_{u \in V_a} s_u^i \leq S^i, \quad \forall i \in V_s \tag{5}$$

$$x_u^i \cdot C_u^{min} \leq c_u^i, \quad \forall u \in V_a, \forall i \in V_s \tag{6}$$

$$x_u^i \cdot R_u^{min} \leq r_u^i, \quad \forall u \in V_a, \forall i \in V_s \tag{7}$$

$$x_u^i \cdot S_u^{min} \leq s_u^i, \quad \forall u \in V_a, \forall i \in V_s \tag{8}$$

$$x_u^i \in \{0, 1\}, \tag{9}$$

$$c_u^i, r_u^i, s_u^i \in \mathbb{N}_0, \tag{10}$$

Figure 9: Virtual Network Embedding problem formulation

The placement variables x_u^iindicate whether application component u is placed on computing node i represented in binary form. Similarly, integer variables c_u^i, r_u^i, and s_u^i represent the CPU, RAM, and storage allocated to component on node i , respectively. Constraints (1)-(2) ensure that each application component is placed on appropriate nodes within the desired geographical areas. Constraints (3)-(5) enforce that the resources (CPU, RAM, and storage) of each node are not exceeded. Constraints (6)-(8) ensure that each component is allocated at least the minimum required amount of CPU, RAM, and storage on the selected node. The objective function aims to minimize the total resource allocation across all application components from the Infrastructure Provider's perspective, while from the user's side, it seeks to minimize a generic QoS objective, such as end-to-end delay.

3.4 Network Slice life cycle (Create/Update/Upgrade/Delete) (NF Set 4)

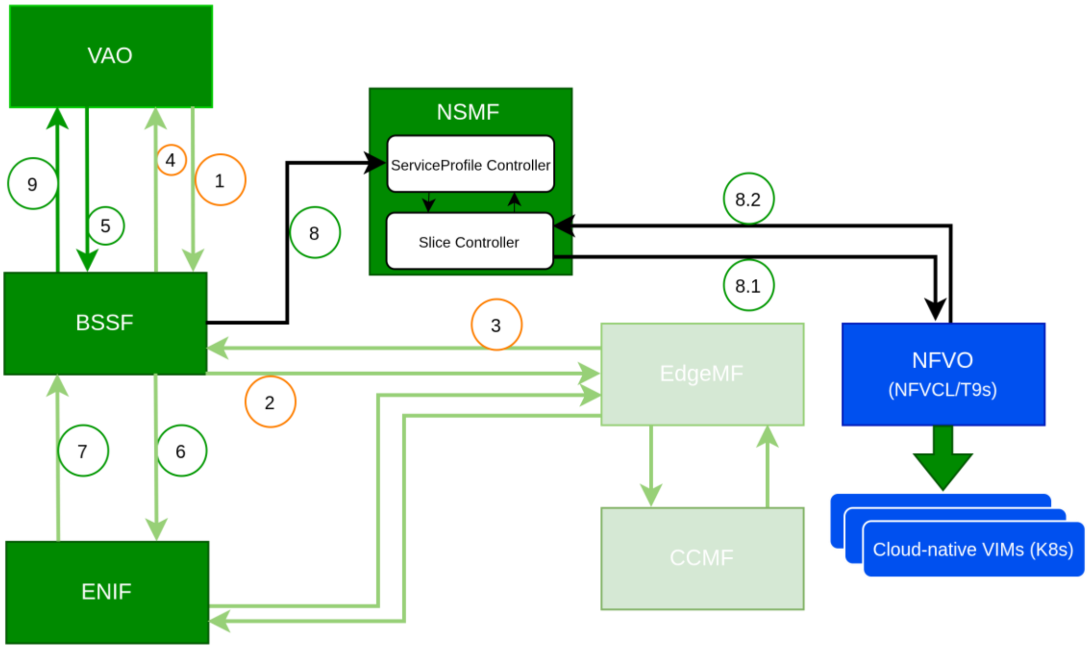


Figure 10: NF Set 4 components for the lifecycle management of network slices

Within the NF set 4, the NSMF and NFMF are responsible for providing the basic functionalities required for the lifecycle management of network slices, including creating, updating, upgrading, and deleting them.

As expressed in D3.2, the cloud-native NSMF has been developed based on the Operator Pattern [5], called the **Slice Operator**, in compliance with TS 28.531 [6] and TS 28.541 [7] specifications, and the cloud-native NFMF, called Athena Manager, has been developed in alignment with TS 28.533 [7].

The Slice Operator incorporates 2 controllers (ServiceProfile controller and Slice controller) for realization of the network slices.

- **ServiceProfile controller:** It is the first controller of NSMF receiving ServiceProfiles from the BSSF as shown in Figure 10. Based on the requested operations, it creates, updates, deletes, or upgrades the relevant Network Slice Instances (NSIs), and then passes the updated list of NSIs to the Slice controller.
- **Slice controller:** It is responsible for creating or updating the Network Service Description (NSD) to realize the requested operations for the NSIs by scaling or slicing the 5G NFs.

NFMF - implemented as a sidecar manager coexisting with 5G NFs—is responsible for configuring and performing the lifecycle management of 5G NFs, ensuring that the components of NSIs are properly deployed in practice.

Below, four main lifecycle management operations for an NSI performed by the NSMF based on requests received from the BSSF are explained:

- **Creating an NSI:** In a ServiceProfile, BSSF specifies the region and zone in which an NSI should be established. The ServiceProfile controller of NSMF maps the combination of the requested region and its zone to the Tracking Area (TA), and based on this mapping, it produces the corresponding SliceProfile and passes it to the Slice controller of NSMF. In the next step, the Slice controller checks whether a 5G Radio Access Network (RAN) NF with the requested TA has already been deployed. If not, it first scales a 5G RAN NF to be deployed in that TA and updates NSD taking into account that radio and computation resources are available in the desired region and zone; otherwise, the NSI is rejected.
- **Upgrading an NSI:** If certain parameters of a ServiceProfile are changed—such as region, zone, or use-case type—NSMF redeploys the relevant NSI. To this end, it first offboards the relevant NSI and then onboards it with the new configuration.
- **Updating an NSI:** When the content-rate parameters of a ServiceProfile change, NSMF updates the QoS policies for the relevant NSI and enforces the new ones. For this reason, it adjusts the radio resources at 5G RAN NFs, such as Physical Resource Blocks (PRBs), for the relevant NSI. Additionally, it updates the downlink and uplink Slice AMBR (Aggregated Maximum Bit Rate) for the NSI and stores the new values in the 5G Core Network.
- **Deleting an NSI:** Upon receiving a request to delete an NSI, NSMF takes care of the isolated or shared 5G NFs that the NSI is using. If one 5G NF is used only by that NSI, NSMF safely deletes it and updates the NSD. On the other hand, if a 5G NF is used by other NSIs, NSMF first tags the NSI as an orphan slice because there is no relevant ServiceProfile for that NSI. Then, while maintaining the idempotency for other NSIs, it converts the S-NSSAI of the NSI to an invalid S-NSSAI, and updates the NSD. Later, if a shared NF does not have a valid S-NSSAI, it will be safely deleted.

Below, the body structure of ServiceProfiles received in the context of the 6Green SBA from BSSF is shown:

```
{
  "apiVersion": "athena.trirematics.io/v1",
  "kind": "ServiceProfile",
  "metadata": {
    "name": "string",
    "namespace": "6green"
  },
  "spec": {
    "coverage": [
      {
        "region": "string",
        "zones": [
          {
            "name": "string",
            "user-density": "Integer"
          }
        ]
      }
    ],
    "user-equipment-type": "string",
    "reporting-period": "string",
  "data-flows": [
   {
    "name": "string",
    "description": "string",
    "data-network": "string",
    "traffic-class": "string",
    "content-rate": {
     "downlink": {
      "maximum": "string",
      "average": "string"
     },
     "uplink": {
      "maximum": "string",
      "average": "string"
     }
    },
      }
  ]
 }
}
```

As discussed in detail in D3.2, the NFMF is responsible for the lifecycle management of workloads, including generating their configuration files, starting them, and stopping them based on the lifecycle operations triggered by the Slice Operator (NSMF). For this reason, the Slice Operator must determine not only the slice information to be provided but also which workloads need to communicate with one another.

3.5 Network Slice Intent Infrastructure Offering (NF Set 5)

This prototype incorporates a functional workflow in which the components that make up the NF Set 5 collaborate with other NFs, such as BSSF and ENIF, to identify the optimal infrastructure for implementing a vApp in a specific geographic area within a 5/6G context (Figure 11).

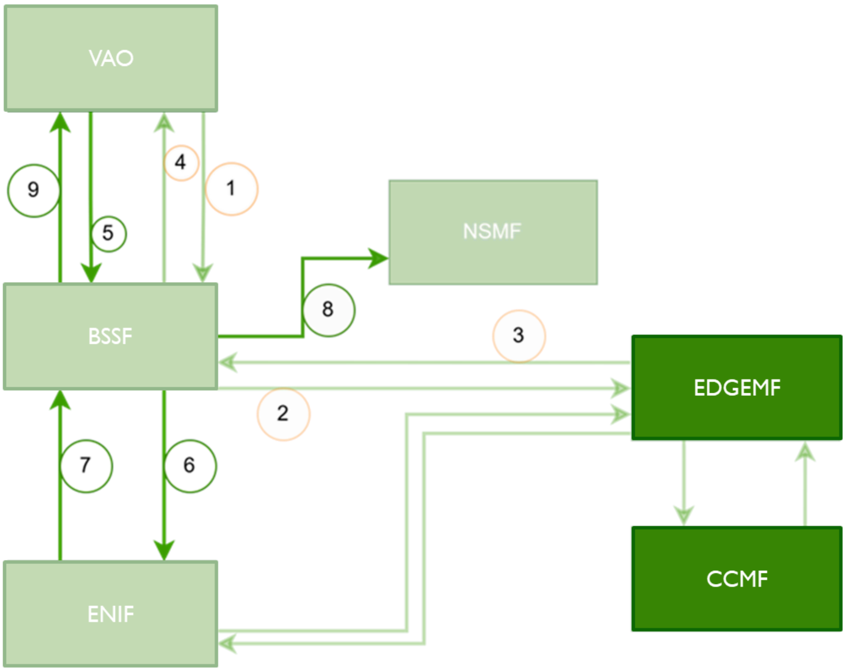


Figure 11: NF Set 5 components offering network edge resources for the slice intent creation

The VAO (Vertical Application Orchestrator) initiates the request for resources to deploy the vertical application in the edge close to a specific area, because it manages the deployment and real-time operations of vertical applications, interacting seamlessly with both end users and network orchestration systems.

Intent translation and discovery of resources

The VAO asks the BSSF (Proxy) for a slice in a concrete geographical area for having resources at the edge to deploy a vertical app (step 1 in Figure 11).

As described in D3.2, and in the prototype Network Slice Intent Creation Process in section 3.3, to find out the areas covered in the existing deployed infrastructure, the BSSF asks the EdgeMF, which is responsible for natively providing computing resources at the network edge for industry-specific applications (vertical applications), which in turn interfaces with the CCMF which acts as a central repository responsible for maintaining an up-to-date registry of the volatile cloud-continuum computing resources and their location and topology, especially at the edge, of the different geographical areas covered by the network. This provides an up-to-date view of the underlying network topology and interconnects these computing resources intra- and inter-site (step 2-3 in Figure 11).

Once the BSSF has the information about the geographic areas, it transforms the intent into a formal segment request, as detailed in the prototype for creating network segment intents in section 3.3.

The intent request contains information about the type of slice required, the computational and network resources needed, and the geographic area in which the slice will be deployed, e.g.:

- Slice type: i.e. URLLC
- Compute requirements: 4 vCPU, 16 GB RAM, optional GPU
- Network requirements: latency <5 ms, jitter <1 ms
- Geographical area: Genoa *company* logistics warehouse

This intent is sent to the ENIF (step 4 in Figure 11), which requests the EdgeMF to retrieve the list of resources in the specific geographical area to initiate the process based on intent-based mechanisms using AI and optimisation methods, to achieve the optimal network slice for the vertical application. When the available resources meet the intent and the most optimal slice has been achieved (if possible), the ENIF communicates with the BSSF returning the slice (cluster) where the application can be deployed according to the application requirements. In parallel, it communicates with the EdgeMF to communicate the resources allocated to the slice, in order to keep the allocated resource information up to date.

Slice composition

The NSMF - Network Slice Management Function starts the assembly:

- Resource reservation on the Genoa area (edge) via the NFV Orchestrator.
- 5G network slicing configuration to enable a slice_id URLLC with QoS.
- Deployment of vertical functionality in containers via Kubernetes or K8s Edge Runtime.
- Configuring access control and security policies.

An instantiated slice event is issued, with ID, initial metrics and endpoint (Cluster) by the ENIF for the BSSF, and this sends it to the VAO.

Feedback and continuous operation

The VAO receives the ACK with endpoints and estimated latency and the vertical application is deployed, starting the traffic of the logistics application starts.

The observability system (NF Set 1) (MDAF/NSDAF - Network Slice Data Analytics Functions) monitors continuously:

- Actual vs. target latency
- CPU/memory usage of the slice
- Degradation or congestion

If any metric falls out of SLA, a remediation intent can be triggered, such as migration of the slice to another node.

At the same time, the CCMF constantly knows the status of the resources available in the infrastructure, by means of calls to the monitoring system (NF Set 1), which evaluates the current load on edge nodes (latency, CPU/GPU usage).

4 Validation and Testing

4.1 Integration Test Strategy

The integration testing strategy focuses on how the different prototypes aligned with the sets of NFs composed of several NFs that are part of 6Green are validated to ensure that they function as a cohesive system. Given that each prototype integrates a different set of NFs focused on specific capabilities, the objective is to define a strategy that allows for consistent testing of their interoperability, coordination, and overall performance.

The prototypes were developed using an incremental integration approach, as defined in WP3. This method gradually combines the network functions to ensure that interfaces and data exchanges operate as expected. A shared testing environment supports this process, allowing automation and continuous monitoring across all prototypes. This approach makes it possible to recreate real operating conditions and detect communication failures between components at an early stage. After each cycle, the results are reviewed so both the prototypes and the shared testing setup can be refined, ensuring that the complete system performs reliably when finalized.

4.2 SBA Prototypes Environment Setup and Configuration

4.2.1 Network Slice Energy Consumption Monitoring (NF Set 1)

The network slice energy consumption is performed at NSDAF level, which is designed as a Flask web service that collects energy consumption data, aggregates it daily, and forecasts future consumption using machine learning models. The system provides REST APIs that allow other components like ENIF (ENI Function) and VAO (Vertical Application Orchestrator) to access both historical and predicted energy data for the labelled slices, as in Figure 12. The functional design enables the API exposure for access to both historical and predicted energy consumption data for network slices.

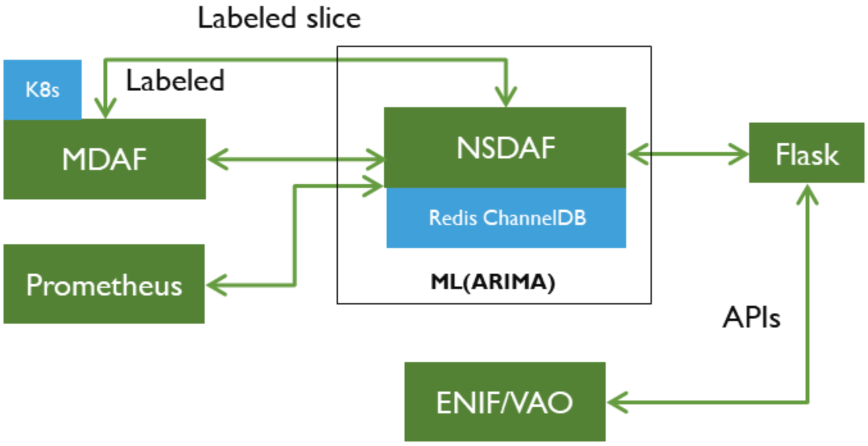


Figure 12: NF SET 1 labelled data slice

This setup allows ENIF, VAO and other systems to easily obtain both historical insights and future predictions through simple REST API calls. NSDAF framework could lead to additional analyses of carbon footprint estimation - by applying conversion factors to the energy data, making the tool highly valuable for energy monitoring, forecasting, and environmental impact assessments.

NSDAF is ingesting data from Redis DB, collectors and exposed through APIs, as designed in Figure 13. Power measurement data for containers and machines is received from a Redis database where it is produced by the MDAF component using data from different agents like Scaphandre[6]. Each container is identified by a unique slice_id, using an internal label configured, described in Chapter 3.1. Labelled with slice_id in order to identify in an easy way each network slice and its own components, are used also for specific metrics to know how to store it and how to process it in NSDAF core function. Main metrics data is related to CPU, memory, storage data usage and network bandwidth. Metrics are correlated with the power consumption of the element.

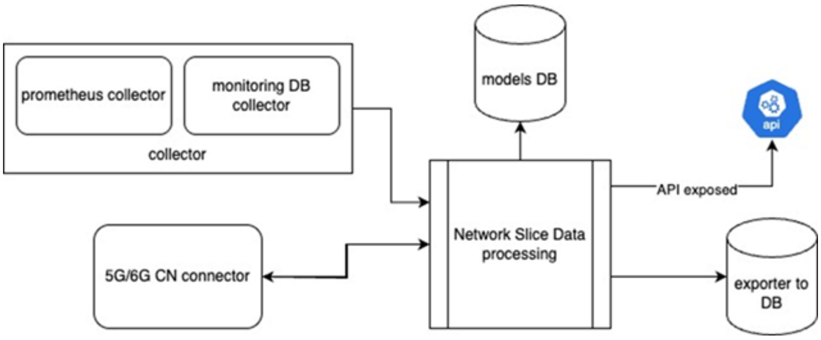


Figure 13: Network slice energy analytics

Based on all data stored, using ML algorithms described below, we can build forecasts for consumption for specific components in specific network slice compared to level of load or resource usage on specific server and power consumption related to it.

Data Preparation:

- The received power measurement data is processed to calculate the daily average energy consumption per slice_id.
- For this phase, we chose a granularity of one day. Finer granularity (e.g., hourly) is possible if needed.

Data Forecasting:

- Implements ARIMA (Autoregressive Integrated Moving Average) model
- Predicts energy consumption 1-28 days into the future
- Model components:
  - AR (p=5): Models relationship with past observations
  - I (d=1): Applies differencing to make data stationary
  - MA (q=1): Models relationship with past forecast errors

Output Delivery via Flask API:

- expose APIs that provide both actual and predicted energy consumption data to systems like ENIF / VAO.
  - Each API call corresponds to a specific slice_id.
  - The data is returned as a list of tuples.
  - List of available slice_ids is retrieved.
  - Metric can have following values: cpu, ram, storage, net.

[6] https://hubblo-org.github.io/scaphandre-documentation/tutorials/getting_started.html

- GET /slice_ids Endpoint, returns a list with all slice ids available in order to return energy metrics per slice ids
    - Output: It returns a JSON response containing the list of slice ids available
- POST /current_power_mircowatts Endpoint, responds to POST requests by returning historical power consumption data.
    - JSON object that includes slice_id and the nr_of_days (an integer specifying how many recent days of data the user wants).
    - The endpoint calls the generate_daily_data() function to get the last 30 days' data, selects the most recent nr_of_days, and then formats this data as a list of tuples.
    - Output returns a JSON response containing the list of date and energy consumption tuples.
- POST /predicted_power_microwatts Endpoint, forecasting future energy consumption.
    - It takes a JSON input with slice_id and nr_of_days (the number of days into the future to forecast).
    - It generates the historical daily data using generate_daily_data() and calls the forecasting function (forecast_energy_consumption()) to predict the energy consumption for the requested number of future days.
    - Output: The endpoint returns a JSON containing a list of tuples, each with a future date (as a string) and the forecasted energy consumption value.

Example of the API structure within the SBA:

```
GET /slice_ids

output:
{
  "slice_ids": [slice_id_1_string, slice_id_2_string, etc],
  "error": Boolean,
  "error_details": None/String
}
```

```
POST /current_power_mircowatts

input:
{
  "slice_id": String,
  "nr_of_days": Integer
}

output:
{
  "data": [(date_string, value_float) ,.....],
  "timestamp": String,
  "error": Boolean,
  "error_details": None/String
}
```

```
POST /predicted_power_microwatts

input:
{
  "slice_id": String,
  "nr_of_days": Integer
}
```

```
output:
{
  "data": [(date_string, value_float), .....],
  "timestamp": String,
  "error": Boolean,
  "error_details": None/String
}
```

Integration with other components, as the API is designed to be consumed by other components in the 6Green ecosystem, particularly:

- ENIF (ENI Function): Uses the API to obtain energy consumption data and predictions for making energy-aware decisions.
- VAO (Vertical Application Orchestrator): Leverages the API to implement energy-efficient orchestration of vertical applications.

Other monitoring and analytics systems: Can use the API to incorporate energy data into broader system analytics.

4.2.2 Network Exposure and Slice Traffic Offloading to Nearby Edge Nodes (NF Set 2)

To validate the functional results of Network Exposure and its integration with the rest of the SBA, a structured interaction has been established among VAO, BSSF, ENIF, and finally NEF, as illustrated in Figure 14.Figure 14: NEF prototype interaction architecture with ENIF, BSSF and VAO

Testing and validation of the interaction between the 5GC and the NEF prototype have already been carried out as part of WP2 activities and are documented in Deliverables D2.3 and D2.4. In this deliverable, the focus will be on the interaction between NEF and the other 6Green SBA components.

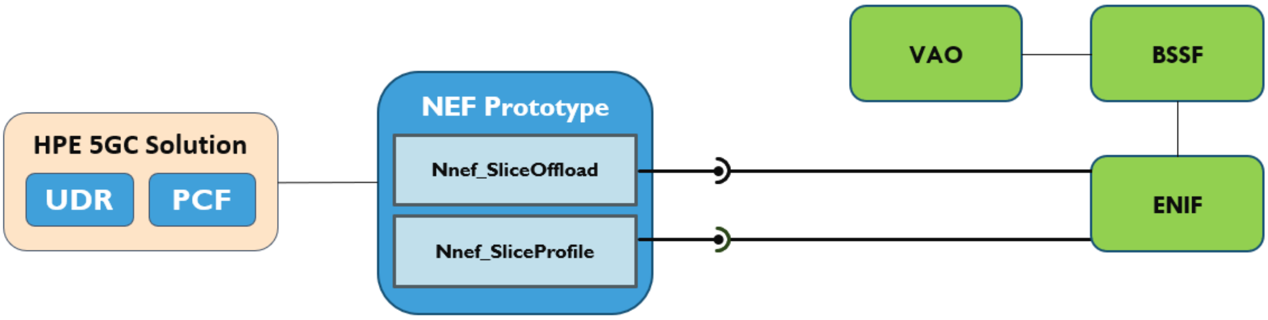


Figure 14: NEF prototype interaction architecture with ENIF, BSSF and VAO

To execute these procedures, the HPE 5GC solution and the NEF prototype application are deployed within the same testbed. The NEF prototype is implemented as a containerized component and configured to connect to the 5GC, exposing its two specified services. The deployment of the other components (VAO, BSSF, and ENIF) has been illustrated and detailed in their respective sections of this deliverable.

Among these, ENIF is the component that directly interacts with NEF through its two services: *Nnef_SliceProfile*, used to request information about the slices currently in use and available, and *Nnef_SliceOffload*, used to request a slice change for a subscriber, a group of subscribers, or all subscribers belonging to a specific initial slice. If ENIF decides that the current active slice is already satisfying the slice modification requirements, it does not anything, if the current slice is not enough to satisfy them, it selects from the available slices the one that satisfy and proceed with the SliceOffload request to the NEF.

In particular, the use cases considered are described in the following sections.

Slice intent modification triggered by VAO

VAO makes a request to modify some of the parameters of the slice intent (e.g., QoS, priority, available bandwidth, different location for UPF). A json body example of slice intent modification is:

```
sliceProfiles": [
    {
      "sliceId": "118",
      "sliceAmbr": "10000.0Mbps",
      "minimumGuaranteedBandwidth": "1231231.0Mbps",
      "enabledUEList": [
        {
          "ICCID": "*"
        }
      ],
      "locationConstraints": [
        {
          "geographicalAreaId": "genoa"
        }
      ],
      "profileParams": {
        "isolationLevel": "NO_ISOLATION",
        "pduSessions": [
          {
            "pduSessionId": "1",
            "flows": [
              {
                "flowId": "f1",
                "gfbr": "20",
                "mfbr": "20",
                "5qi": "6"
              }
            ],
            "pduSessionAmbr": "10Mbps"
          }
        ],
        "ueAmbr": "10000.0Mbps",
        "maximumNumberUE": "0"
      },
      "sliceType": "EMBB"
    }
  ]
```

The slice intent modification passes through the BSSF and then ENIF, which receives the SliceProfile and uses it to compare with the slice(s) currently active in the 5GC.

Slice offload triggered by ENIF

According to specific analytics and decisions, ENIF can decide to make a slice offload request to the NEF. The procedure is similar to the previous section.

4.2.3 Network Slice Intent Creation Process (NF Set 3)

In order to validate the correct functioning of the component, it has been deployed in Kubernetes cluster in the shared test lab. In D3.4, a step-by-step guide is described to deploy the BSSF component in a Kubernetes cluster along with its requirements. The one node cluster for the test is setup on top of a virtual machine with Ubuntu 24.04.2 LTS, 8Gb of RAM, and 30Gb of disk space. The Kubernetes environment is built with MicroK8s v1.32.9 revision 8511 and Docker version 27.5.1.

To test the interface, this intent request can be done to the API endpoint (Figure 15). In case the intent is not following the standardized structure (Figure 16), it returns an error specifying the expectation of such error. This validation step is achieved thanks to tools such as OpenAPI[7] and directly taking the intent definition by 3gpp[8] and extending the schemas[9] to fit the needs of the components in the project.

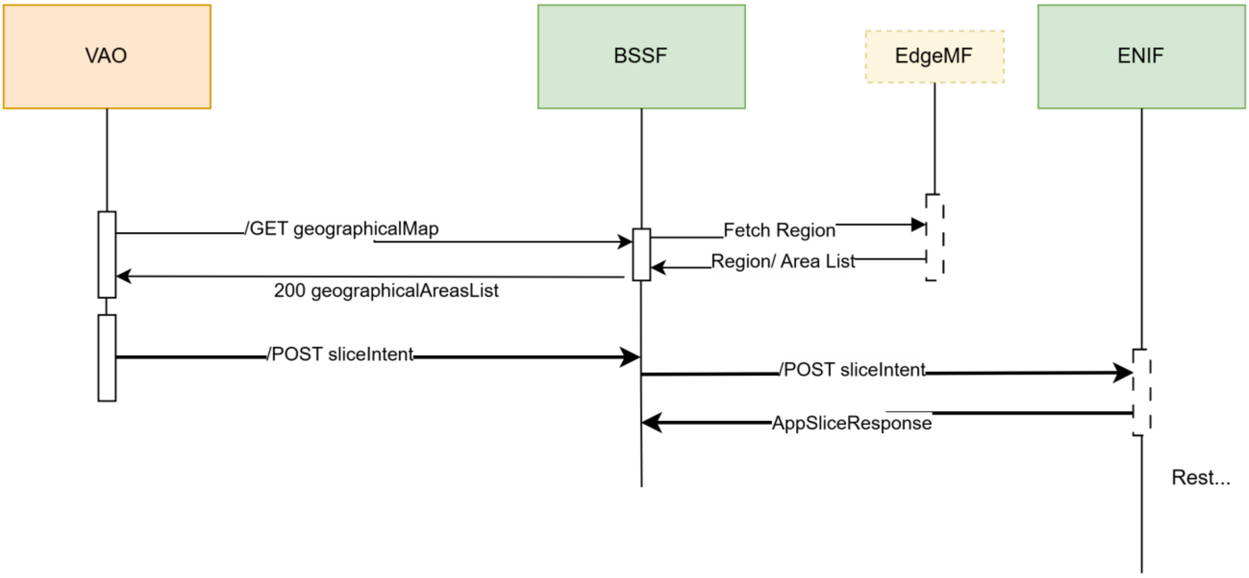
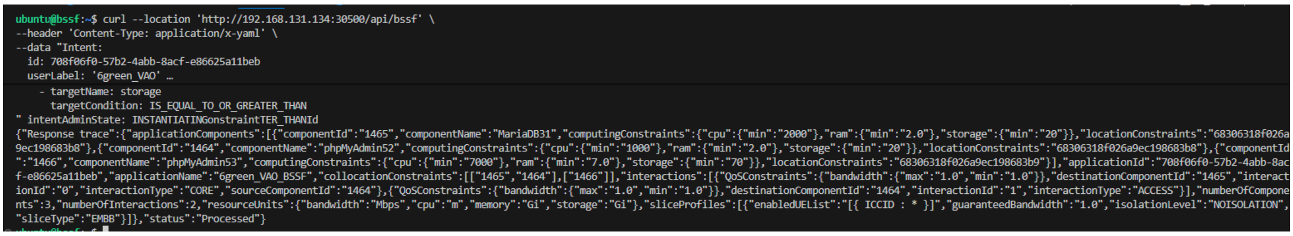


Figure 15: Sequence diagram of tested interfaces



```
ubuntu@bssf:~$ curl --location 'http://192.168.131.134:30500/api/bssf' \
--header 'Content-Type: application/x-yaml' \
--data "Intent:
  id: 708f06f0-57b2-4abb-8acf-e86625a11beb
  userLabel: '6green_VAO' …
    - targetName: storage
      targetCondition: IS_EQUAL_TO_OR_GREATER_THAN
" intentAdminState: INSTANTIATINGonstraintTER_THANId
{"Response trace":{"applicationComponents":[{"componentId":"1465","componentName":"MariaDB31","computingConstraints":{"cpu":{"min":"2000"},"ram":{"min":"2.0"},"storage":{"min":"20"}},"locationConstraints":"68306318f026a
9ec198683b8"},{"componentId":"1464","componentName":"phpMyAdmin52","computingConstraints":{"cpu":{"min":"1000"},"ram":{"min":"2.0"},"storage":{"min":"20"}},"locationConstraints":"68306318f026a9ec198683b8"},{"componentId
":"1466","componentName":"phpMyAdmin53","computingConstraints":{"cpu":{"min":"7000"},"ram":{"min":"7.0"},"storage":{"min":"70"}},"locationConstraints":"68306318f026a9ec198683b9"}],"applicationId":"708f06f0-57b2-4abb-8ac
f-e86625a11beb","applicationName":"6green_VAO_BSSF","collocationConstraints":[["1465","1464"],["1466"]],"interactions":[{"QoSConstraints":{"bandwidth":{"max":"1.0","min":"1.0"}},"destinationComponentId":"1465","interact
ionId":"0","interactionType":"CORE","sourceComponentId":"1464"},{"QoSConstraints":{"bandwidth":{"max":"1.0","min":"1.0"}},"destinationComponentId":"1464","interactionId":"1","interactionType":"ACCESS"}],"numberOfCompone
nts":3,"numberOfInteractions":2,"resourceUnits":{"bandwidth":"Mbps","cpu":"m","memory":"Gi","storage":"Gi"},"sliceProfiles":[{"enabledUEList":"[{ ICCID : * }]","guaranteedBandwidth":"1.0","isolationLevel":"NOISOLATION",
"sliceType":"EMBB"}]},"status":"Processed"}
```

Figure 16: Slice request to the BSSF test

[7] OpenAPI Specification - Version 3.1.0 | Swagger: https://swagger.io/specification/

[8] 5GC_APIs/TS28312_IntentNrm.yaml at Rel-18 jdegre/5GC_APIs: https://github.com/jdegre/5GC_APIs/blob/Rel-18/TS28312_IntentNrm.yaml

[9] BSSF slice request schema: https://github.com/6green/BSSF/blob/main/intent_engine/tools/intent_openapi/slice_request_bssf.yaml

4.2.4 Network Slice life cycle (Create/Update/Upgrade/Delete) (NF Set 4)

To independently test and evaluate the basic functionalities of the NSMF (Slice Operator) and NFMF (Athena Manager) for performing lifecycle management of network slices, a use case named *Campus Event* was defined. In this use case, there are three types of traffic: (1) video streaming traffic sent from security cameras to the event's security room, (2) video chat traffic to provide a dedicated connection among the organizers, and (3) data transfer traffic to provide internet access for the event's participants. Accordingly, three distinct ServiceProfiles were defined, each capturing the network requirements of one traffic type. Figure 17 shows the network connections among the 5G NFs after deploying the ServiceProfiles.

For testing the network connectivity of the created NSIs, a Kubernetes cluster consisting of five machines was used. Additionally, a USRP B210 was employed to provide the over-the-air link, and three Quectel modules were used to connect to the network slices. The RAN workloads ran on OpenAirInterface5g v2.3.0, while the CN workloads ran on Open5GS v2.7.5. Table 4 shows the specifications of the testbed. It should be mentioned that NSMF (Slice Operator) is tested against Trirematics framework.

Table 4: Specifications of the testbed for verifying NSMF and NFMF

Device	OS	CPU	RAM	Role in K8s Cluster	Number
Machine 1	Ubuntu 22.04.4 LTS	Intel Core i9-10920X	64GB	Master Node	1
Machine 2	Ubuntu 22.04.4 LTS	Intel Core i9-10920X	64GB	Worker Node	1
Machine 3	Ubuntu 22.04.4 LTS	AMD Ryzen 9 9950X	64GB	Worker Node	1
Machine 4	Ubuntu 22.04.4 LTS	Intel Core i9-10980XE	32GB	Worker Node	2
Quectel RM520NGL				User Equipment	3
USRP B210				Radio Unit (RU)	1
DGS-2000-10P				Switch	1

4.2.5 Network Slice Intent Infrastructure Offering (NF Set 5)

The Network *Slice Intent Infrastructure Offering* prototype aims to determine and instantiate the optimal infrastructure for deploying a vertical application (vApp) in a specific Slice automatically created for this purpose, based on the infrastructure available in a specific geographic area, and in compliance with the computational requirements of the vApp. The workflow involves several NFs that must interact with each other to deploy the vApp.

To validate the correct functioning of the components covered directly by NF Set 5, both CCMF and EdgeMF have been implemented on separate virtual machines within the shared test lab.

The workflow executed in the prototype covers the following steps: the Vertical component queries the AF (Application Function) for available geographic areas. The AF forwards this request to the BSSF, which retrieves the information by contacting the EdgeMF. The EdgeMF, in turn, queries the CCMF, which maintains a database of available infrastructure resources across Kubernetes clusters.

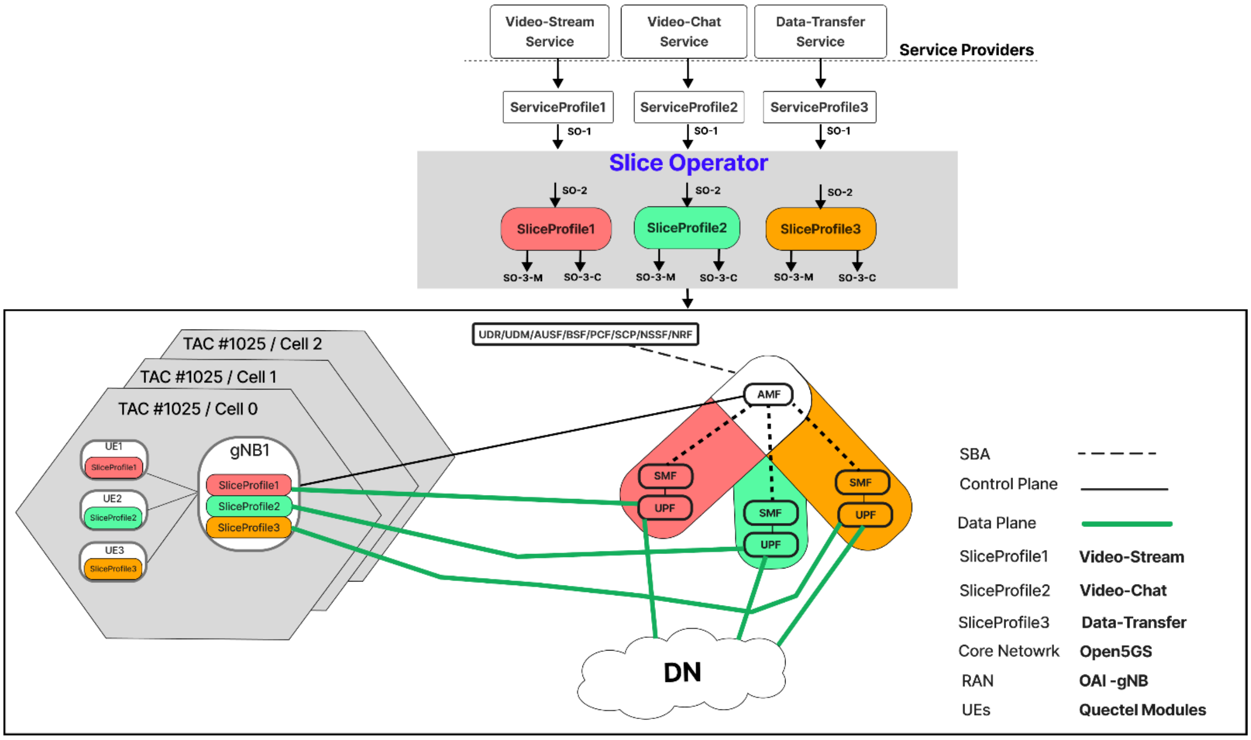


Figure 17: Campus Event use case high level architecture after being deployed by NSMF

Once the geographic areas are returned to the Vertical, the Vertical by the user selects one of them, triggering a new request to identify the infrastructures (K8S clusters) in that area that satisfy the vApp deployment requirements. This request is expressed through slice intents sent from the AF to the BSSF. The BSSF contacts the ENIF, which optimizes the placement of the vApp components across clusters. ENIF returns the placement data to the BSSF, which then instructs the NSMF to instantiate the specific slice. Once instantiated, the NSMF returns the result to the BSSF, which passes it to the AF. The AF deploys the vApp and notifies the Vertical. To perform the test, a request for geographic areas is sent to EdgeMF, and the request is translated to CCMF. Once the response is returned, it is transformed to the appropriate data model expected by BSSF and returned to it.

To test the interface, EdgeMF's Restful Swagger API was used to query geographic areas. An example of the returned json is shown below.

```
[
   {
      "geographicalAreaId":"my-area-id",
      "locationName":"my-location-name",
      "latitude": 44.40478,
      "longitude": 8.94439,
      "coverageRadius":3,
      "segment":null,
      "cluster":{
         "name":"my-cluster",
         "type":"KUBERNETES",
         "nodes":[
            {
               "name":"r1hpgen9-4",
               "labels":{
                  "area":"4",
                  "beta.kubernetes.io/arch":"amd64",
                  "beta.kubernetes.io/os":"linux",
                  "kubernetes.io/arch":"amd64",
                  "kubernetes.io/hostname":"r1hpgen9-4",
                  "kubernetes.io/os":"linux",
                  "node-role.kubernetes.io/control-plane":"",
                  "status/capacity/cpu":"48",
                  "status/capacity/ephemeral-storage":"3074898844Ki",
                  "status/capacity/hugepages-1Gi":"0",
                  "status/capacity/hugepages-2Mi":"0",
                  "status/capacity/memory":"462025428Ki",
                  "status/capacity/pods":"110"
               }
            }
         ]
      }
   }
]
```

D3.4 provides step-by-step details on how to deploy both CCMF and EdgeMF containerized in Docker, as well as how they can be deployed automatically based on helm charts if it is necessary to deploy the components in a Kubernetes cluster.

4.3 KPIs and Performance Metrics

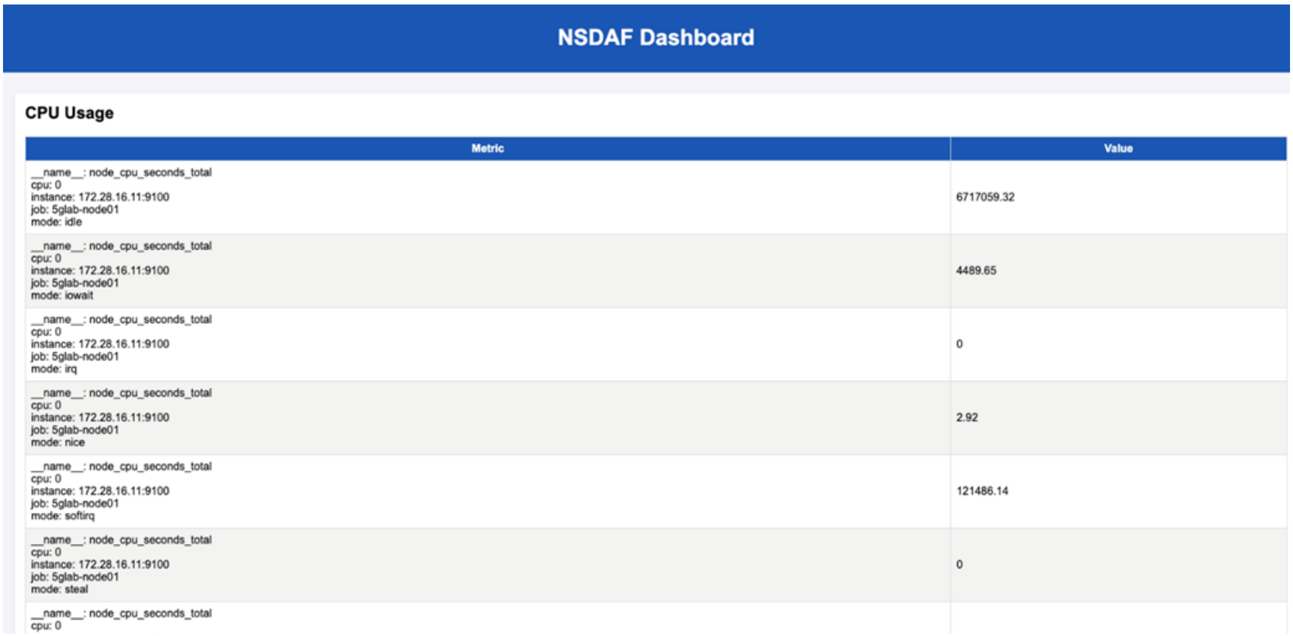
This section describes how the effectiveness of the NF assembly prototypes has been evaluated. Focusing on the technical objectives of each prototype, specific key performance indicators (KPIs) have been defined to achieve these objectives, detailing the measurement points and how they have been monitored, providing graphical evidence such as tables and performance charts. Finally, an analysis of the results is presented.

4.3.1 Network Slice Energy Consumption Monitoring (NF Set 1)

The implementation follows a modular architecture with data ingestion from multiple sources, processing capabilities, machine learning-based forecasting, and a comprehensive RESTful API for data exposure.

NF Set 1 delivers slice energy consumption and forecasting accuracy, as KPI the prediction accuracy of the ARIMA model for energy consumption forecasting. The measurements are statistical metrics including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE), evaluating through comparison between predicted values and actual energy consumption measurements. A time series analysis is used in order to evaluate forecast quality.

The Results of the ARIMA model (with parameters p=5, d=1, q=1) demonstrated effective temporal dependency modelling, proposed implementation allowing for parameter auto-tuning to adapt to different data characteristics. Multiple iterations and training phases are planned to further improve prediction accuracy. The Data Collection and processing efficiency KPI is the ability to collect and process infrastructure metrics and power measurements, using tools as Prometheus time series database for infrastructure metrics collection and Redis database for power measurement data storage and retrieval. The results demonstrate the successful integration with Prometheus for collecting CPU, RAM, disk, and network usage metrics, effective labelling mechanism for identifying containers by unique slice_id and the efficient data processing pipeline for calculating daily average energy consumption per slice, as demonstrated in Figure 18, Figure 19, Figure 20 and Figure 21.

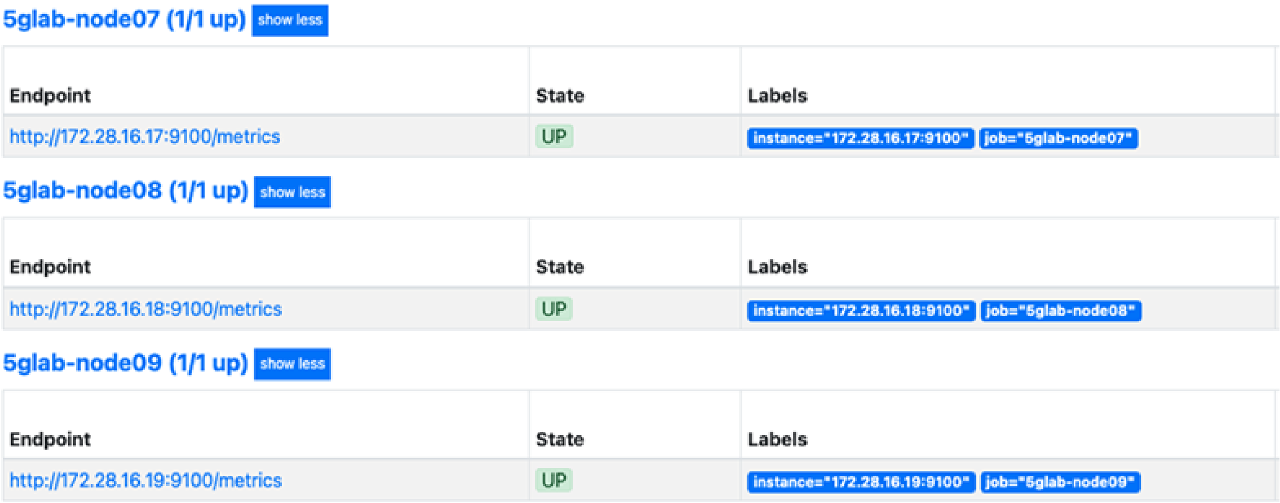


NSDAF Dashboard

CPU Usage

Metric	Value
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: idle	6717059.32
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: iowait	4489.65
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: irq	0
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: nice	2.92
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: softirq	121486.14
__name__: node_cpu_seconds_total cpu: 0 instance: 172.28.16.11:9100 job: 5glab-node01 mode: steal	0
__name__: node_cpu_seconds_total cpu: 0	

Figure 18: NF Set 1 Network Slice analytics dashboard



5glab-node07 (1/1 up) show less

Endpoint	State	Labels
http://172.28.16.17:9100/metrics	UP	instance="172.28.16.17:9100" job="5glab-node07"

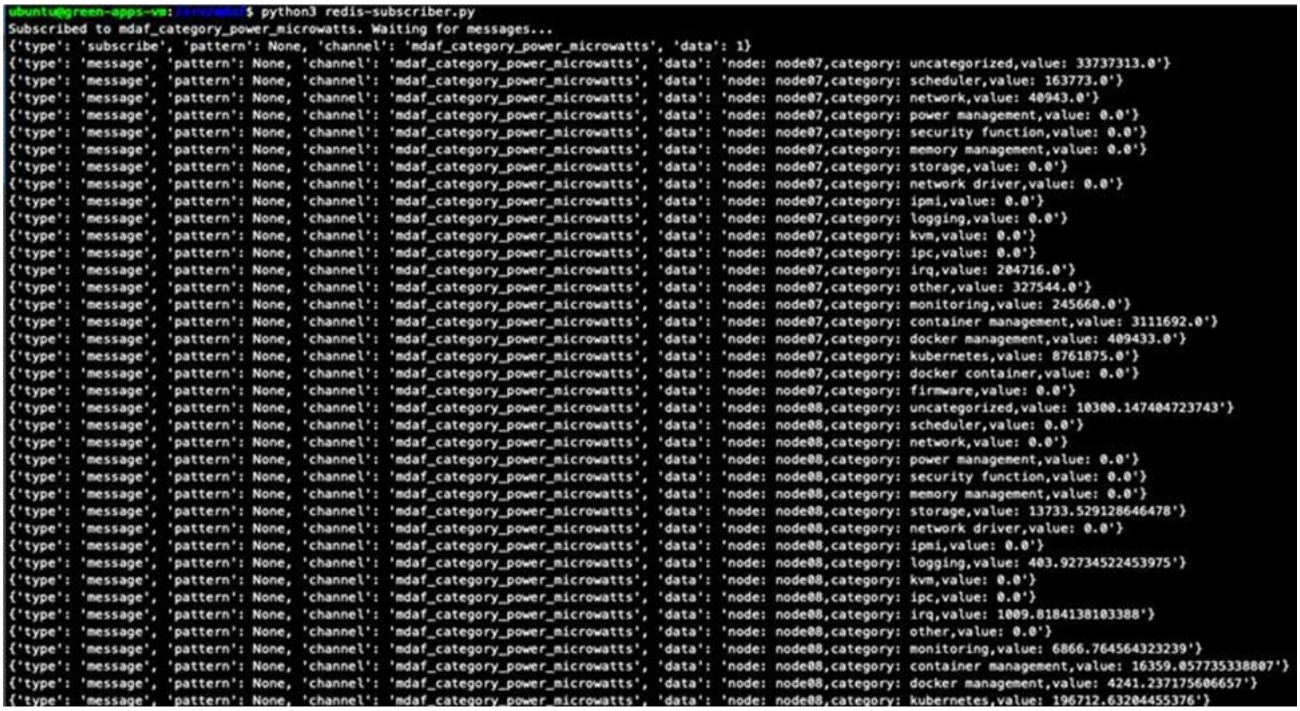
5glab-node08 (1/1 up) show less

Endpoint	State	Labels
http://172.28.16.18:9100/metrics	UP	instance="172.28.16.18:9100" job="5glab-node08"

5glab-node09 (1/1 up) show less

Endpoint	State	Labels
http://172.28.16.19:9100/metrics	UP	instance="172.28.16.19:9100" job="5glab-node09"

Figure 19: NF Set 1 K8s metrics and labelling



```
ubuntu@green-apps-vm:[redacted]$ python3 redis-subscriber.py
Subscribed to mdaf_category_power_microwatts. Waiting for messages...
{'type': 'subscribe', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: uncategorized,value: 33737313.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: scheduler,value: 163773.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: network,value: 40943.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: power management,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: security function,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: memory management,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: storage,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: network driver,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: ipmi,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: logging,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: kvm,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: ipc,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: irq,value: 204716.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: other,value: 327544.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: monitoring,value: 245660.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: container management,value: 3111692.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: docker management,value: 409433.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: kubernetes,value: 8761875.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: docker container,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node07,category: firmware,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: uncategorized,value: 10300.147404723743'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: scheduler,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: network,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: power management,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: security function,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: memory management,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: storage,value: 13733.529128646478'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: network driver,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: ipmi,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: logging,value: 403.92734522453975'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: kvm,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: ipc,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: irq,value: 1009.8184138103388'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: other,value: 0.0'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: monitoring,value: 6866.764564323239'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: container management,value: 16359.057735338807'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: docker management,value: 4241.237175606657'}
{'type': 'message', 'pattern': None, 'channel': 'mdaf_category_power_microwatts', 'data': 'node: node08,category: kubernetes,value: 196712.63204455376'}
```

Figure 20: NF Set 1 REDIS power collection

The system delivers a Redis pub/sub communication channel between the MDAF and the NSDAF. Specifically, it shows the subscription to and messages from the mdaf_aggregated_k8s_container_power_microwatts channel, which transmits power consumption data for Kubernetes containers, as in Table 5. Json example below:

```
json
{
  "type": "subscribe",
  "pattern": null,
  "channel": "mdaf_aggregated_k8s_container_power_microwatts",
  "data": 1
}
```

Table 5: Redis Channel Communication

```
{'type': 'subscribe', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 1}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
1a3953a2b96b5ca32e3288a95540c9de8b40667f0df6a6ea5bec52368d11e63e,node: r1hpgen9-2,namespace:
6green,pod: oai-mysql-77b948697b-jwgqd,cmdline: mysqld,exe: /usr/sbin/mysqld,pid:
63588,label_slice_id: -,value: 17320.37034573427'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
d49368e79f168dd95accea35bdb14b903841d8a8d1f3898b8d35aa213f713e25,node: r1hpgen9-2,namespace:
metallb-system,pod: controller-85499b6dc9-fr8nj,cmdline: /controller--port=7472--log-
level=info--tls-min-version=VersionTLS12,exe: /controller,pid: 402226,label_slice_id: -
,value: 4694.245822600126'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
d6b394993fad39594a3a72f19a2cc29bd35dfdd9eadf91bf71916cfbc1506429,node: r1hpgen9-2,namespace:
openebs,pod: openebs-ndm-operator-6469f6bb4c-pqm8t,cmdline: /usr/local/bin/ndo,exe:
/usr/local/bin/ndo,pid: 4954,label_slice_id: -,value: 0'}
```

```
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
eb6d6da35db7f720fd046d2274ecf5c340cb21e4b0406a08601fa1b1250982de,node: r1hpgen9-2,namespace:
openebs,pod: openebs-ndm-xdkj4,cmdline: /bin/bash/usr/local/bin/entrypoint.sh-v=4--feature-
gates=_GPTBasedUUID_,exe: /usr/bin/bash,pid: 3472,label_slice_id: -,value: 0'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
eb6d6da35db7f720fd046d2274ecf5c340cb21e4b0406a08601fa1b1250982de,node: r1hpgen9-2,namespace:
openebs,pod: openebs-ndm-xdkj4,cmdline: /usr/sbin/ndmstart-v=4--feature-
gates=_GPTBasedUUID_,exe: /usr/sbin/ndm,pid: 3485,label_slice_id: -,value: 0'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
815f8a5244b59a080308086d18fe5ab55e06f80af4d0f12740f1cf245fd45bcd,node: r1hpgen9-2,namespace:
kube-system,pod: kube-proxy-hj8lw,cmdline: /usr/local/bin/kube-proxy--config=/var/lib/kube-
proxy/config.conf--hostname-override=r1hpgen9-2,exe: /usr/local/bin/kube-proxy,pid:
3073,label_slice_id: -,value: 418.489994258728'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
29f3217fbe01cfdd7dc49e43788949492eb3952fadb957ecbc4b27f0910442b4,node: r1hpgen9-2,namespace:
metallb-system,pod: speaker-m2ldf,cmdline: /speaker--port=7472--log-level=info,exe:
/speaker,pid: 402195,label_slice_id: -,value: 4694.245822600126'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
48109aca7ab44631d4a40144bd3b14b6c8c3c1b53e9d103c979be10fcd925901,node: r1hpgen9-2,namespace:
openebs,pod: openebs-ndm-node-exporter-g59ts,cmdline: /usr/local/bin/exporterstart--
mode=node--port=:9101--metrics=/metrics,exe: /usr/local/bin/exporter,pid:
4387,label_slice_id: -,value: 127.94713469575629'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
a479f7a4ce633f258067057ba1039877fa268d735ac3edaf5bd75587855ee2e9,node: r1hpgen9-2,namespace:
openebs,pod: openebs-ndm-cluster-exporter-b49987ffb-lq7hh,cmdline:
/usr/local/bin/exporterstart--mode=cluster--port=:9100--metrics=/metrics,exe:
/usr/local/bin/exporter,pid: 4435,label_slice_id: -,value: 0'}
{'type': 'message', 'pattern': None, 'channel':
'mdaf_aggregated_k8s_container_power_microwatts', 'data': 'container_id:
ba9c2738f64cb50a7b2257f777770d9f93f6d3f65f4e3bdae7e6dbae56237481,node: r1hpgen9-2,namespace:
mdaf-demo,pod: mdaf-k8s-mdaf-sitelevelcarbonemissionscalculator-7c6b8b9dbgxndf,cmdline:
./metric-switch,exe: /go/src/github.gsissc.myatos.net/GLB-BDS-ETSN-SEED/Continuum-Energy-
Aware-Monitoring/metric-switch/metric-switch,pid: 115241,label_slice_id: -,value: 0'}
```

The analysis of container power consumption data is presented in Table 6, hourly power consumption data collected by the MDAF for a specific container, data is published to the Redis channel *mdaf_aggregated_docker_container_power_microwatts* and represents a critical input for the NSDAF.

Table 6: NF Set 1 Container Power Consumption Data

type	channel	container_id	container_scheduler	node	pid	value	date
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	07:46.1
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	08:46.2
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	09:46.2
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	10:46.3
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	11:46.3
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	12:46.4
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	13:46.4
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	14:46.5
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	15:46.5
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	16:46.6
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	17:46.6
message	mdaf_aggregated_docker_container_power_**microwatts**	ID_container	kubernetes	node07	4812	446131.3	18:46.7

Data Structure:

- **Type**, entries are of type "message", data messages in the Redis pub/sub system.
- **Channel**, messages are published to *mdaf_aggregated_docker_container_power_microwatts*.
- **Container_id**, is a unique identifier for the Docker container (afedf584a74b14a3012ca5d2e36f84063cc5484df1e7828f583e0267b04e267b).
- **Value, entries** show a consistent power consumption of "446131.3" microwatts.
- **Date**, the timestamps showing hourly measurements from 07:46.1 to 18:46.7.

In conclusion, this data serves as input to the NSDAF component, which collects this power measurement data from the Redis channel and processes it to calculate daily average energy consumption, as seen in Figure 21. The system uses the processed data to train and feed the ARIMA prediction model and exposes both historical and predicted data through its REST API.

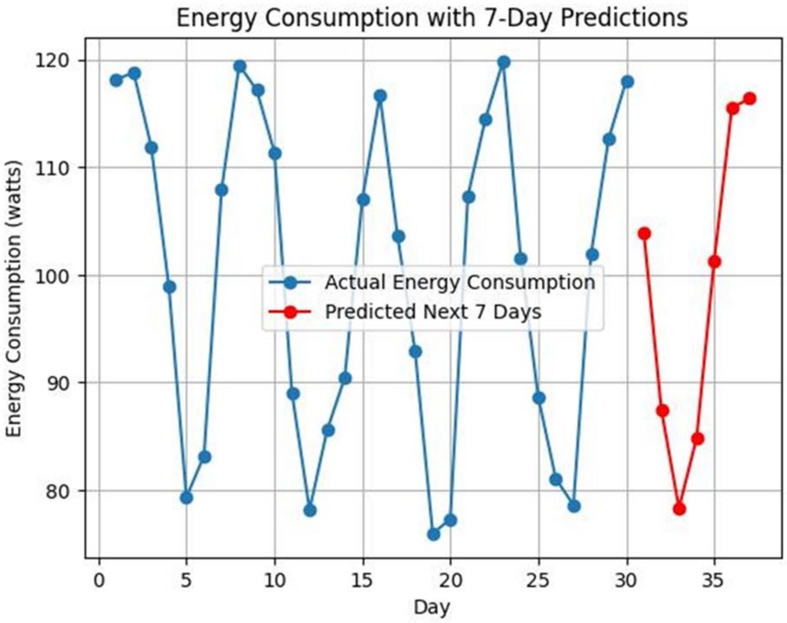


Figure 21: NF Set 1 Energy Consumption prediction

4.3.2 Network Exposure and Slice Traffic Offloading to Nearby Edge Nodes (NF Set 2)

The evaluation of the NEF prototype within this project focuses exclusively on functional validation, rather than performance or KPI-based assessment. The primary objective is to verify that the implemented services (Nnef_SliceProfile and Nnef_SliceOffload) operate correctly and enable interaction with other Network Functions (e.g., ENIF) and the 5GC as intended. Validation of interaction between NEF and 5GC is already done in WP2.

Unlike other components where performance metrics, resource usage, and QoS indicators are measured, the NEF prototype was tested to ensure compliance with 3GPP specifications (TS 29.502 and TS 29.522) and correct execution of the defined procedures. Therefore, no KPIs have been defined for this prototype, as the tests did not include throughput, latency, or resource consumption measurements. Also, validation criteria were based on successful request/response flows and correct handling of slice-related operations.

The functional tests confirmed that:

- NEF correctly retrieves slice profiles from the 5GC and exposes them to ENIF.
- Slice offloading requests are processed and reflected in the 5GC configuration.

4.3.3 Network Slice Intent Creation Process (NF Set 3)

This prototype of Network Slice Intent Creation Process has been evaluated for its effectiveness, considering the placement optimization time (ms), which is the time it takes for the ENIF to calculate the optimal location, based on resource analysis and determining the optimal location for each component of a vApp.

We conducted multiple experiments while varying the number of application components. By aggregating all experimental records and computing their mean, we derived the average execution time of the ENIF optimization solver as a function of the number of application components. The solver addresses a placement optimization problem formulated as a Virtual Network Embedding (VNE) problem. This analysis highlights the scalability behaviour of the solver and provides insight into how its computational complexity evolves as the problem size increases, as shown in the following Figure 22.

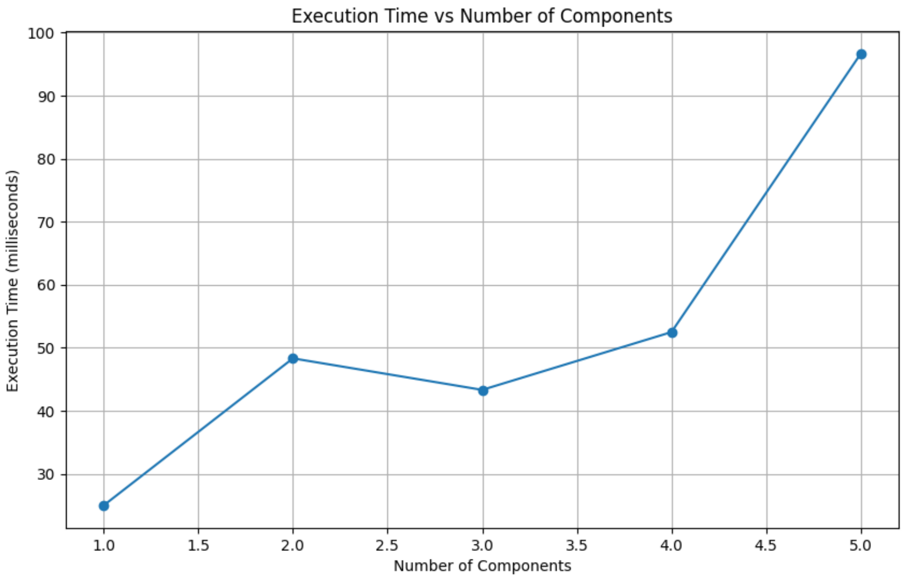


Figure 22: ENIF Optimization Solver results

4.3.4 Network Slice life cycle (Create/Update/Upgrade/Delete) (NF Set 4)

The NSMF (Slice Operator) and NFMF (Athena Manager) prototypes have been evaluated in terms of lifecycle management of NSIs against the T9s framework. The achieved results are discussed in the following subsubsections.

As described in Section 4.2.4, the Campus Event use case has been considered to evaluate NF Set 4. In this use case, there are three different types of traffic: security cameras’ traffic, organizers’ traffic, and participants’ traffic. For each traffic type, one ServiceProfile has been defined, and each ServiceProfile is mapped to one NSI. Table 7 presents the characteristics of these traffic types.

Table 7: Specifications of Traffic Types in the Campus Event Use Case

ServiceProfile	Use Case Type	QoE Requirment	Description
Security Cameras	Video Streaming	Downlink=10Mbit/s, Uplink =10Mbit/s	Continuous HD video streaming for surveillance
Organizers	Video Chat	Downlink=50Mbit/s, Uplink =8Mbit/s	Peer-to-peer communication for event staff
Participants	Data Transfer	Downlink=40Mbit/s, Uplink =3Mbit/s	Typical work like browsing and social media apps

4.3.4.1 Lifecycle Management Evaluation Result

To evaluate the Lifecycle management done by NF Set 4, a storyline has been considered as shown in Figure 23. Based on that, at the interval times T0, T1, and T2 the NSIs of security cameras, organizers, and participants are created, respectively. Then, at the interval times T3 and T4, the NSIs of organizers and participants are updated. Afterwards, at T4, the NSI of organizers is upgraded by changing the use-case type from Video Chat to Push-to-talk. At the end of the event, the created NSIs are deleted reversely when participants and organizers leave the campus and the security cameras' NSI is not needed any more. It should be noted that the entire scenario was repeated 20 times, and the reported values at Figure 24 are the averages over these repetitions.

Table 9 presents the definitions for the terms showed in the Figure 24, and how they are measured based on the time recorded by the Kubernetes' control plane. According to the presented results, not only is the claim of lifecycle management of NSIs validated, but the time required for Day 0, Day 1, and Day 2+ operations of NSIs can also be inferred. To this end, Table 8 represents the target times for the Day 0, Day 1, and Day 2+ operations of NSIs, along with what we were able to achieve in practice by using the T9s framework.

Table 8: Comparison of Target and Actual Times for Day 0, Day 1, and Day 2+ NSI Operations

Operation Phase	Target Time	Actual time
Day 0	< 10s	≤ 5s
Day 1	< 60s	≤ 30s
Day 2+	< 120s	≤ 75s

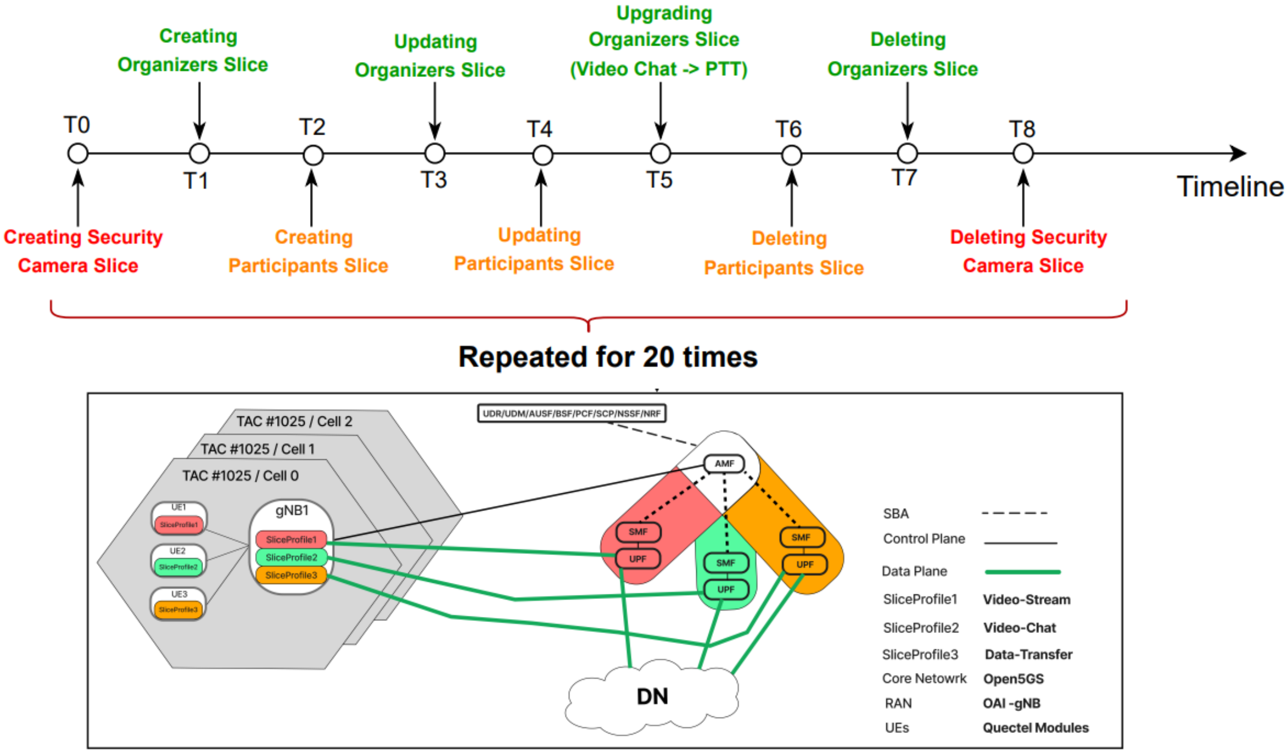


Figure 23: Lifecycle Management Storyline for Campus Event

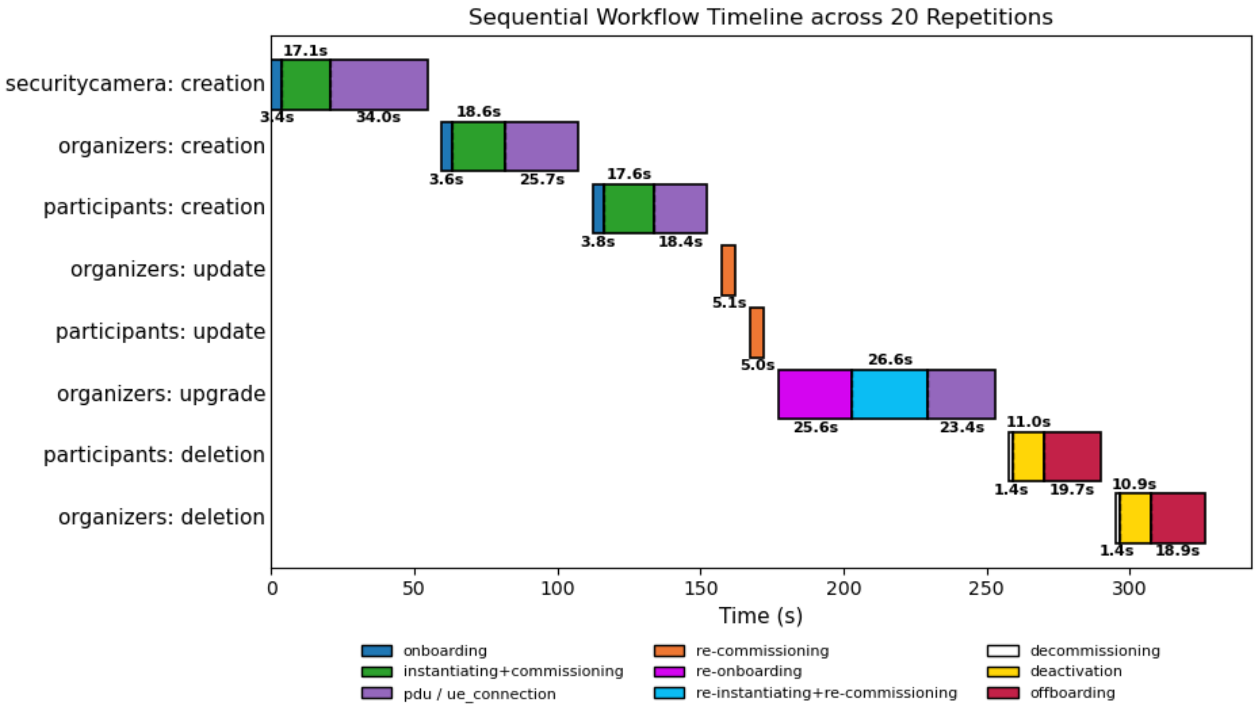


Figure 24: Lifecycle Management Result for Campus Event

Table 9: Explaining the terms in Figure 24

Term	Description	KPI Measurement
Onboarding	Mapping an NSI' components to 5G NFs	Time difference between the CreationTimestamp of a ServiceProfile reported by Kubernetes and the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdatedTime} for the relevant 5G NFs of a ServiceProfile.
Instantiating + Commissioning	Instantiating 5G NFs or reconfiguring them to realize an NSI' components	Time difference between the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdateTime} and the maximum of {Pod.Ready.lastTransitionTime} for the relevant 5G NFs of a ServiceProfile
PDU / UE Connection	Validating the creation of an NSI by connecting a UE and establishing a PDU session for the slice.	Time difference between the maximum of {Pod.Ready.lastTransitionTime} and the time when a UE establishes a PDU session for the slice
Re-commissioning	Reconfiguring 5G NFs	Time difference between the lastModifiedTime of a ServiceProfile and the time of re-enforcing QoS for the relevant ServiceProfile
Re-onboarding	Remapping an NSI' components to 5G NFs	Time difference between the lastModifiedTime of a ServiceProfile and the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdatedTime} for the relevant 5G NFs of a ServiceProfile
Re-instantiating + Recommissioning	Re-instantiating 5G NFs or Re-reconfiguring them	Time difference between the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdateTime} and the maximum of {Pod.Ready.lastTransitionTime} for the relevant 5G NFs of a ServiceProfile after re-onboarding
Decommissioning	Removing an NSI' components from 5G NFs	Time difference between the deletionTimestamp of a ServiceProfile and the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdatedTime} for the relevant 5G NFs of a ServiceProfile
Deactivation	Disconnecting UEs and disable NSIs	Time difference between the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdatedTime} for the relevant 5G NFs of a ServiceProfile and the Disconnecting UEs from the slice
Offboarding	Scaling down 5G NFs or reconfigure them to delete NSI' components	Time difference between the maximum of {Deployment.CreationTimestamp, Deployment.lastUpdatedTime} and the maximum of {Pod.Ready.lastTransitionTime, Pod.deletionTimestamp} for the relevant 5G NFs of a ServiceProfile after deletion

4.3.5 Network Slice Intent Infrastructure Offering (NF Set 5)

This prototype has been evaluated at service level, with the aim of measuring the infrastructure discovery latency (ms), considering it as the time elapsed from selecting a geographic area to obtaining the list of candidates K8S clusters for deployment. The efficiency, accuracy, and reliability of the end-to-end process proposed in the prototype will be provided in WP5, through the use cases defined for the validation of the 6Green SBA architecture, since the final integration of all components involved in the proposed prototype covering different NF Sets has not yet been completed or fully tested at this time.

The infrastructure discovery process identifies the available infrastructure (K8s clusters) for deploying vApp components within a specific geographic area. This is done by requesting the geographic areas and, based on the selection of one of them, requesting the available infrastructure in that area, which is returned in a JSON file, including Kubernetes clusters, nodes, and associated metadata. Information about the available infrastructure is stored in an internal persistence system and changes over time, so discovery latency is not constant and depends on the size and complexity of the underlying infrastructure.

The simulation carried out focuses on the latency of the request to retrieve the infrastructure of a selected area, which represents the most critical part for workflow performance. This approach to simulation allows us to make a realistic approximation of how infrastructure discovery latency evolves as the platform scales, without relying on a specific physical implementation. Therefore, the validation has been performed based on simulated latency, composed of the following elements: *i*) a base latency, representing the network communication, API gateway forwarding, and basic service processing, *ii*) a component proportional to the number of Kubernetes clusters per geographic area, which accounts for increased query complexity, larger response payloads, and higher serialization and transmission costs, *iii*) a component proportional to the total number of nodes returned, as each node contributes additional metadata and tags that must be processed and included in the response, and *iv*) a component proportional to the total number of geographic areas in the system.

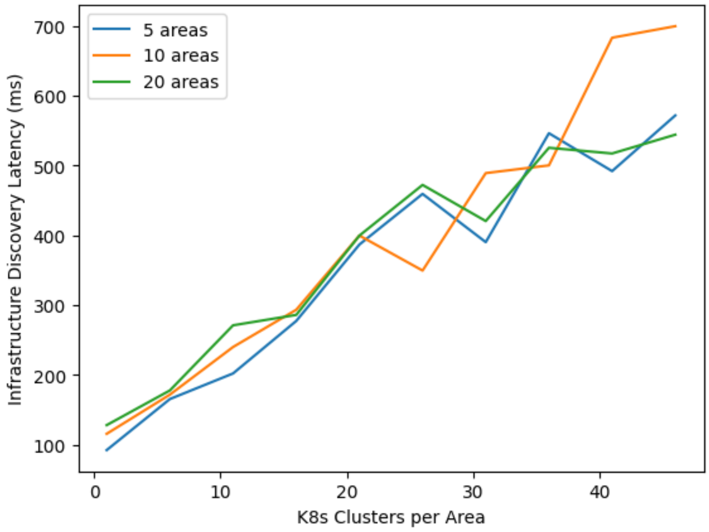


Figure 25: Infrastructure Discovery Latency vs Clusters per Area

The graph shows infrastructure discovery latency (milliseconds) based on the number of Kubernetes clusters per geographic area. The horizontal axis shows infrastructure growth within a single area, and the vertical axis shows end-to-end latency of the discovery request, with each coloured curve corresponding to a different total number of geographic areas in the 6Green ecosystem.

The graph shows that latency increases progressively as the number of clusters per area grows, primarily due to the greater amount of data that must be processed and returned. It also shows that the greater the number of geographic areas, the greater the reference latency, even when querying a single area. This highlights the impact of the global scale of the system on localized discovery operations.

In addition, to complement the KPI evaluated here, other candidate KPIs have been identified to assess the effectiveness of the final SBA 6Green, which can be measured in WP5. These are as follows.

In terms of operational efficiency:

- Infrastructure selection accuracy (%), reflecting the percentage of cases in which the selected infrastructure fully meets the vApp geographical and computational requirements (Memory, CPU, capacity, location).
- Slice instantiation success rate (%), the ratio of successfully instantiated Slices to the total number of intentions issued.

Regarding Reliability:

- Slice instantiation latency (ms), the total time elapsed from sending the “Slice intent” to receiving confirmation of the instantiated Slice from the NSMF.
- Time to deployment / Implementation time (s), which is the total elapsed time from the initial request from Vertical to the operational implementation of the vApp. Covering from the vertical [NFs: AF/VAO – BSSF – EdgeMF – CCMF - ENIF – NSMF].

5 Conclusions

Deliverable D3.3 updates and complements what was reported back in M12 in D3.1 “Work-in-Progress: Enabling Green Interoperability in the 5/6G SBA”. It offers a global view of the innovate 6Green 5/6G Service Based Architecture, providing information about the new and customised network functions grouped into functional groups or frameworks, that enables the new evolved features to support the three 6Green's “Edge Agility,” “Green Elasticity,” and “Energy-aware Backpressure” paradigms.

It also provides the basis, together with the technical requirements identified in WP2, which have been considered to define and implement the 6Green 5/6G service-based architecture. Modularity aspects in the SBA have also been addressed to ensure the separation of different functionalities, interoperability, and security, in order to ensure that it is fully aligned with 3GPP, considering its NFs and service-based interfaces.

The report also provides detailed information on the different functional frameworks and their respective NFs that make up the overall 6Green 5/6G service-based architecture, as well as the technologies proposed to provide connectivity, interoperability, analytical and AI capabilities to the 6Green 5/6G service-based architecture. To this end, five prototypes are presented and evaluated, that represent the different frameworks defined and how their components are integrated, covering the data flow and the nature of the messages exchanged.

Annex A: NF Service APIs

AF/VAO

AF/VAO Services	Operations	Consumers
Naf_Resources	MADE BY GUI	User / human
Naf_ServiceDeploymentRequirement	MADE BY GUI	User / human
Naf_ServiceSliceReply	SliceInstantiationReply	BSSF
Naf_ServiceSlicePolicies	Policies	User / human / NFs

Naf_ServiceSliceReply_SliceInstantiationReply request

Endpoint URI	`<apiRoot>/api/v1/oss/slice`
Method	POST

Request Body (Example)

```
{
  "vasStatus": {
    "vasi": "708f06f0-57b2-4abb-8acf-e86625a11beb",
    "status": "INSTANTIATED",
    "message": null
  },
  "vaQuotaInfo": [
    {
      "clusters": [
        {
          "cluster": {
            "clusterId": "test",
            "certificate-authority-data": "test",
            "server": "test",
            "namespace": "test"
          }
        }
      ],
      "contexts": [
        {
          "namespace": "test",
          "context": {
            "namespace": "test"
          }
        }
      ],
      "users": [
        {
          "token": "test",
          "user": {
            "token": "test"
          }
        }
      ]
    }
  ],
  "componentsMapping": [
    {
      "applicationNodeInstanceId": "1465",
      "clusterId": "test",
      "tag": "test",
      "labels": {}
    },
    {
      "applicationNodeInstanceId": "1464",
      "clusterId": "test",
      "tag": "test",
      "labels": {}
    }
  ],
  "vasConfiguration": {
    "id": "708f06f0-57b2-4abb-8acf-e86625a11beb",
    "name": "a6green1",
    "applicationGraph": {
      "applicationComponents": [
        {
          "applicationNodeInstanceId": "1465",
          "applicationNodeInstanceName": "MariaDB31"
        },
        {
          "applicationNodeInstanceId": "1464",
          "applicationNodeInstanceName": "phpMyAdmin52"
        }
      ],
      "applicationComponentEndpoints": [
        {
          "toApplicationComponentId": "1465",
          "type": "CORE",
          "applicationComponentEndpointId": "1464",
          "fromApplicationComponentId": "1464"
        },
        {
          "toApplicationComponentId": "1464",
          "type": "ACCESS",
          "applicationComponentEndpointId": null,
          "fromApplicationComponentId": null
        }
      ]
    },
    "locationConstraints": [
      {
        "applicationComponentId": "1465",
        "geographicalAreaId": "678780944e5d9af7d4a5c4dc"
      },
      {
        "applicationComponentId": "1464",
        "geographicalAreaId": "678780944e5d9af7d4a5c4dc"
      }
    ],
    "computingConstraints": [
      {
        "applicationComponentId": "1465",
        "ram": "2.0Gi",
        "cpu": "2000m",
        "storage": "20Gi"
      },
      {
        "applicationComponentId": "1464",
        "ram": "2.0Gi",
        "cpu": "1000m",
        "storage": "20Gi"
      }
    ],
    "sliceProfiles": [
      {
        "sliceId": "105901",
        "sliceAmbr": "15000.0Mbps",
        "minimumGuaranteedBandwidth": "1.0Mbps",
        "locationConstraints": [
          {
            "geographicalAreaId": "678780944e5d9af7d4a5c4dc"
          }
        ],
        "profileParams": {
          "isolationLevel": "ISOLATION",
          "pduSessions": [
            {
              "pduSessionId": "1",
              "flows": [
                {
                  "flowId": "f0",
                  "5qi": "6",
                  "gfbr": null,
                  "mfbr": null
                }
              ],
              "pduSessionAmbr": null
            }
          ],
          "ueAmbr": null,
          "maximumNumberUE": null
        },
        "sliceType": "eMBB",
        "enabledUEList": [
          {
            "ICCID": "*"
          }
        ]
      }
    ],
    "callbackUrl": null,
    "networkingConstraints": [
      {
        "applicationComponentId": "2371",
        "applicationComponentEndpointId": "2371",
        "additionalParams": [
          {
            "pduSessionId": "1",
            "flowId": "f0",
            "enableInternetAccess": true
          }
        ],
        "sliceId": "105901"
      }
    ]
  }
}
```

Response Code and Body (Example)	Status Code 200

BSSF

BSSF Services	Operations	Consumers
IntentManagement	GetLocation	AF/VAO
	SliceIntent	AF/VAO

Nbssf_IntentManagement_GetLocation request

Endpoint URI	`<apiRoot>/bssf/v1/location`
Method	`GET`
Request Body (Example)	*Empty*

Response Code and Body (Example)	[
	{
	"geographicalAreaId": "geo_area_123",
	"locationName": "location_A",
	"latitude": 40,
	"longitude": -75,
	"coverageRadius": 50,
	"segment": "segment1"
	}
	]

Nbssf_IntentManagement_SliceIntent request

Endpoint URI	`<apiRoot>/bssf/v1/lcm/instances`
Method	`POST`

Request Body (Example)	{
	"name": "2asdc352scv33434",
	"id": null,
	"callbackUrl": "http://192.168.1.1:9090",
	"applicationGraph": {
	"applicationComponents": [
	{
	"applicationNodeInstanceId": null,
	"applicationNodeInstanceName": null,
	"componentNodeInstanceID": "31",
	"componentNodeInstanceName": "MariaDB"
	},
	{
	"applicationNodeInstanceId": null,
	"applicationNodeInstanceName": null,
	"componentNodeInstanceID": "32",
	"componentNodeInstanceName": "PhpMyAdmin"
	}
	],
	"applicationComponentEndpoints": [
	{
	"applicationComponentEndpointId": "32",
	"fromApplicationComponentId": "32",
	"toApplicationComponentId": "31",
	"type": "CORE"
	},
	{
	"applicationComponentEndpointId": null,
	"fromApplicationComponentId": null,

```
      "toApplicationComponentId": "32",
      "type": "ACCESS"
    }
  ]
},
"locationConstraints": [
  {
    "applicationComponentId": "31",
    "geographicalAreaId": "genoa"
  },
  {
    "applicationComponentId": "32",
    "geographicalAreaId": "genoa"
  }
],
"computingConstraints": [
  {
    "applicationComponentId": "31",
    "ram": "2.0Gi",
    "cpu": "1",
    "storage": "20Gi"
  },
  {
    "applicationComponentId": "32",
    "ram": "2.0Gi",
    "cpu": "1",
    "storage": "20Gi"
  }
],
"networkingConstraints": [
  {
    "applicationComponentId": "32",
    "applicationComponentEndpointId": "32",
    "additionalParams": [
      {
        "pduSessionId": "1",
        "flowId": "f1",
        "enableInternetAccess": true
      }
    ],
    "sliceId": "118"
  }
],
"sliceProfiles": [
  {
    "sliceId": "118",
    "sliceAmbr": "10000.0Mbps",
    "minimumGuaranteedBandwidth": "1231231.0Mbps",
    "enabledUEList": [
      {
        "ICCID": "*"
      }
    ],
    "locationConstraints": [
      {
        "geographicalAreaId": "genoa"
      }
    ],
    "profileParams": {
      "isolationLevel": "NO_ISOLATION",
      "pduSessions": [
        {
          "pduSessionId": "1",
          "flows": [
```

	{
	"flowId": "f1",
	"gfbr": "20",
	"mfbr": "20",
	"5qi": "6"
	}
	],
	"pduSessionAmbr": "10Mbps"
	}
	],
	"ueAmbr": "10000.0Mbps",
	"maximumNumberUE": "0"
	},
	"sliceType": "EMBB"
	}
	]
	}

Response Code and Body (Example)	`Service ID, Status Code 200`

MDAF

MDAF Services	Operations implemented	Consumers in 6Green SBA
Nmdaf_PowerMetricsEnrichment	Notify	NWDAF NSDAF
Nmdaf_Categories	Notify	NWDAF NSDAF
Nmdaf_ConsumptionCategory	Notify	NWDAF NSDAF
Nmdaf_K8sContainerConsumption	Notify	NWDAF NSDAF
Nmdaf_DockerContainerConsumption	Notify	NWDAF NSDAF

Nmdaf_PowerMetricsEnrichment

Endpoint URI	`GET /api/enriched_metrics (a list of enriched produced metrics)`
Method	`GET`
Request Body (Example)	*Empty*

Response Code and Body (Example)	[
	{
	"name": "example",
	"description": "Average consumpion of ...",
	"category": "K_PROCESS"
	"value": 15.7,
	"nfid": "mdaf-001",
	"nfType": "MDAF"
	}
	]

Nmdaf_Categories

Endpoint URI	`GET /api/metrics_categories`
Method	`GET`
Request Body (Example)	*Empty*
Response Code and Body (Example)	`["K_PROCESS", "...", "..."]`

Nmdaf_ConsumptionCategory

Endpoint URI	`GET /api/category_consumption`
Method	`GET`
Request Body (Example)	*Empty*

Response Code and Body (Example)	`[`
	`  {`
	`    "K_PROCESS": 21449.9`
	`  },`
	`  {`
	`    "...": 34.34`
	`  },`
	`  {`
	`    "APP": 12345.67`
	`  }`
	`]`

Nmdaf_K8sContainerConsumption

Endpoint URI	`GET /api/k8s_container_consumption`
Method	`GET`
Request Body (Example)	*Empty*

Response Code and Body (Example)	`[`
	`  {`
	`    "container_id_1": 349.9`
	`  },`
	`  {`
	`    "...": 46.24`
	`  },`
	`  {`
	`    "container_id_n": 1245.65`
	`  }`
	`]`

Nmdaf_DockerContainerConsumption

Endpoint URI	`GET /api/docker_container_consumption`
Method	`GET`
Request Body (Example)	*`Empty`*

Response Code and Body (Example)	`[`
	`  {`
	`    "container_id_1": 349.9`
	`  },`
	`  {`
	`    "...": 46.24`
	`  },`
	`  {`
	`    "container_id_n": 1245.65`
	`  }`
	`]`

NWDAF

NWDAF Services	Operations implemented	Consumers in 6Green SBA
Nnwdaf_AnalyticsSubscription (TS 23.501) **Nwdaf_EventsSubscription (TS 29.520)**	Subscribe Unsubscribe	ENIF NSDAF
Nnwdaf_AnalyticsInfo	Subscribe Unsubscribe	ENIF NSDAF
Nnwdaf_MLModelProvision	Subscribe Unsubscribe	ENIF NSDAF
Nnwdaf_AnalyticsNumberUEsSARIMA	Subscribe Unsubscribe	ENIF NSDAF

Nnwdaf_AnalyticsInfo

Endpoint URI	`GET /api/metrics`
Method	`GET`
Request Body (Example)	*`Empty`*

Response Code and Body (Example)	`[`
	`  {`
	`    "name": "latency_ms",`
	`    "description": "Average network latency in milliseconds",`
	`    "value": 15.7,`
	`    "nfid": "upf-001",`
	`    "nfType": "UPF"`
	`  },`
	`  {`
	`    "name": "authentication_success_rate",`
	`    "description": "Successful authentication rate percentage",`
	`    "value": 98.5,`
	`    "nfid": "ausf-001",`
	`    "nfType": "AUSF"`
	`  }`
	`]`

Nnwdaf_MLModelProvision

Endpoint URI	`GET /api/models`
Method	GET
Request Body (Example)	*Empty*

Response Code and Body (Example)	{
	  "id": "AX2304LS",
	  "model": "SARIMA",
	  "description": "Sarima module for forecasting a specific value",
	  "required_metrics": [
	    {
	      "name": "authentication_success_rate",
	      "description": "Successful authentication rate percentage",
	      "value": null,
	      "nfid": "ausf-001",
	      "nfType": "AUSF"
	    }
	  ],
	  "produced_metrics": [{
	    "name": "authentication_success_rate_prediction",
	    "description": "Forecasting of successful authentication rate
	percentage",
	    "value": null,
	    "nfid": "nwdaf-001",
	    "nfType": "NWDAF"
	  }]
	}

Nnwdaf_AnalyticsNumberUEsSARIMA

Endpoint URI	`GET /api/plugins/sarima_nue/forecast`
Method	GET
Request Body (Example)	*Empty*

Response Code and Body (Example)	{
	  "status": "success",
	  "data": {
	    "resultType": "matrix",
	    "result": [
	      {
	        "metric": {
	          "app": "nwdaf-forecast",
	          "nfid": "nwdaf-001",
	          "nfType": "NWDAF"
	        },
	        "values": [
	          [1738503540, "45"],
	          [1738503555, "46"],
	          [1738503570, "44"],
	          ...
	        ]
	      }
	    ]
	  }
	}

NSDAF

NSDAF Services	Operations implemented	Consumers in 6Green SBA
Nnsdaf_AnalyticsSubscriptionSlice	Subscribe Unsubscribe Notify	ENIF
Nnsdaf_AnalyticsInfoSlice		ENIF
Nnsdaf_DataManagementSlice		ENIF
Nnsdaf_MLModelProvisionSlice		N/A

Nnsdaf_AnalyticsSubscription_Subscribe request

Endpoint URI	`POST /analytics-subscription`
Method	`POST`

Request Body (Example)	`Content-Type: application/json`
	`{`
	` "method": "subscribe",`
	` "nsdaf-analytics": "traffic-monitor-module"`
	`}`
	`{`
	` "ID": 10243,`
	` "timestamp": "2025-12-05T14:22:19Z",`
	` "state": "subscribed",`
	` "error_details": null`
	`}`

Response Code and Body (Example)	`200 OK`

Nnsdaf_AnalyticsSubscription_Unsubscribe request

Endpoint URI	`POST /analytics-subscription`
	`Content-Type: application/json`

Method	`DELETE`

Request Body (Example)	`DELETE /analytics-unbscription`
	`Content-Type: application/json`
	`{`
	` "method": "unsubscribe",`
	` "nsdaf-analytics": "traffic-monitor-module"`
	`}`
	`{`
	` "ID": 10243,`
	` "timestamp": "2025-12-05T14:25:47Z",`
	` "state": "success",`
	` "error_details": null`
	`}`

Response Code and Body (Example)	`200 OK`

Nnsdaf_AnalyticsSubscription_Notify request

Endpoint URI	`POST /analytics-subscription`
	`Content-Type: application/json`

Method	`POST`

Request Bodyv (Example)	`POST /analytics-subscription`
	`Content-Type: application/json`
	`{`
	` "method": "state-of-subscription",`
	` "nsdaf-analytics": "traffic-monitor-module"`
	`}`
	`{`
	` "ID": 10243,`
	` "timestamp": "2025-12-05T14:25:47Z",`
	` "state": "success",`
	` "error_details": null`
	`}`

Response Code and Body (Example)	`200 OK`

PCF

NSDAF Services	Operations implemented	Consumers in 6Green SBA
Npcf_PolicyAuthorization	Subscribe	ENIF

Npcf_PolicyAuthorization_Subscribe request

Endpoint URI	`<apiRoot>/ npcf-policyauthorization/v1/app-sessions`
Method	`POST`

Request Body (Example)	`{`
	`  "ascReqData": {`
	`    "dnn": "internet",`
	`    "medComponents": {`
	`      "1": {`
	`        "fStatus": "ENABLED",`
	`        "marBwDl": "1000000 bps",`
	`        "marBwUl": "1000000 bps",`
	`        "medCompN": 1,`
	`        "medSubComps": {`
	`          "1": {`
	`            "fDescs": [`
	`              "permit out ip from any 1-65535 to any 1-65535",`
	`              "permit in ip from any 1-65535 to any 1-65535"`
	`            ],`
	`            "fNum": 1,`
	`            "flowUsage": "NO_INFO"`
	`          }`
	`        },`
	`        "medType": "DATA",`
	`        "mirBwDl": "1000000 bps",`
	`        "mirBwUl": "1000000 bps",`
	`        "rrBw": "1000000 bps",`
	`        "rsBw": "1000000 bps"`
	`      }`
	`    },`
	`    "notifUri": "http://127.0.0.1:8765",`

	"resPrio": "PRIO_2", "suppFeat": "0", "ueIpv4": "10.20.0.1" } }
Response Code and Body (Example)	Same body as above as confirmation.

NEF

NSDAF Services	Operations implemented	Consumers in 6Green SBA
Nnef_AFSessionWithQoS	Subscribe	ENIF
Nnef_MonitoringEvent (EventExposure)	Subscribe Unsubscribe	ENIF
Nnef_SliceOffload	Subscribe	ENIF

Nnef_AFSessionWithQoS_Subscribe request

Endpoint URI	{apiRoot}/nef/3gpp-af-session-with-qos/v1/{afId}/subscriptions
Method	POST

Request Body (Example)	{
	"dnn": "internet",
	"snssai": {
	"sst": 1,
	"sd": "000001"
	}
	"ueIpv4Addr": "10.20.0.14",
	"ipDomain": "10.20.0.0",
	"qosReference": "string"
	}

Response Code and Body (Example)	Status code: 201, Created
	{
	"dnn": "internet",
	"snssai": {
	"sst": 1,
	"sd": "000001"
	}
	"ueIpv4Addr": "10.20.0.14",
	"ipDomain": "10.20.0.0",
	"qosReference": "string"
	}

Note: qosReference attribute is mapped by NEF to a specific media component (medComponent) to then be used in PolicyAuthorization request to PCF.

Nnef_MonitoringEvent_Subscribe request

Endpoint URI	{apiRoot}/nef/3gpp-monitoring-event/v1/{afId}/subscriptions
Method	POST
Request Body (Example)	{ "gpsi": "0039473646367", "externalGroupId": "01", "ipv4Addr": "10.0.2.30", "eventList": [

	"type": "REACHABILITY_REPORT"
	  ],
	  "nfId": 12345",
	  "eventNotifyUri": http://127.0.0.1:8123/notify
	  "notifyCorrelationId": "001",
	  "options": {
	    "trigger": "CONTINUOUS",
	    "expiry": " 2024-11-05T11:25:00.503155Z"
	  }
	}

Response Code and Body (Example)	Status code: 201, Created
	{
	  "gpsi": "0039473646367",
	  "externalGroupId": "01",
	  "ipv4Addr": "10.0.2.30",
	  "eventList": [
	    "type": "REACHABILITY_REPORT"
	  ],
	  "nfId": 12345",
	  "eventNotifyUri": http://127.0.0.1:8123/notify
	  "notifyCorrelationId": "001",
	  "options": {
	    "trigger": "CONTINUOUS",
	    "expiry": " 2024-11-05T11:25:00.503155Z"
	  }
	}

Nnef_SliceOffload_Subscribe request

Endpoint URI	{apiRoot}/nef/slice-offload/v1/subscriptions
Method	POST

Request Body (Example)	{
	  "afServiceId": "string",
	  "nfId": "string",
	  "gpsi": "string",
	  "supi": "string",
	  "externalGroupId": "string",
	  "dnn": "string",
	  "snssai": {
	    "sst": "integer",
	    "sd": "string",
	  }
	  "routeSlice": [
	     {
	       "relatPrecedence": int,
	       "dnn": "string",
	       "snssai": {
	         "sst": int,
	         "sd": "string"
	       }
	     }
	  ]
	}

Response Code and Body (Example)	Status code: 201, Created *Empty Body*

ENIF

ENIF Services	Operations	Consumers
CreateSliceDeploymentPlan	Create initial slice intent deployment plan	BSSF
UpdateSliceDeploymentPlan	Update slice intent deployment plan	BSSF
OptimizationEngine	Solve VNE problems in the form of MILP	ENIF

ENIF_CreateSliceDeploymentPlan request

<table>
<tr><td><b>Endpoint URI</b></td><td>{apiRoot}/enif/v1/placement/</td></tr>
<tr><td><b>Method</b></td><td>POST</td></tr>
<tr><td><b>Request Body (Example)</b></td><td><pre>
{
  "applicationComponents": [

           {
             "componentId": "1465",
             "componentName": "MariaDB31",
             "computingConstraints": {
               "cpu": {
                 "min": "2"
               },
               "ram": {
                 "min": "2"
               },
               "storage": {
                 "min": "2"
               }
             },
             "locationConstraints": "3"
           },
           {
             "componentId": "1464",
             "componentName": "phpMyAdmin52",
             "computingConstraints": {
               "cpu": {
                 "min": "1"
               },
               "ram": {
                 "min": "2"
               },
               "storage": {
                 "min": "2"
               }
             },
             "locationConstraints": "3"
           },
           {
             "componentId": "1466",
             "componentName": "phpMyAdmin53",
             "computingConstraints": {
               "cpu": {
                 "min": "2"
               },
               "ram": {
                 "min": "2"
</pre></td></tr>
</table>

```
              },
              "storage": {
                "min": "2"
              }
            },
            "locationConstraints": "2"
          }
        ],
        "applicationId": "708f06f0-57b2-4abb-8acf-e86625a11beb",
        "applicationName": "6green_VAO_BSSF",
        "collocationConstraints": [
          [
            "1465",
            "1464"
          ],
          [
            "1466"
          ]
        ],
        "interactions": [
          {
            "QoSConstraints": {
              "bandwidth": {
                "max": "1.0",
                "min": "1.0"
              }
            } ,
            "destinationComponentId": "1465",
            "interactionId": "0",
            "interactionType": "CORE",
            "sourceComponentId": "1464"
          },
          {
            "QoSConstraints": {
              "bandwidth": {
                "max": "1.0",
                "min": "1.0"
              }
            },
            "destinationComponentId": "1464",
            "interactionId": "1",
            "interactionType": "ACCESS"
          }
        ],
        "numberOfComponents": 3,
        "numberOfInteractions": 2,
        "resourceUnits": {
          "bandwidth" : "Mbps",
          "cpu": "m",
          "memory": "Gi",
          "storage": "Gi"
        },
        "sliceProfiles": [
          {
            "enabledUEList": "[{ ICCID : * }]",
            "guaranteedBandwidth": "1.0",
            "isolationLevel": "NOISOLATION",
            "sliceType": "EMBB"
          }
        ]
}
```

Response Code and Body (Example)

```
Status code: 201, Created
{
  "applicationComponents": [
    {
      "componentId": "1465",
      "componentName": "MariaDB31",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1464",
      "componentName": "phpMyAdmin52",
      "computingConstraints": {
        "cpu": {
          "min": "1"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "1",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1466",
      "componentName": "phpMyAdmin53",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "2",
      "placement": {
        "node_name": "0fjval-vm-w-1",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    }
  ],
  "applicationId": "708f06f0-57b2-4abb-8acf-e86625a11beb",
  "applicationName": "6green_VAO_BSSF",
  "collocationConstraints": [
    [
      "1465",
      "1464"
    ],
    [
      "1466"
    ]
  ],
  "interactions": [
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "0",
      "destinationComponentId": "1465",
      "interactionType": "CORE",
      "sourceComponentId": "1464"
    },
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "1",
      "destinationComponentId": "1464",
      "interactionType": "ACCESS",
      "sourceComponentId": null
    }
  ],
  "numberOfComponents": 3,
  "numberOfInteractions": 2,
  "resourceUnits": {
    "bandwidth": "Mbps",
    "cpu": "m",
    "memory": "Gi",
    "storage": "Gi"
  },
  "sliceProfiles": [
    {
      "enabledUEList": "[{ ICCID : * }]",
      "guaranteedBandwidth": "1.0",
      "isolationLevel": "NOISOLATION",
      "sliceType": "EMBB"
    }
  ]
}
```

ENIF_UpdateSliceDeploymentPlan request

Endpoint URI	{apiRoot}/enif/v1/redeployment/
Method	POST

Request Body (Example)

```
{
  "applicationComponents": [
    {
      "componentId": "1465",
      "componentName": "MariaDB31",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1464",
      "componentName": "phpMyAdmin52",
      "computingConstraints": {
        "cpu": {
          "min": "1"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "1",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1466",
      "componentName": "phpMyAdmin53",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "2",
      "placement": {
        "node_name": "0fjval-vm-w-1",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    }
  ],
  "applicationId": "708f06f0-57b2-4abb-8acf-e86625a11beb",
  "applicationName": "6green_VAO_BSSF",
  "collocationConstraints": [
    [
      "1465",
      "1464"
    ],
    [
      "1466"
    ]
  ],
  "interactions": [
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "0",
      "destinationComponentId": "1465",
      "interactionType": "CORE",
      "sourceComponentId": "1464"
    },
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "1",
      "destinationComponentId": "1464",
      "interactionType": "ACCESS",
      "sourceComponentId": null
    }
  ],
  "numberOfComponents": 3,
  "numberOfInteractions": 2,
  "resourceUnits": {
    "bandwidth": "Mbps",
    "cpu": "m",
    "memory": "Gi",
    "storage": "Gi"
  },
  "sliceProfiles": [
    {
      "enabledUEList": "[{ ICCID : * }]",
      "guaranteedBandwidth": "1.0",
      "isolationLevel": "NOISOLATION",
      "sliceType": "EMBB"
    }
  ]
}
```

Response Code and Body (Example)

```
Status code: 201, Created
{
  "applicationComponents": [
    {
      "componentId": "1465",
      "componentName": "MariaDB31",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1464",
      "componentName": "phpMyAdmin52",
      "computingConstraints": {
        "cpu": {
          "min": "1"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "3",
      "placement": {
        "node_name": "0fjval-vm-w-3",
        "cpu": "1",
        "ram": "2",
        "storage": "2"
      }
    },
    {
      "componentId": "1466",
      "componentName": "phpMyAdmin53",
      "computingConstraints": {
        "cpu": {
          "min": "2"
        },
        "ram": {
          "min": "2"
        },
        "storage": {
          "min": "2"
        }
      },
      "locationConstraints": "2",
      "placement": {
        "node_name": "0fjval-vm-w-1",
        "cpu": "2",
        "ram": "2",
        "storage": "2"
      }
    }
  ],
  "applicationId": "708f06f0-57b2-4abb-8acf-e86625a11beb",
  "applicationName": "6green_VAO_BSSF",
  "collocationConstraints": [
    [
      "1465",
      "1464"
    ],
    [
      "1466"
    ]
  ],
  "interactions": [
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "0",
      "destinationComponentId": "1465",
      "interactionType": "CORE",
      "sourceComponentId": "1464"
    },
    {
      "QoSConstraints": {
        "bandwidth": {
          "max": "1.0",
          "min": "1.0"
        }
      },
      "interactionId": "1",
      "destinationComponentId": "1464",
      "interactionType": "ACCESS",
      "sourceComponentId": null
    }
  ],
  "numberOfComponents": 3,
  "numberOfInteractions": 2,
  "resourceUnits": {
    "bandwidth": "Mbps",
    "cpu": "m",
    "memory": "Gi",
    "storage": "Gi"
  },
  "sliceProfiles": [
    {
      "enabledUEList": "[{ ICCID : * }]",
      "guaranteedBandwidth": "1.0",
      "isolationLevel": "NOISOLATION",
      "sliceType": "EMBB"
    }
  ]
}
```

EGMF

EGMF Services	Operations	Consumers
Negmf_AccessProvMns	Providing Access for trusted or untrusted VAOs to 6Green NFs	VAO

Negmf_AccessProvMns

<table>
<tr><td><b>Endpoint URI</b></td><td><pre>$APISERVER/apis/athena.trirematics.io/v1/namespaces/6green/egmfs</pre></td></tr>
<tr><td><b>Method</b></td><td><pre>POST/GET/Delete/Patch</pre></td></tr>
<tr><td><b>Request Body (Example)</b></td><td><pre>
{
  "apiVersion": "athena.trirematics.io/v1",
  "kind": "EGMF",
  "metadata": {
    "name": "remotedesktop",
    "namespace": "6green"
  },
  "spec": {
    "vao_identity": {
      "username": "remotedesktop",
      "vao_description": "remote desktop VAO provding service for sharing screens."
    },
    "category": "trusted"
  }
}
</pre></td></tr>
<tr><td><b>Response Code and Body (Example)</b></td><td><pre>
Status code: 201, Created
{
  "apiVersion": "athena.trirematics.io/v1",
  "kind": "EGMF",
  "metadata": {
    "name": "remotedesktop",
    "namespace": "6green"
  },
  "spec": {
    "vao_identity": {
      "username": "remotedesktop",
      "vao_description": "remote desktop VAO provding service for sharing screens."
    },
    "category": "trusted"
  },
  "status": {
   "ca.crt":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJVjc3RFlaa0
FNYUF3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6Q
WVGdzB5TlRBM01EUXhNREExTWpOYUZ3MHpOVEEzTURJeE1ERXdNak5hTUJVeApFekFSQmdOVkJB
TVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3QXdnZ0VLCkF
vSUJBUUNnOWY1UE90Y0FOVlNSejMyR2tJbUZUQk13bWRadGtpWCtkdkVCZlhTM2lQYURPbCtVci
tkMGdxQUsKbGdrc0RoTlpycGExVlBhN0JDMVZHVWU1anpwNGR0U2h5Q2FycWdKd3phUGorQnNoO
WZZNG0xQXNGVUJMK09vbwpMTVlZcy9sOUdybHptN1p5Mi9XS0g1bFViVVlIS2YvaU5zeHpXOHRt
UHZwR3A3TDlvWlRlUEg1WTNudlpGd0hoCnFucXYvMlNHRHhYekJ6aEp2VFlqMGlpR09WcDQvUDJ
oK2dBeE5sS0M1bFhSaHdqNHU3Yi9UNG8xUGpRlcxQ1QKK2hUcnZFVTNxQXY2VURxUzVLS01XUk
djaGNadGpkYkxoWDE2dEE1dzE1d3hDK2hVRE1vbzBCV0Y4QlhEZFo3cgo3bE9sUDNTaUc2NWpIQ
2RaQlE4RmNDYUhXK2pOQWdNQkFBR2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQQkJnTlZI
Uk1CQWY4RUJUQURBUUgvTUIwR0ExVWREZ1FXQkJUUtrenU2REl4WGpnWXlYdDJNTjlZa3FFZFZ
qQVYKQmdOVkhSRUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQk
FRQVdocVFhRWxJaQpwZWd6dmF6NEtwV1EvZHliemRaaHJ2RW8zUWxjU3NKdzRtN2ZVOUJLWTQ2R
Ug0dmVnVG9kZ2hLeklLZWpBdkdLCmpjVnoyTnI1MWt0djBJREFOMlFKQXVhTzMvdlcvZWs0ODdk
</pre></td></tr>
</table>

```
bzRGNUg0SDFXWEIyTitORitncjBGdHBQTFlZZzYKTVVxRDF0enlBT0FtUTBIaWJ4bkRjQmtIUlB
FYzRzSGJiS2JnNTNSb0NuTXBMOExJWXZUWC8wc01meTB6SDJTQgpaQ3ZIZkdVK2JwMnc2YzZCSC
tIK3NXcWNRK3pUZ25lMWFpR2NKYS9FVFJDRzZTWFNhbk4wdW5xbU9Oekd3bDhxCitrelJJM0tTc
WF1bGV1cDdWZHNNU1V6eTZkbzFoaHZFMUgwaTcyQWdFRWtGaXJmYkoxbVJ5a1VDSEh5STdlaEMK
THlTb2Rpei9LSUJ1Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K",
    "token":
"ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXRwWkNJNklsRXpkVXhIVmpScWQzTlJSak5FZGpGT1JFWm
ZYMmw1WmtaaE5VVkpZV1ZqYldWR01ETmpWRVJ3VFhjaWZRLmV5SnBjM01pT2lKcmRXSmxjbTVsZ
EdWekwzTmxjblpwWTJWaFkyTnZkVzUwSWl3aWEzVmlaWEp1WlhSbGN5NXBieTl6WlhKMmFXTmxZ
V05qYjNWdWRDOXVZVzFsYzNCaFkyVWlPaUowY21seVpXMWhkR2xqY3lJc0ltdDFZbVZ5Ym1WMFp
YTXVhVzh2YzJWeWRtbGpaV0ZqWTI5MWJuUXZjMlZqY21WMExtNWhiV1VpT2lKbmNtOTFjQzF2Ym
1VdGRHOXJaVzRpTENKcmRXSmxjbTVsZEdWekxtbHZMM05sY25acFkyVmhZMk52ZFc1MEwzTmxjb
lpwWTJVdFlXTmpiM1Z1ZEM1dVlXMWxJam9pWjNKdmRYQXRiMjVsSWl3aWEzVmlaWEp1WlhSbGN5
NXBieTl6WlhKMmFXTmxZV05qYjNWdWRDOXpaWEoyYVdObExXRmpZMjkxYm5RdWRXbGtJam9pTUR
oaE5Ea3hPR1F0Tm1ZM1ppMDBOVEZoTFRsaU4yVXRaRFkzWVdGbFpUSXlaVFJtSWl3aWMzVmlJam
9pYzNsemRHVnRPbk5sY25acFkyVmhZMk52ZFc1ME9uUnlhWEpsYldGMGFXTnpPbWR5YjNWd0xXO
XVaU0o5Lk53em1rUTdfbEl3TWJ2VXQzZTRqelpIT0F3OGlwZVNZVENjUXRSb3o1akNzcVE3VjJC
N19DMVpMX0JDOWhyTkwtLW93RUlrWkpnbll4QmJvR0t2YnpRVTFzbVZLV3lIM3I0aVgzWnNWNG9
JRW13czUwZVJXWnBNX2lVWjF5bFZTUmI0ZmNPZk9MZkNXaFh6dGJCbkE5MFp3bEl6SUFEZ0QyNH
BUSkp2Z2hnSEZoU2ZZdy1ERjQ4eVFPTWswQnVlMDNCZGRPa2ozbEg2TU1RSFhTS2xRVXdEMjNxc
FZjUnRsLWNEZlVPM0hEU0NiVW9aUk5PR0tpQU5jNU10ODFTMjNyTlM3XzhRM3Bfb0c0VGJ1LUxxQ
SmJLUUdVMjA0UU1WdzB2Z3J2aUdINVItMzdOa2NQbmxSbldkU05DQnctUURYVEZ2Qk5jQTlMYVp
fOHRPMGR4N3hEZw=="
  }
}
```

NSMF

NSMF Services	Operations	Consumers
Nnsmf_NSProvMns	AllocateNsi	BSSF
	DeallocateNsi	BSSF
	GetStatusNsi	BSSF

Nnsmf_NSProvMns

Endpoint URI	$APISERVER/apis/athena.trirematics.io/v1/namespaces/6green/serviceprofiles
Method	POST/GET/Delete/Patch
Request Body (Example)	

```
{
  "apiVersion": "athena.trirematics.io/v1",
  "kind": "ServiceProfile",
  "metadata": {
    "name": "securitycamera2",
    "namespace": "6green"
  },
  "spec": {
    "coverage": [
      {
        "region": "biot",
        "zones": [
          {
            "name": "eurecom",
            "user-density": 1
          }
        ]
      }
    ],
    "user-equipment-type": "rfsim",
    "reporting-period": "620m",
    "data-flows": [
      {
        "name": "securitycamera-traffic",
        "description": "Data flows designated for security camera traffic.\n",
        "data-network": "miot",
        "traffic-class": "persistent",
        "content-rate": {
          "downlink": {
            "maximum": "10Mbps",
            "average": "10Mbps"
          },
          "uplink": {
            "maximum": "10Mbps",
            "average": "10Mbps"
          }
        },
        "latency-class": "mec",
        "payload-types": [
          {
            "use-case": "video-stream",
            "application-protocol": "srtp",
            "transport-protocol": "udp",
            "content-type": [
              "video/h264",
              "audio/opus"
            ]
          }
        ]
      }
    ]
  }
}
```

Response Code and Body (Example)

```
Status code: 201, Created
{
  "sliceProfiles": [
    {
      "sliceId": "000002",
      "sliceType": "MIOT",
      "dnnList": [
        "miot"
      ],
      "profileParams": {
        "isolationLevel": "ISOLATION",
        "sliceAmbr": "80 Mbps",
        "ueAmbr": "80 Mbps",
        "maximumNumberUE": 1,
        "pduSessions": [
          {
            "pduSessionId": "0",
            "pduSessionAmbr": "80 Mbps",
            "flows": [
              {
                "flowId": "1",
                "ipAddrFilter": "8.8.4.4",
                "qi": "15",
                "gfbr": "10 Mbps"
              }
            ]
          }
        ]
      },
      "locationConstraints": [
    {
     "geographicalAreaId": "99",
     "tai": "00101000099"
    }
   ],
   "enabledUEList": [
    {
     "iCCID": "*"
    }
   ]
  }
 ]
}
```

NFMF

NSMF Services	Operations	Consumers
Nnfmf_ProvMns	WMI (Workload Management Interface)	NSMF
	CPI (Container Probe Interface)	NSMF
	MCI (Management & Composition Interface)	NSMF

Nnfmf_ProvMns

Endpoint URI	`MCI: /api/v1/mci/resolve/protocol`
	`CPI: /api/v1/cpi/{healthAliases}`
	`WMI: /api/v1/wmi/start/{NF}, /api/v1/wmi/stop/{NF}, /api/v1/wmi/status/{NF}, /api/v1/wmi/init/{NF}`

Method	`POST/GET`

Request Body (Example)	`MCI: Get http://nrf.ogs-nrf-shared-50331649.networkso.t9s.local.:60001/api/v1/resolve/eth0`
	`CPI: GET http://127.0.0.1:60003/api/v1/cpi/health`
	`WMI: POST http://127.0.0.1:60002/api/v1/wmi/start/smf`

Response Code and Body (Example)	`Status code: 201, Created`
	`MCI: [INFO] [logger:dependency-resolver.amf] [name:nrf.ogs-nrf-shared-50331649.networkso.t9s.local.] [protocol:eth0] [strength:"weak"] Dependency is satisfied`
	`CPI: [INFO] running`
	`WMI: [INFO] [logger:manager.WMI] [client-ip:127.0.0.1] [latency:1253138] [method:START] [path:Process for app amf started successfully with pid 681] [status:202] [time:2026/01/15 - 17:27:43]`

EdgeMF

EdgeMF Services	Operations	Consumers
Nedgemf_EdgeCapability	Compute resources in geographical areas	BSSF
Nedgemf_EDNManagement	Manage Edge Data Networks (EDN)	EGMF
Nedgemf_EdgeClusterManagement	Manage compute resources (clusters)	BBSF / ENIF

Nedgemf_EdgeCapability

Endpoint URI	`<apiRoot>/v1/geographical_areas`
Method	`GET`
Request Body (Example)	*`Empty`*

Response Code and Body (Example)	`[`
	`   {`
	`      "geographicalAreaId":"my-area-id",`
	`      "locationName":"my-location-name",`
	`      "cluster":{`
	`         "name":"my-cluster",`
	`         "type":"KUBERNETES",`
	`         "nodes":[`
	`            {`
	`               "name":"r1hpgen9-4",`
	`               "labels":{`
	`                  "kubernetes.io/arch":"amd64",`
	`                  "beta.kubernetes.io/os":"linux",`
	`                  "kubernetes.io/hostname":"r1hpgen9-4",`
	`                  "status/capacity/cpu":"48",`
	`                  "kubernetes.io/os":"linux",`
	`                  "status/capacity/hugepages-2Mi":"0",`
	`                  "area":"4",`
	`                  "beta.kubernetes.io/arch":"amd64",`
	`                  "status/capacity/pods":"110",`
	`                  "status/capacity/memory":"462025428Ki",`
	`                  "status/capacity/ephemeral-storage":"3074898844Ki",`
	`                  "status/capacity/hugepages-1Gi":"0",`
	`                  "node-role.kubernetes.io/control-plane":""`
	`               }`
	`            }`
	`         ]`
	`      },`
	`      "latitude":44.40478,`
	`      "longitude":8.94439,`
	`      "coverageRadius":50,`
	`      "segment":null`
	`   }`
	`]`

Nedgemf_EDNManagement

Endpoint URI	`<apiRoot>/v1/edn`
Method	`GET`
Request Body (Example)	*`Empty`*

Response Code and Body (Example)	`{`
	`   "description":"List all EDNs",`
	`   "response":{`
	`      "status":200,`
	`      "body":[`
	`         {`
	`            "id":"edn-1",`
	`            "name":"Edge Node 1",`
	`            "status":"active",`
	`            "serviceAreaId":"sa-1",`

```
            "serviceType":"topological",
            "cellIdentifiers":[
               "cell-1",
               "cell-2"
            ],
            "nodeIdentifiers":[
               "node-1",
               "node-2"
            ]
         },
         {
            "id":"edn-2",
            "name":"Edge Node 2",
            "status":"inactive",
            "serviceAreaId":"sa-2",
            "serviceType":"topological",
            "cellIdentifiers":[
               "cell-3"
            ],
            "nodeIdentifiers":[
               "node-3"
            ]
         }
      ]
   }
}
```

Nedgemf_EdgeClusterManagement

Endpoint URI	`<apiRoot>/v1/clusters`
Method	GET
Request Body (Example)	*Empty*

Response Code and Body (Example)	[
	   {
	      "id":"cluster-1",
	      "name":"Edge Cluster 1",
	      "type":"kubernetes",
	      "capacity":100,
	      "edns":[
	         "edn-1",
	         "edn-2"
	      ]
	   },
	   {
	      "id":"cluster-2",
	      "name":"Edge Cluster 2",
	      "type":"docker",
	      "capacity":50,
	      "edns":[
	         "edn-3"
	      ]
	   }
	]

CCMF

CCMF Services	Operations	Consumers
Nccmf_CapabilityExposure	Provide K8s clusters in geographical areas	EdgeMF

Nccmf_CapabilityExposure

<table>
<tr><td><b>Endpoint URI</b></td><td><code>&lt;apiRoot>/geographical_areas</code></td></tr>
<tr><td><b>Method</b></td><td><code>GET</code></td></tr>
<tr><td><b>Request Body (Example)</b></td><td><i>Empty</i></td></tr>
<tr><td><b>Response Code and Body (Example)</b></td><td>

```
[
   {
      "geographicalAreaId":"my-area-id",
      "locationName":"my-location-name",
      "latitude": 44.40478,
      "longitude": 8.94439,
      "coverageRadius":50,
      "segment":null,
      "cluster":{
         "name":"my-cluster",
         "type":"KUBERNETES",
         "nodes":[
            {
               "name":"r1hpgen9-4",
               "labels":{
                  "area":"4",
                  "beta.kubernetes.io/arch":"amd64",
                  "beta.kubernetes.io/os":"linux",
                  "kubernetes.io/arch":"amd64",
                  "kubernetes.io/hostname":"r1hpgen9-4",
                  "kubernetes.io/os":"linux",
                  "node-role.kubernetes.io/control-plane":"",
                  "status/capacity/cpu":"48",
                  "status/capacity/ephemeral-storage":"3074898844Ki",
                  "status/capacity/hugepages-1Gi":"0",
                  "status/capacity/hugepages-2Mi":"0",
                  "status/capacity/memory":"462025428Ki",
                  "status/capacity/pods":"110"
               }
            }
         ]
      }
   }
]
```

</td></tr>
</table>

References

[1] 3GPP TS 23.501 System architecture for the 5G System (5GS), Release 18

[2] 3GPP TS 23.502 Procedures for the 5G System (5GS), Release 18

[3] 3GPP TS 29.518 5G System; Access and Mobility Management Services; Stage 3, Release 18

[4] 3GPP TS 28.912 "*Study on enhanced intent driven management services for mobile networks*", Release 18.

[5] Operator Framework [Online]. Available: https://operatorframework.io/

[6] 3GPP TS 28.531, "*Management and Orchestration – Provisioning*", Release 18.

[7] 3GPP TS 28.541, "*Management and Orchestration – 5G Network Resource Model (NRM) – Stage 2 and Stage 3*" Release 18.

[8] 3GPP TS 28.533, "Management and Orchestration – Architecture Framework", Release 18.