Deliverable D5.5

Modelling, Design and Implementation of QoE Monitoring, Analytics and Vertical-Informed QoE Actuators

Iteration I

Editors:	Dean Lorenz, IBM Research Haifa (IBM) Salvatore Spadaro, Universitat Politècnica de Catalunya (UPC)
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	28/2/19
Actual delivery date:	
Suggested readers:	Network Administrators, Vertical Industries, Telecommunication Vendors, Telecommunication Operators, Service Providers
Version:	1.0
Total number of pages:	74
Keywords:	Network Slicing, AIOps, Network Slice, 5G, SDN, Cognitive management

Abstract

This document reports the design and prototype implementation of the foundation components that enable SliceNet Quality of Experience (QoE)-aware slice management. These enable the embodiment of the SliceNet Cognition Plane architecture described in deliverable D2.4. SliceNet QoE-aware slice management combines the established MAPE (Monitoring, Analysis, Planning, and Execution) autonomic control loop with state-of-the-art data-driven management and AIOPS (Artificial Intelligence for IT Operations); enabling intelligent, adaptive end-to-end (E2E) 5G slice management with respect to the use-cases (UCs) defined within the architecture of SliceNet. The components included in this report provide Proof-of-Concept (PoC) implementations covering the entire Cognition Plane; in particular, the required analytic methods, the machine-learning (ML) pipeline, QoE optimization, and vertical-informed QoE Actuators.

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number 761913.

Impressum

[Full project title]	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualized Multi-Domain, Multi-Tenant 5G Networks
[Short project title]	SliceNet
[Number and title of work-package]	WP5 - Cognitive, Service-Level QoE Management
[Number and title of tasks]	T5.3. Modelling, Design and Implementation of QoE Monitoring, Analytics and Optimisation Engine; T5.4 Modelling, Design and Implementation of Vertical-Informed QoE Actuators
[Document title]	Modelling, Design and Implementation of QoE Monitoring, Analytics and Vertical-Informed QoE Actuators; iteration I
[Editor: Name, company]	Dean Lorenz, IBM Research - Haifa (IBM); Salvatore Spadaro, Universitat Politècnica de Catalunya (UPC)
[Work-package leader]	Dean Lorenz, IBM Research - Haifa (IBM)

Executive summary

The provisioning of network slices with proper Quality of Experience (QoE) guarantees is seen as one of the key enablers of future 5G-enabled networks. However, it poses several challenges in the slices management that need to be addressed for efficient end-to-end (E2E) services delivery, including estimating QoE Key Performance Indicators (KPIs) from monitored metrics and reconfiguration operations (actuations) to support and maintain the desired quality levels. SliceNet provides a design and implementation of cognitive slice management that leverages Machine Learning (ML) techniques to proactively maintain the network in the required state to assure E2E QoE, as perceived by the vertical customers.

This deliverable defines the overall functional architecture of the SliceNet Cognition Plane, including the interfaces with the other SliceNet architectural components. It specifies the mechanisms and methodologies that allow the integration of cognitive methods into the SliceNet management and control, enabling verticals-oriented, QoE-driven 5G network slicing across multi-operator domains. The SliceNet Cognition Plane adds ML capabilities to the established Monitor-Analyse-Plan-Execute governed by a Knowledge-base (MAPE-K) control loop to implement an autonomic E2E slice management and control. The proposed architecture covers the entire loop, including cognition-based monitoring to obtain QoE KPIs, a ML pipeline for the analysis phase, and an actuation framework.

We report on the implemented Cognition Plane components and developed algorithms. In anticipation of the prototyping of the vertical use cases (UCs) and the implementation of the SliceNet management and orchestration components, we implemented several Proof-of-Concept (PoC) workflows to validate the overall methodologies, infrastructure, and processes, and to demonstrate the algorithms needed to support the UC scenarios. We exercise all phases of the cognition-enabled, QoE-aware MAPE control loop – a ML pipeline, QoE estimation, proactive control, optimization, and triggering of the actuations needed to maintain desired slice QoE properties.

It should be noted, that this deliverable is the first iteration of the overall Cognition Plane implementation. It exercises the logical workflows and represents the first integration phase of the different Cognition Plane components. The next iteration will integrate other SliceNet components and refine the existing analytic methods.

List of authors

Company	Author
Altice Labs SA, Portugal	Rui Pedro, Guilherme Cardoso, Pedro Neves, Nuno Henriques
Eurecom	Xenofon Vasilakos, Nasim Ferdosian
IBM Research – Haifa	Dean Lorenz, Kenneth Nagin
Orange SA, France	Marouane Mechteri, Yosra Ben Slimen
Universitat Politècnica De Catalunya	Albert Pagès, Fernando Agraz, Salvatore Spadaro, Rafael Montero
University of The West Scotland	Antonio Matencio Escolar, Enrique Chirivella Perez, Jose M. Alcaraz Calero, Qi Wang, Ricardo Marco Alaez, Zeeshan Pervez

List of Reviewers

Company	Reviewer
Creative Systems Engineering (CSE)	Kostas Koutsopoulos
Cork Institute of Technology (CIT)	Mark Roddy

Table of Contents

List of figures

List of tables

Abbreviations

5G	Fifth Generation (mobile/cellular networks)
5G PPP	5G Infrastructure Public Private Partnership
AIOPS	Artificial Intelligence for IT Operations
API	Application Programming Interface
C-App	Control Application
CN	Core Network
CP	Control Plane
CPSR	Control Plane Service Registry
CPU	Central Processing Unit
CRUD	Creation, Read, Update and Delete
DDCM	Data-Driven Control and Management
DP	Data Plane
DSCP	Differentiated Services Code Point
DSP	Digital Service Provider
E2E	End to End
ECA	Event-Condition-Action
eNB	Evolved Node B
EPC	Evolved Packet Core
FaaS	Function as a Service
FCA	Flow Control Actuator
FCAPS	Fault, Configuration, Accounting, Performance and Security
IED	Intelligent Electronic Device
JSON	Java Script Notation Object
KB	Knowledge-Base
KPI	Key Performance Indicator
LTE	Long Term Evolution
MANO	Management and Orchestration
MAPE-K	Monitoring, Analysis, Planning and Execution governed by a Knowledge-base
ML	Machine Learning
NFV	Network Function Virtualisation
NS	Network Slice
NSaaS	Network Slice as a Service
NSP	Network Service Provider
NSS	Network Sub-Slice
OAI	OpenAirInterface
OSA	One Stop API
OSM	Open Source MANO
OSS	Operations Support System
OVS	Open Virtual Switch
P&P	Plug & Play
PAP	Policy Administration Point
PCI	Policy Catalogue & Inventory
PCM	Policy Context Manager
PCS	Proactive Control Scheme
PCF	Policy Control Function
PDP	Policy Decision Point
PF	Policy Framework

PoC	Proof of Concept
PR	Policy Recommender
QoE	Quality of Experience
QoI	Quality of Information
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RRM	Radio Resource Management
SDK	Software Development Kit
SDN	Software Defined Networking
SLA	Service Level Agreement
SliceNet	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualized Multi-Domain, Multi-Tenant 5G Networks
SMA	Spectrum Management Application
SNMP	Simple Network Management Protocol
UC	Use Case
UE	User Equipment
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VoIP	Voice over IP

1 Introduction

1.1 Scope

The objective of WP5 is to define and prototype the framework for cognitive QoE-aware slice management and control of the 5G network. WP5 embeds service-level cognitive abilities into the SliceNet architecture, enabling intelligent, adaptive E2E 5G slice management; it harnesses ML methods to allow 5G network slices that can support QoE requirements as demanded by the verticals. In this document we describe our framework and methodology for integrating cognitive methods into SliceNet's management plane, as well as overall design of the software components developed to validate the SliceNet approach and to support QoE-aware slice management. This report covers the first iteration of the SliceNet Cognition Plane; it is intended to be updated as WP6 and WP7 materialize. Figure 1 highlights the logical SliceNet components addressed by WP5 as described in WP2 deliverables (see D2.4, Section 5) [1].

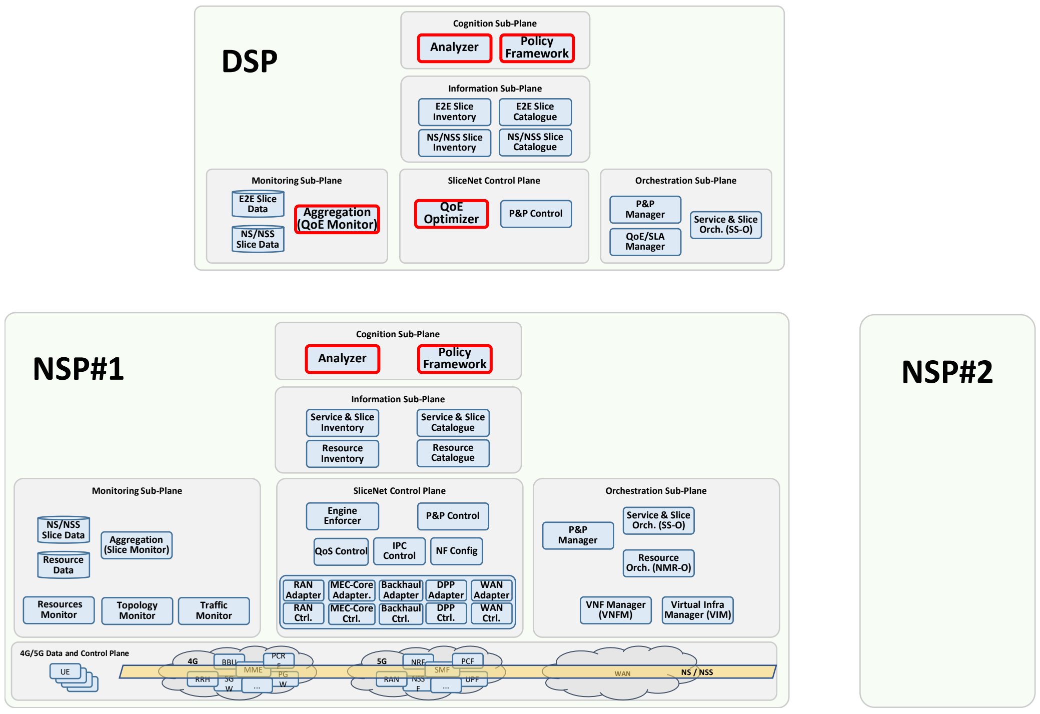


Figure 1 Logical SliceNet components addressed by WP5

1.2 Document structure

This document is structured as follows:

1. The rest of Section 1 provides an overview of the overall design principles of the Cognition Plane architecture and the adopted MAPE-K approach.
2. Section 2 describes the functional architecture of the elements comprising the Cognition Plane and the methodologies used.
3. Sections 3 and 4 describe concrete workflows exercising the functional architecture, providing PoC implementations that validate the methodologies, algorithms, and feasibility. The detailed workflows, and the software components are meant to evolve with the implementation of the SliceNet Management Plane (WP6) and with the prototyping of the vertical UCs (WP7).
4. Section 5 documents the main interfaces that have been designed and implemented for the first iteration of the Cognition Plane.

5. Section 6 summarizes the implemented software modules and PoCs, providing the links for their source code.
6. Lastly, Section 7 contains the main conclusions of the deliverable.

1.3 Overview of SliceNet Cognition Plane

In this section we briefly describe the overall SliceNet approach for applying cognition to 5G slice management and control, as introduced in Deliverable D2.4 [1]. A more detailed description of our methodology is given in Section 2.

1.3.1 Motivation for using cognition

SliceNet envisions an intelligent cost-effective network management, control, and orchestration framework that can cope with the challenges of multi-domain slicing over 5G networks, while minimizing human intervention. Some of the main challenges include:

1. **Flexibility** – 5G networks support many configurations, which leads to a multitude of system states. It is no longer possible to prescribe what needs to be done per every conceivable system state, as there are too many options and heterogeneous cases. The management and control mechanism must be able to flexibly handle states it has never encountered before and for which there are no specific rules (e.g., by applying rules from similar states); thus, it must be able to *generalize* rules and *comprehend* their intent.
2. **Scale** – the sheer scale of 5G networks makes it impractical (even from cost considerations alone) to require human input in the control loop. The control must be *autonomous* and *automated* with humans only prescribing desired behaviour (e.g., as policies).
3. **Dynamicity** – 5G networks support a combination of many types of workloads stemming from a variety of UC. These workloads can come and go and may even change dynamically as needed by the verticals. As a result, the derived requirement from the network may change often and these changes may be significant. The management mechanism cannot focus only on the common case; it must constantly *adapt* to and *anticipate* changes.
4. **Abstraction** – the 5G network is managed through several layers of abstraction. There are multiple information owners providing multiple data sources, each with its own semantics; moreover, depending on a specific role, only partial information may be available from some layers. For example, a Network Service Provider (NSP) that implement a network sub-slice (NSS) might provide only partial network information to the Digital Service Provider (DSP) that manages the E2E network slice (NS).[1] Thus, the management and control, per role, must be able to combine multiple information sources, *interpret* their meaning, and fill in the gaps ("*guess*") when needed.
5. **QoE and QoS** – in order to support service-level E2E QoE, a collection of complex tasks is required; these tasks must translate the vertical E2E UC requirements into network-level Service Level Agreements (SLAs) and concrete KPIs, *estimate* and *predict* QoE KPIs from network-level Quality of Service (QoS) metrics, and optimize the network resources to support the needs of multiple NSes.

Cognitive network management addresses these challenges by utilizing ML to understand the network behaviour and proactively steer that behaviour towards its desired state (see [2] for a more detailed discussion). SliceNet advocates a declarative rather than an imperative approach to network management, where cognition is used to learn the best actions to achieve declared goals; moreover, the goals are abstracted in a user-centric manner to utilize the full potential of 5G slicing and fulfil the "Verticals-in-the-loop" SliceNet vision.

[1] Unless the SLA prescribes it, the NSP is not obliged to expose any details into its network.

1.3.2 Technical approach to Cognitive management

The SliceNet Cognition Plane embraces the MAPE-K approach for automated and autonomic management; MAPE-K is a loop of **M**onitor-**A**nalyse-**P**lan-**E**xecute governed by a **K**nowledge-base that encapsulates policies, rules, algorithms, etc. SliceNet is designed to support ML for the Monitoring and Analysis steps, as well as for creating new Knowledge. SliceNet QoE Monitoring (as described in D5.2 [3]), separates the acquisition of monitoring data from the processing of that data and transforming it into NS QoE metrics. The Analysis step uses the acquired knowledge to assess the NS QoE and possible impact on corrective actions; this is done by both inferring learned cognitive models and by applying more traditional automated management methods. The Planning and Execution steps (termed *Actuation*) are governed through a Policy Framework (PF).

SliceNet employs a Data-Driven Network Operations methodology, a.k.a. AIOPS (Artificial Intelligence for IT Operations) [4]. Network analysis applications react to collected operations data (both raw and processed) and generate new metrics and signals (e.g., QoE metrics and QoE-aware insights) that in turn trigger network operation actions. With this methodology, most components interact only with the data store, acting as consumers and producers. This approach minimizes the direct interfaces, provides flexibility, and easier integration of cognitive tools. It also allows existing techniques to be used with little change, as the outputs of the cognitive tasks can be treated as advanced sensor metrics; indeed, this is how SliceNet's QoE sensors are implemented.

The main components of the Cognition Plane and its relation to the other SliceNet planes are shown in Figure 2. The figure details only the SliceNet logical components that have significant interaction with the Cognition Plane; other components of the SliceNet logical architecture are hinted by empty boxes.

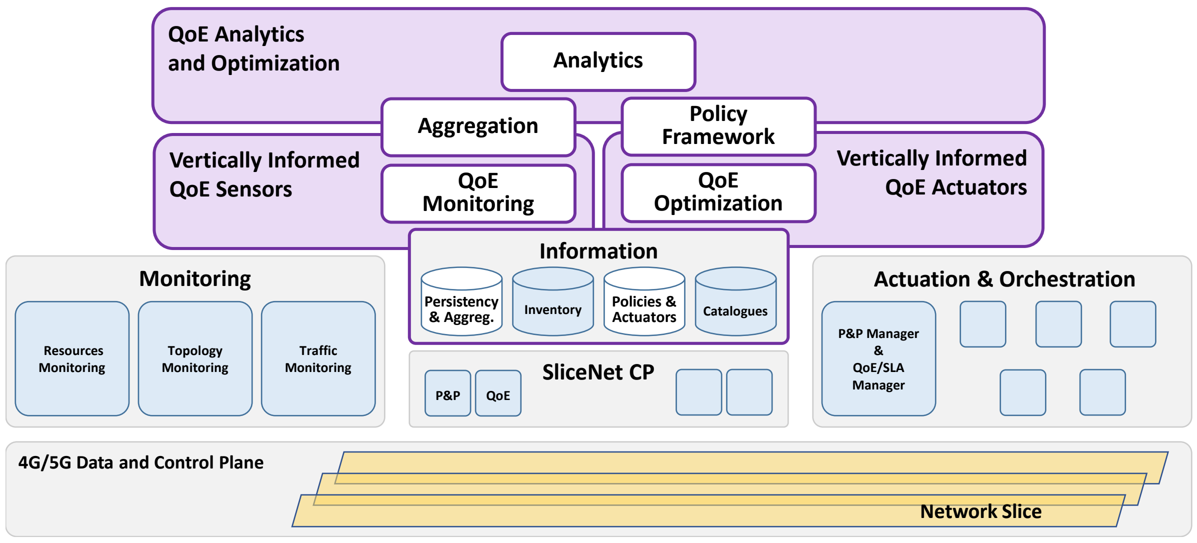


Figure 2 Cognition Plane overview

Vertical informed sensors (as described in D5.2) provide *QoE monitoring* capabilities and prepare *aggregated data* from the ML-based analytic processes. Although monitoring utilizes information from several sources, it has direct interfaces only with the persistent data stores, simplifying the interfaces and following a Data-Lake [5] approach (see Section 2.2).

Analytics and optimization tasks process the QoE sensor data to provide cognitive insights that enable the vertical-informed actuation. Analysis is applied first to historical data to learn new models and derive new policies and optimization parameters. At run-time, similar analysis is applied to verify and refine the learned models. Finally, the learned models are inferred in near-real-time to provide concrete input signals for actuation.

A *policy framework* governs the actuation, applying *QoE optimization* tasks that combine inferred signals with measured KPIs to control vertical informed QoE actuators. Actuation may affect the monitoring tasks (e.g., initialize more sensors) or may trigger new analysis. Actuation may be triggered by feedback from the vertical and may invoke application-specific controls or deliver alerts and SLA metrics to the vertical. These unique vertical-oriented capabilities are achieved through SliceNet's Plug & Play (P&P) framework.

1.3.3 Control (MAPE-K) loops deployment

The MAPE-K approach provides the logical buildings blocks for autonomous cognitive management. SliceNet employs these loops multiple times at different layers of the architecture to achieve various management tasks. Several loops are illustrated in Figure 3.

The first loop (Figure 3a) enables training of ML models – QoE monitoring and aggregation curates the necessary data for the training phase, the results of the ML process are fed back into the knowledge base (K) as policies to be applied in other loops. A similar process (Figure 3b) is the refinement of existing models; this is done periodically (by policy) or may triggered by actuation (e.g., if there are indications that there is too much deviation from model assumptions). Actuation (Figure 3c and Figure 3d) can act "out" invoking external control tasks or act "in", adjusting the Cognition Plan components (e.g., change the monitoring configuration). The actuation can be triggered near-real-time, directly from monitored KPIs (Figure 3c); or follow the full loop (Figure 3d), where monitoring feeds a ML process that infers a previously learned model. The loop may include the vertical (Figure 3e and Figure 3f), either as a source (e.g., providing feedback that may trigger an action) or a destination (e.g., being alerted of imminent failure). The loop may be a real-time Fault, Configuration, Accounting, Performance, and Security (FCAPS) management loop (Figure 3g), where cognition is used to populate configuration and policy parameters; namely, it provides the "K" for this short autonomic control loop.

Finally, it should be noted that the Knowledge may be federated (Figure 3h); for example, a DSP-level E2E NS may combine information from the cognition functions that are implemented at the NSP-level. Moreover, while the focus of WP5 is at the service-level (E2E QoE), cognition is applied at all levels. (1) Cognition is applied at the NSP level to manage resources for multiple NSSes; in this case QoS data is collected from shared resources and is analysed (utilizing ML methods) to determine the optimal resource allocation. (2) Cognition is applied at NSP level for NSS management; in this case NSS data is analysed to maintain the desired NSS SLA. (3) At the DSP level, cognition is applied to infer the E2E QoE KPIs through both estimation and prediction; this can be done either in single domain or multidomain configuration to maintain the E2E SLA and proactively manage the NS QoE.

The different options are further explored in the following sections. Section 2 provides further details on the methodology and the main logical workflows (including the application of Cognition at both NSP and DSP levels), while specific scenarios and workflow implementations are described in Sections 3 and 4.

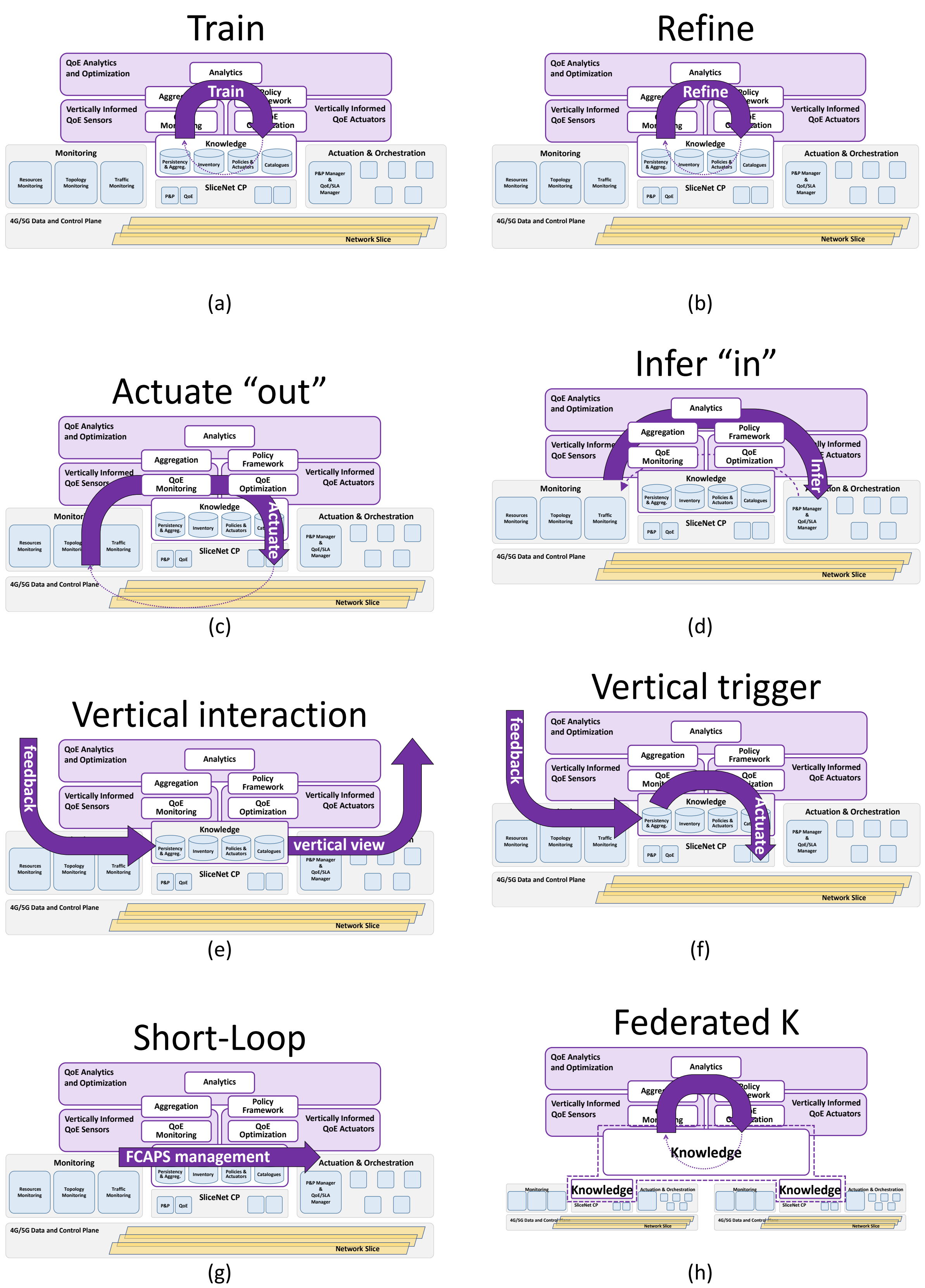


Figure 3 Multiple MAPE-K loops

2 Cognition Plane Architecture and Functional Components

In this section, we provide an in-depth description of the Cognition Plane methodology. First, we explain the application of the MAPE-K control loop to proactive network management and control. Then, we detail our approach to the different elements of the MAPE-K loop and their role in NS QoE management. Finally, we highlight the relevant SliceNet architecture workflows (taken from the on-going WP7 work) showcasing the Cognition Plane role at both DSP and NSP levels.

2.1 Cognitive Control Loop

As described above, SliceNet follows the MAPE-K for automated management of network QoE. SliceNet employs a Proactive Control Scheme (PCS) in managing the run-time lifecycle of a NS, while utilizing cognitive methods to maintain the desired SLA. The main control loop of the PCS is portrayed in Figure 4. The model details how ML models can be integrated into a traditional control loop. Note that this PCS is both reactive and proactive; it can address an already identified issue such as congestion in the Core Network (CN) segment or in the Radio Access Network (RAN) (either at the backhaul or the fronthaul) or it can address imminent or developing problems (e.g., utilizing anomaly detection).

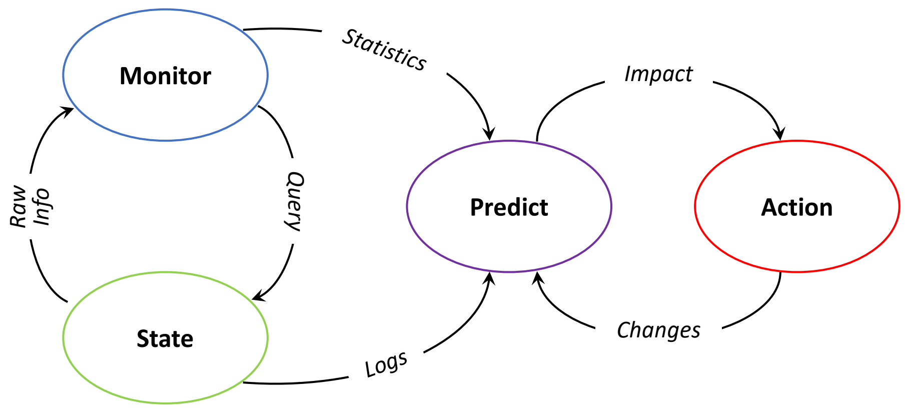


Figure 4 The Proactive Control Scheme

Monitor and **State** generate the data required for analysis. The monitoring component is continuously querying the current **State**, in order to extract raw data about the NS, including resource telemetry, topology, and traffic metrics. For example, it can collect KPIs related to wireless link quality, based on RAN network conditions. In addition to the raw metrics, the **Monitor** component generates calculated KPIs by processing the data as network statistics and by applying aggregations at different levels (e.g., per cell, per slice, etc.). Another important role of the **Monitor** is to enrich its raw input data. One important enrichment example is labelling the raw data, e.g., adding identifiers mapping. The enrichment enables custom aggregations and other advanced statistical methods by analytics tools further down the pipeline. Finally, the **Monitor** controls the granularity of data capture and the aggregation bucket. There is a trade-off between monitoring at the finest possible granularity (which provides the most accurate information) and the resources required to compute, transport and store the data. Therefore, the level of granularity must support dynamic adjustments, depending on the current needs of the cognition loop.

The analytics and ML are encapsulated in the **Predict** component. It further enriches the data, adding insights, predictions, and impact analysis, to allow proactive management with better system diagnostics and prognosis. The calculation applies (infers) already learned ML models, combining the real-time monitored data with log data and historical data (e.g., NS behaviour under normal operating conditions). The enrichment information is added to the monitored data as an extra source for the **Action** component. This approach allows the Cognition Plane to support both real-time and near-real-time models, in which the **Action** component may react immediately to time-sensitive metrics and utilize the results of more complex analytics for proactive actions. For example, a Virtual Machine (VM) Central Processing Unit (CPU) load KPI may indicate that the Virtual Network Functions (VNFs) for creating a slice are overloaded, and more resources are required to maintain the slice SLAs; a predicted load KPI may suggest adding resources before the SLA is affected; and a QoE-impact KPI may predict which resources will mostly affect the QoE as perceived by the vertical. In another example, an anomaly detection alert such as for Evolved Node B (eNB) bandwidth congestion may trigger finer grain monitoring to verify such a potential anomaly and monitor its progress, allowing controlled remedial actions, while still being able to react immediately if needed, such as by performing User Equipment (UE) handovers or turning on more eNBs.

The **Action** component has a dual role, as both (1) an internal ML control loop; and (2) a network control loop for taking network actions. The ML control loop can imply actions such as (re-)validating a ML model in use, e.g. the one used for resource allocation or the one used for predictions. Following validation, it may choose to replace one ML model with another one that better fits the current conditions (e.g. heavy congestion). Other possible actions include changing monitoring granularity, (re-)starting learning, etc. Note that all of these actions are intertwined, therefore they affect each other. Network control loop actions, on the other hand, include examples such as adjusting the transmission power of an eNB, altering cell breathing (range of coverage), adding or reducing CN resources (e.g., CPU cycles for serving packet queues), and so forth. These actions may affect one slice or even all slices (such is the case when altering cell breathing). Finally, the changes caused by actions are fed back to “Prediction” alongside raw data like logs and processed statistics in order to maintain ML models, thus closing the loop and allowing for consecutive iterations of the monitor-predict-action cycle.

2.2 Knowledge & Monitoring

SliceNet employs a Knowledge-Centric, Data-Driven approach to network operations (see Figure 5). A logical Data-Lake data store acts as the Knowledge-base (KB) of the MAPE-K loop. All data sources are logically merged into one data store and analysis outcomes are shared through the same data store. This paradigm is used at the NSP level to manage NSSes deployed at its underlying infrastructures (i.e., FCAPS and SLA), to offer a Network Slice as a Service (NSaaS) towards the DSP level. The DSP constructs and manages E2E NSes and offers a NSaaS to its vertical customers, based on the QoE defined and SLA. The combination of data-store and data processing applications supports several roles within the SliceNet architecture and is implemented across both the Cognition and Management planes.

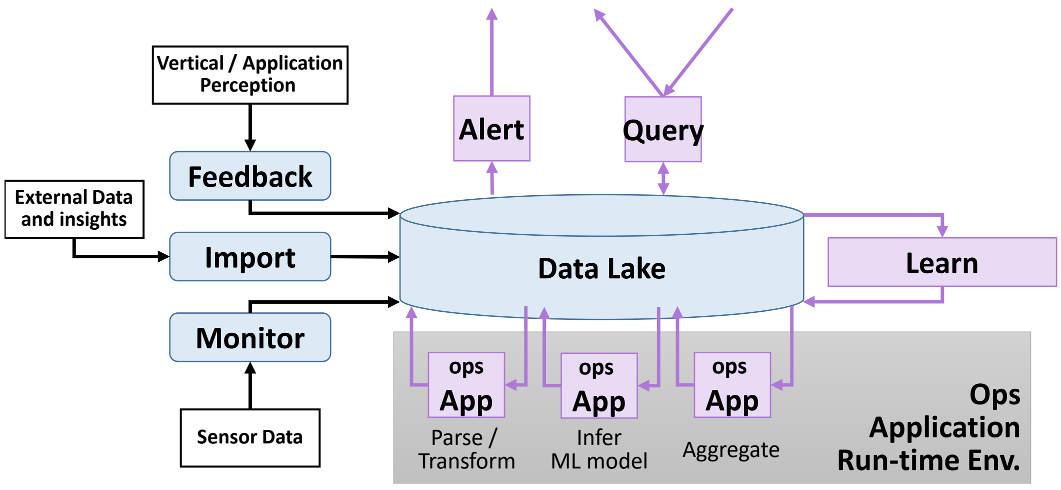


Figure 5 Data-Oriented operational analytics

Multiple data sources are logically merged to provide all the required information for QoE management. Control Plane (CP) and Data Plane (DP) sensor outputs are collected and persisted to support traditional monitoring through parsing, transformation, and aggregation. However, this data is also used for ML model training and for extracting QoS metrics. Feedback from the vertical is combined to allow the data processing application to assume the role of QoE sensors, learning and estimating the vertical perspective. The data source may be fed from external data sources (both raw and processed) that are not created within the SliceNet controlled system. Leveraging external data sources enables the training of ML models by means of historical data of the infrastructure or other deployed services. This allows for QoE NS management under practical limitations, where some information must be curated to hide sensitive data or anonymize. For example, an NSP may not be willing to provide some of its raw network metrics but may share processed alerts. As another example, data from multiple NSes may be merged and provided as an external source, allowing insights from one NS to be applied to another.

Our data-oriented approach, described above, is a continuation of the Monitoring Framework to support QoE sensing, as described in D5.2 [3]. Data-operations applications may be deployed for each NS/NSS to filter relevant data, apply security, add context, aggregate slice metrics, etc. This addresses several of the design challenges related to the monitoring framework (for example, the approach is scalable and allows attributing the cost of monitoring to each slice). Moreover, flexible QoE sensors may be employed; from simple aggregation and transformation tasks to inference of elaborate ML models. Several examples of such sensors are detailed in Section 3.

Ingesting data from external sources is a crucial part of the SliceNet knowledge acquisition process. ML algorithms perform and learn more efficiently with massive amounts of data (Big data); this holds almost independently of the objectives or UCs being addressed. It is unlikely that slices generate enough operations data to support their own learning processes. Thus, there is a need to combine multiple sources. However, data from other slices or data from the underlying slice infrastructure may be subject to confidentiality or privacy limitations. Data ingestion must enforce governance rules dictated by the data owner. In addition, external data may contain corrupt or partial data; thus, it must be parsed, validated, and cleaned before it enters the SliceNet Data-Lake. Finally, the data ingestion must receive the data on the sender’s “terms”; namely, it must handle the data volume and maximal data rates. A detailed example of importing a (real) external data source is described in Section 3.2.

2.3 Analysis

2.3.1 Machine learning pipeline

Having in mind the great degree of flexibility that network slicing brings to the network management domain, we can foresee that dealing with such an environment, where NSes come and go, where their monitoring can live in different time and resource domains, brings a new set of challenges while we try to apply ML techniques to solve, for instance, network optimization scenarios. Due to the presence of these challenges, the need for having an automated ML pipeline arises, not only to deal with network dynamicity but also with the problem scopes that come from different business roles (i.e. DSP and NSP).

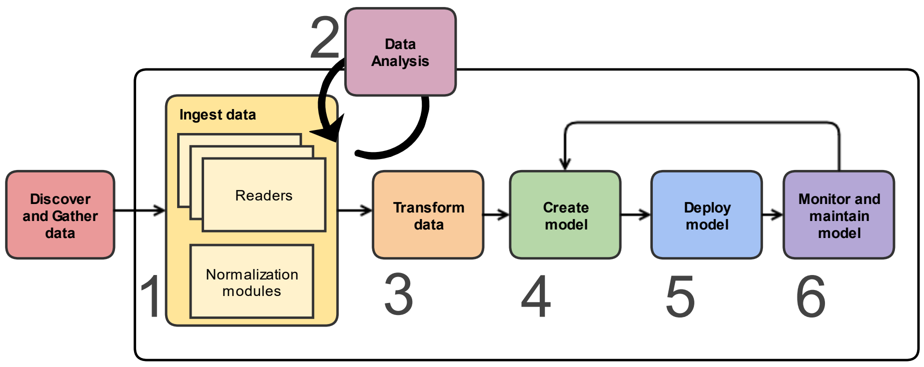


Figure 6 ML pipeline architecture

Figure 6 depicts the ML pipeline defined for SliceNet's Cognition Plane architecture. As a starting point, and external to the pipeline, there is a data discovery and gathering phase, this is where input for ML occurs. Logically, this step represents a data source from the pipeline point of view. Internally, it is divided into six different functional areas, covering all phases from data collection to the ML models' lifecycle:

1. **Ingest data:** this module enables the pipeline to read data and its responsibility is divided into two components:
   a. **Readers:** data input can be multiple files containing observations or streaming data. Each reader abstracts the medium source of the observations and their nuances;
   b. **Normalization modules:** data normalization is the process of combining, merging, and cleaning, according to the knowledge gathered from the data analysis. Includes removing duplicate observations, removing invalid and/or badly formed data;
2. **Data analysis:** the initial analysis serves the purpose of gaining data insights and further problem contextualization. This module runs statistical queries (i.e. counting, averaging, grouping), to check if the dataset is balanced, incomplete or how to focus its modelling;
3. **Transform data:** data transformation depends on data analysis and problem objectives. This module transforms the data into ML-ready. This is where features are extracted and their normalization (e.g. ordinal, one-hot encoding) happens;
4. **Create model:** ML algorithms, which can cover the classification, prediction or clustering ML areas, are applied in this phase. This is where models are effectively trained, optimized (i.e. hyperparameter tuning) and their testing strategies are put in place: cross-validation, feature importance analysis, dimensionality reduction and so on;
5. **Deploy model:** during the training/testing phase, if a model shows significant fitness metrics values it can then be deployed into production and start being used to predict, classify or cluster data in the real-time problem domain;
6. **Monitor and maintain model:** deployed models can lose their effectiveness over time, especially when the data domain is too volatile and dynamic, this means that certain models may be unfit for usage since they no longer properly represent the real world. When models show fitness metrics that are below the configured acceptable values, they are archived, and a re-training task is scheduled to update them. As identified in Figure 6, when such a situation occurs, the process reverts back to step 4, the "**Create model**" phase.

2.3.2 Analysis tools and model training

Following the ML pipeline design presented in Section 2.3.1, this section delves further into the model training process, also referred as the "**Create model**" phase as identified in Figure 6. In order to further explore this process, Figure 7 zooms into phase 4 of the ML pipeline.

Figure 7 showcases the details regarding the “**Create model**” phase, as seen in Figure 6, which handles all processes related to ML model training. Its design is specially focused on training several models at the same time, enabling a way to compare different training strategies at run-time. This allows the pipeline to be flexible enough to onboard different ML frameworks and approaches when solving cognition problems. It also aligns the pipeline design with the emerging automated ML concept in the ML community, which brings automation to the process.

Furthermore, Figure 7 also depicts a simple example in order to fully understand how the design maps into the implementation. It showcases three different processes that use different frameworks, algorithms and data structures to train ML models. After the training ends the best model is shipped into phase 5, to be deployed.

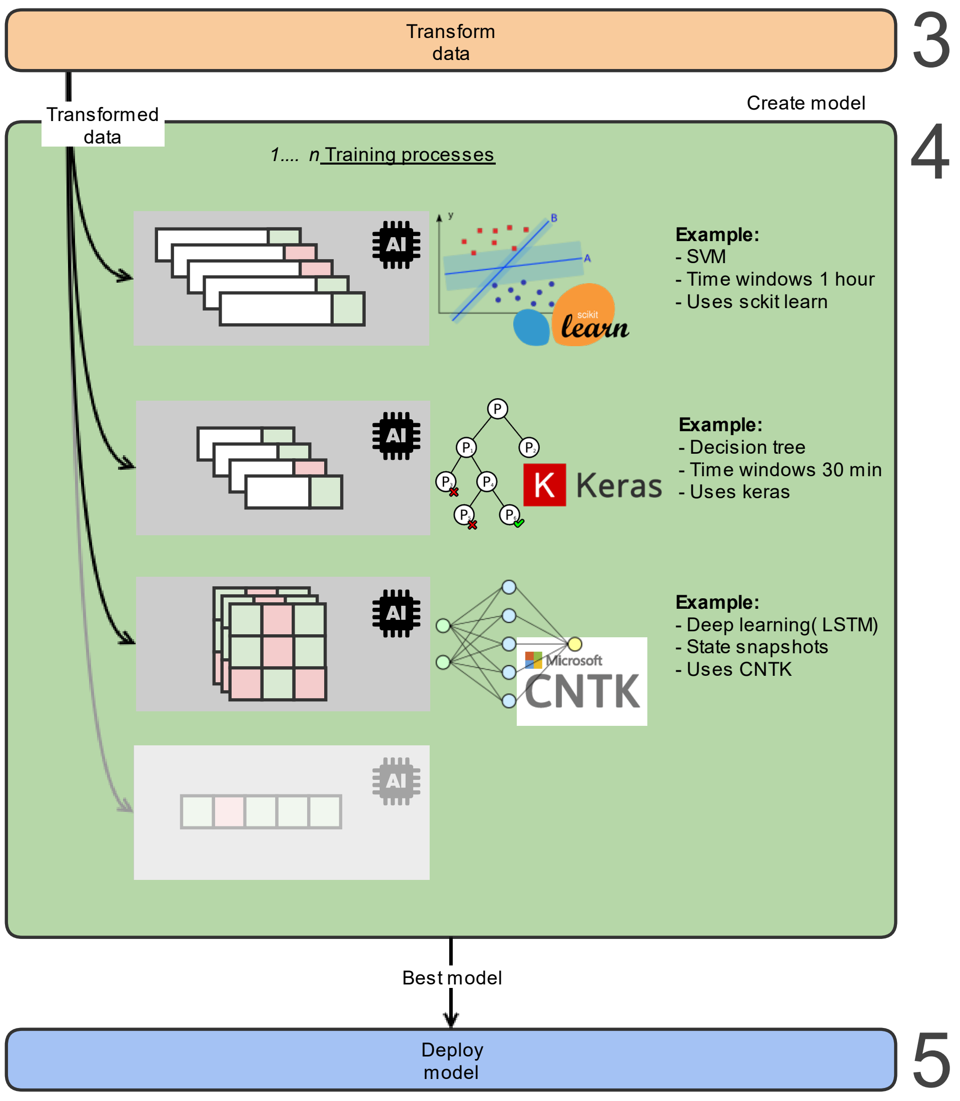


Figure 7 ML pipeline - Model training

2.3.3 Model application / inference

The Cognition Plane architecture uses data processing applications not only to process data for training, but also to infer the learned models at slice run-time. This approach provides a flexible run-time environment that can support complex cognitive models as well as simple rule-based optimizations. It also has the advantage that the processing resources required to build the analytic data pipeline can be attributed to the slice(s) that consume the resulting information. The orchestration and life-cycle management of these cognition-plane applications has similarities with the orchestration and management of the slice CP and DP VNFs, as exercised through the Network Function Virtualization (NFV) Management and Orchestration (MANO) architecture [6]. Furthermore, it requires functionalities of P&P, since these data processing applications are slice specific and dynamic. Note that this run-time environment is only logically centric; actual deployment may be distributed as needed on the slice infrastructure. The analytic models are just functions that process data events and are thus compatible with a Function-as-a-Service (FaaS) execution environment. Finally, more traditional control mechanisms (such as threshold-based triggers and rule-based alarms) and management utilities (such as user-defined alerts and dashboards) can also be implemented as data-processing applications.

2.3.4 Data-Driven Network Control and Management

To enable run-time execution of ML models and functions/operations/processes, SliceNet introduces a Data-Driven Control and Management (DDCM) loop, which employs prediction-based decision policies and network services, customized for slices. Figure 8 depicts the DDCM loop and illustrates the process of cognitive and autonomic network control. It also shows how to apply the cognitive control process pattern of PCS through the DDCM loop, focusing on the overall Cognitive Plane. DDCM is a combination of analytic techniques and modelling approaches on top of SliceNet's CP and DP, making it possible to actuate over the physical network infrastructure domain (e.g. RAN, CN). Various methods for supporting network slicing in the RAN and the CN segments have previously been explained in deliverable D4.2 [7]. Here, network slicing over a cognitive control and management-ready architecture allows operators to create customized control and management models for different slices.

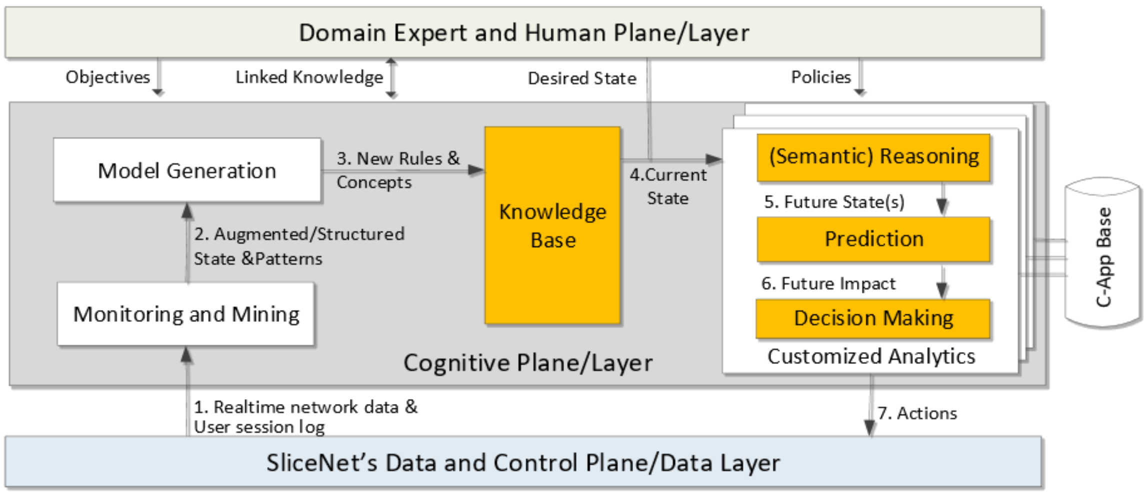


Figure 8 Cognition Plane entailing Data-Driven Control and Management loop

The steps of the DDCM loop over the Cognitive Plane, depicted in Figure 8, are described as follows:

1. **Real-time network data & User session logs** are monitored from DP and CP.
2. **Monitoring and Mining** consists of the **Monitoring** module that extracts and arranges raw data from heterogeneous sources; and the **Mining** module that performs a chain of certain mining operations on raw data, such as data cleaning, pre-processing and transformation, in order to prepare data for a later analysis stage, for example in some data-set D.
3. **Model Generation** is a module that works on the previously obtained data-set D. This data-set is the descriptor of the current network state used to discover new semantic rules and concepts which will be added into a KB as new updates. It is worth noting that the discovered knowledge can be encoded in a description model such as an ontology that is suitable for knowledge

representation. Any proper statistical and analytics techniques can be used for this module to select the most appropriate data features and their inter-relations. For example, the network input variables which affect network KPIs (e.g., throughput) are defined and the most effective ones are selected as the most appropriate data features out of the high dimensional (in terms of number of features) dataset. Traditional supervised ML and statistical analysis algorithms can also be used to define how the pre-defined effective network input variables affect network KPIs.

4. **Current network state** from the KB and **desired state** from **Domain Expert and Human Plane** are sent to **Customized Analytics**. The desired state is deduced out of the services-based objectives, policies and the linked knowledge from other network domains.
5. **Reasoning** module checks and determines if the current state of the network is following the targeted optimization goal (network KPIs) and predefined policies. It goes through the updated KB, and extracts the respective inferences, which define the **future states** of the network. In this way, the decision maker is able to assess how far the current network state is from the desired state (global network optimization goals).
6. After this, the **Prediction** module predicts the future impact of the inferred future state of the network. By using proper ML algorithms, such as regression algorithm, the future states' input data variables can be mapped to some functions, which predict their future real impact.
7. Finally, the **Decision-maker** defines a vector of control actions including a set of network configurations and parameter settings, and sends them to their respective controllers.

Control Applications (C-Apps), performing different network control operations (e.g. mobility management, handover management, policy and charging), provide an abstraction layer over the underlying network and controllers to facilitate the RAN and CN programmability, as well as the interaction with SliceNet DP and CP [7]. Note that C-Apps, residing in C-App Base, includes all "Reasoning", "Prediction" and "Decision-maker" modules, having a set of predefined Application Programming Interfaces (APIs) between them, and translates high-level/technology-agnostic policies and service definitions to low-level/technology-dependent ones RAN and the Evolved Packet Core (EPC).

The described Cognitive Plane involving DDCM loop can be followed, as a baseline approach, by all centralized and distributed network management and control solutions (for slices and services) to make global and local control decisions and satisfy network optimization goals.

2.4 Planning & Execution

2.4.1 Actuation framework and vertical-informed actuators

The Planning and Execution phases of the MAPE-K loop are responsible for planning required (re-)configurations of the network infrastructure, deployment of new elements and functions on the configured services to remedy undesired situations (faults, underperformance, etc...) and performance optimization of the overall system.

The complexity and dynamicity of NSes require autonomous loops for enforcing concrete actions on the underlying infrastructure and deployed services to ensure an optimal system performance. Moreover, given the more user-oriented perspective of 5G networks, in which the delivery of quality guarantees towards users/customers is a key aspect, both planning and execution phases must be built around the concept of **quality optimization** at all levels.

SliceNet defines an actuation framework, which covers both planning and execution phases, with the core goal of maintaining and optimizing the perceived QoE by vertical customers. To this goal, the actuation framework focuses on determining the required changes to E2E NSes that support the verticals' services, while taking charge of enforcing such changes through the appropriate means. The actuation framework is designed with two main components in mind.

1. The PF component is a rule-based policy engine, in which rules define what actions are executed in response to system and NS events.
2. The QoE Optimizer component is responsible for all (re-)configurations necessary to maintain the QoE of a specific E2E NS. Thus, given the rules specified by the PF, and monitoring data gathered both from the SliceNet monitoring stack (specifically, the QoE Monitoring) and verticals' feedback (through the P&P controller), the QoE Optimizer triggers the necessary actions to carry out the desired actuations.

Figure 9 depicts a schematic of the logical architecture of the actuation framework, also depicting the main functional blocks of the SliceNet architecture with which it interacts. Main cross-module interactions are also highlighted.

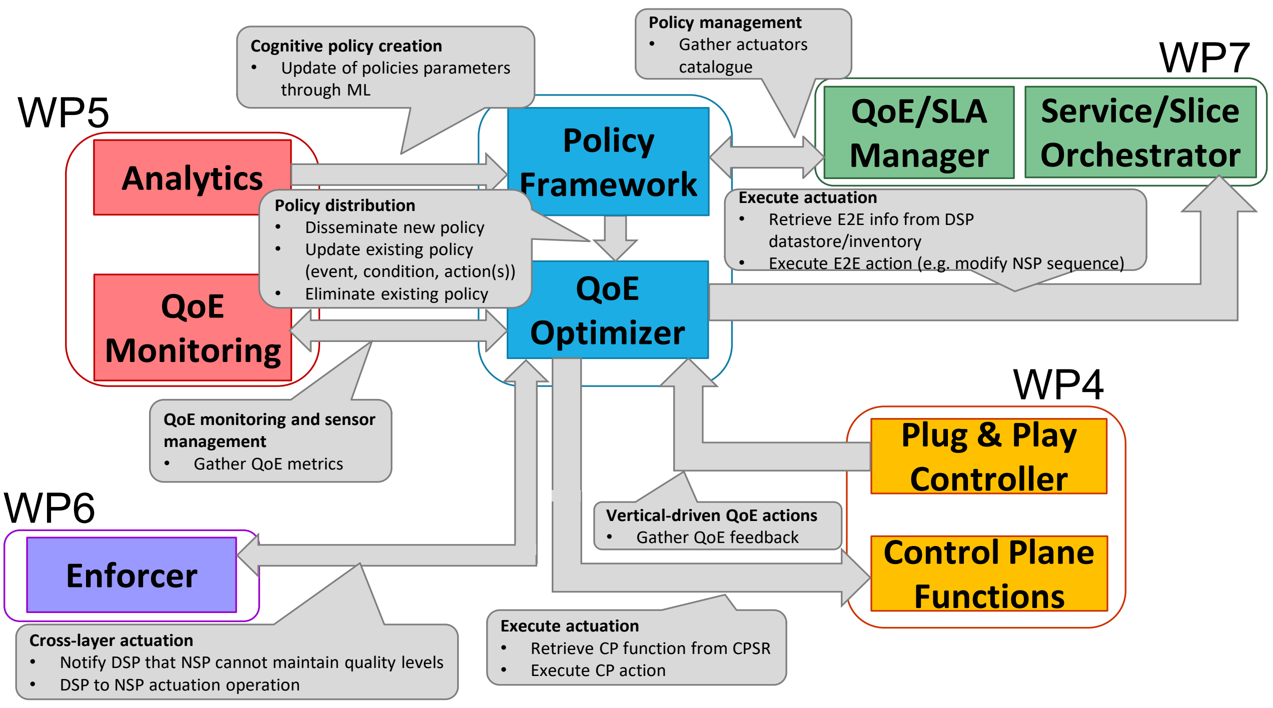

Cognitive policy creation
• Update of policies parameters through ML
Policy management
• Gather actuators catalogue
WP7
QoE/SLA Manager
Service/Slice Orchestrator
WP5
Analytics
Policy Framework
Policy distribution
• Disseminate new policy
• Update existing policy (event, condition, action(s))
• Eliminate existing policy
Execute actuation
• Retrieve E2E info from DSP datastore/inventory
• Execute E2E action (e.g. modify NSP sequence)
QoE Monitoring
QoE Optimizer
WP4
QoE monitoring and sensor management
• Gather QoE metrics
Plug & Play Controller
WP6
Vertical-driven QoE actions
• Gather QoE feedback
Enforcer
Control Plane Functions
Cross-layer actuation
• Notify DSP that NSP cannot maintain quality levels
• DSP to NSP actuation operation
Execute actuation
• Retrieve CP function from CPSR
• Execute CP action

Figure 9 Schematic of the logical architecture of the actuation framework

While the planning, decision and trigger of the actions is done within the core actuation framework (PF and QoE Optimizer), the actual execution of the actions is carried out by specific functions at both SliceNet CP and orchestration layer. The PF and QoE Optimizer get their input from both analytics and external monitoring data to determine when and how actuations should be carried out. Then, the actual actuation is triggered by timely collaborations across functions at different layers. The next subsections elaborate on both the PF and QoE Optimizer as central parts of the whole actuation framework.

In a nutshell, the rules defined by the PF are translated into operations that the QoE Optimizer can trigger/execute. These trigger/executions are abstractions of the operations exposed by execution points (mainly, CP and orchestration layers). A clear definition of the exposed operations, including its context within a particular NS, is required to compose the policy rules. To this end, an actuator catalogue is to be designed. This actuator catalogue is intended to provide all the details of the abstracted actions regarding their type and main parameters that are accepted. Table 1 depicts an example of the catalogue, providing descriptions for all the elements of the catalogue entries. Given this, it will be possible to construct policies with a specific target action in mind, tailored to the context of the NS for which they are intended, thus customizing the whole QoE optimization to the type of service/slice deployed. The presented catalogue will reside on the information sub-plane as part of the SLA/QoE Manager entity at the orchestration plane, which is responsible for the management of the lifecycle of QoE Optimizer instances, as well as maintaining the consistency of the actuators repository, or directly as part of the repository maintained by the PF (explained later in Section 2.4.2). For the present iteration, only the design details are presented. More specific details, mainly, on the data model followed and the concrete structure of the table as well as the entity responsible for it are tasks still under development and will be presented at later iterations of the Actuation Framework.

Table 1 Example of actuators catalogue

Family	**Class**	**Type**	**Description**	**Parameters**
Family of the actuator related to the enforcement point that will be contacted [Control Plane, Orchestrator, …]	Class of the actuator related to function/operation to influence [QoS, IPC, NF Control, …]	Particular type of action inside the class (modifyBW, modifyPriority, …)	String describing the actuator scope (optional)	List of variable parameters that the actuator accepts Parameter: - Name - Type - Value
…	…	…	…	…
Control Plane	QoS	changeBW	Changes the bandwidth of the stated segment(s) by the stated value	Parameter1: - Name: SegmentID - Type: String - Value: X Parameter2: - Name: Segment Technology - Type: String - Value: Y Parameter3: - Name: Bandwidth Change - Type: Double - Value: Z

2.4.2 Policy Framework

The traditional Operations Support Systems (OSS) are evolving towards more flexible, agile and service-oriented management platforms, also known as Service Operations. The new Service Operations are catalogue-driven, data-centric and cognitive-driven. Additionally, when dealing with dynamic and programmable 5G network environments, it is required to guide system decisions to achieve desired outcomes in a highly configurable way. This can be achieved through policies, which can be seen as high-level directives that convey what the software components should do under certain conditions, complementing the Service Operations key features with policy-driven operational management. In fact, policies have always been present in software components - the key aspect is to extract these rules/policies from the software allowing service designers and run-time system operators to control the system behaviour.

The PF designed in SliceNet enables the project system architecture with policy-driven capabilities. Figure 10 illustrates the high-level view of the PF. SliceNet's PF design and software implementation is partially aligned with ONAP Policy Subsystem [8]. Since ONAP is already working on a policy-driven operational management architecture and therefore developing a Policy Subsystem, it was decided to experiment and adopt some of their software components. Other PF components that are not under development in the ONAP Policy Subsystem are currently under development in SliceNet. Further details about which policy components are being reused from ONAP Policy Subsystem and which ones are being developed in SliceNet will be given in the following paragraphs.

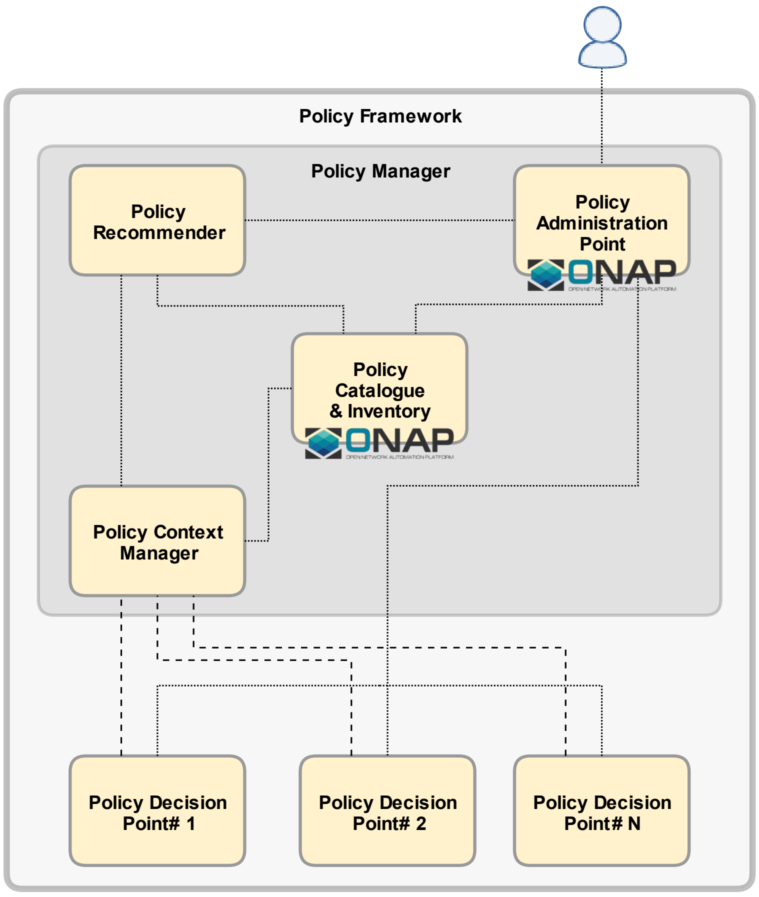


Figure 10 SliceNet Policy Framework architecture

In detail, the PF enables the service designer and/or system operators to **manage the entire policies lifecycle**, that is, allowing the policies Creation/Configuration, Read, Update and Delete (CRUD) operations. Policies are stored in the **Policy Catalogue & Inventory (PCI)** logical component.

The second key feature of the PF is the **policies administration** capability. That is, after policies are on-boarded to the PCI, the system operator should be able to activate the policies deployment on the SliceNet architecture components that should run and execute them - in policy-related terminology, these components are known as Policy Decision Points (PDPs). The policy deployment/distribution capability is delivered by the **Policy Administration Point (PAP)** component.

The third feature of the PF is to **run the deployed policies** at the **Policy Decision Points (PDPs)**. This happens during runtime and will dictate the system behaviour as indicated in the policy parameters - for example, Event-Condition-Action (ECA) policies that implement closed management/control loops in the architecture (e.g. imminent faults mitigation).

Another key capability of the PF is the **policies monitoring** feature. Whenever a specific policy is distributed by the PAP to be executed at a specific PDP, it should be collected information about the policy application - e.g. number of times it was executed and when it was executed. This capability is delivered by the **Policy Context Manager (PCM)** logical component.

Finally, the fifth key feature of the PF, and definitely one of the most challenging, is the capability to **analyse and evaluate the effectiveness and impact of the applied policies** in the system architecture components and, if necessary, **recommend updates to the existing policies** to improve their efficiency in the overall system behaviour. This capability is delivered by the **Policy Recommender (PR)** logical component. Such components may be enriched by insights provided by the analytics components in the overall Cognition Plane, providing better ways to recommend a policy or some of its parameters. Both PCI and PAP components are inherited from ONAP Policy Subsystem, whereas the PDPs, PCM and PR are developed within SliceNet.

Since the PCI, PAP, PCM and PR logical components are responsible for the policies management, they are grouped in the **Policy Manager** logical component. The Policy Manager is illustrated in Figure 10. On the other hand, the policy execution components (PDPs), also illustrated in Figure 10, are entities that can exist in different architectural locations - for example, PDPs at the virtual infrastructure management (e.g. **Virtual Infrastructure Manager - VIM as PDP**) level, as well as at the network control level (e.g. **5G Policy Control Function (PCF) as PDP**). In this regard, the PF is designed to work at different levels and roles (e.g. NSP and DSP), providing policies for different functional components. With respect to the SliceNet architecture, at the NSP level, a PDP is required for managing intra and inter slice aspects (e.g. **FCAPS Engine as PDP**). As they are different and independent business entities, the Policy Manager entity and PDPs must be instantiated per each SliceNet business role, with the responsibility to manage and execute policies in each business domain, respectively. This is illustrated in Figure 11 for the NSP scenario.

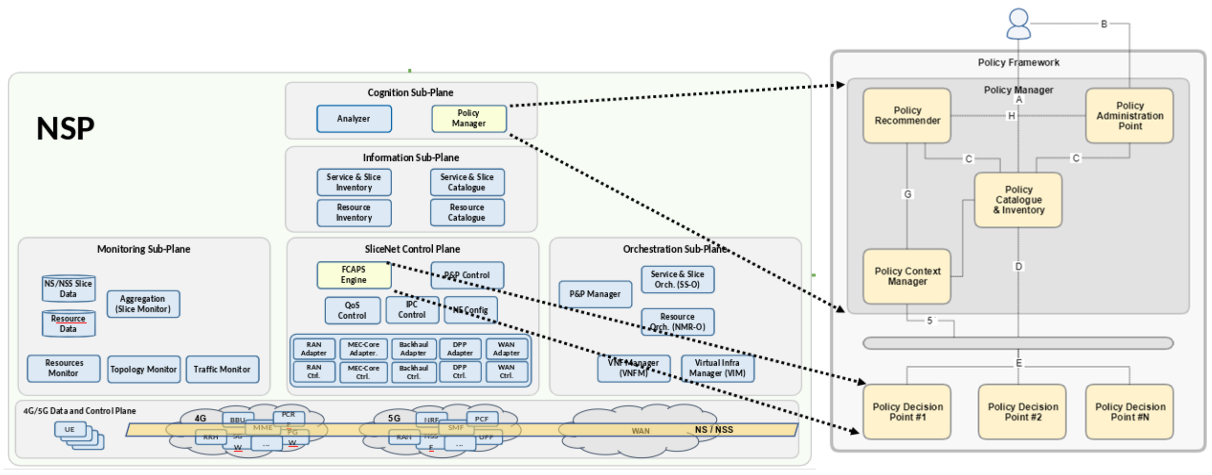


Figure 11 SliceNet PF architecture (instantiated at NSP)

On the other hand, Figure 12 illustrates the DSP scenario. In this case, a PDP is required at DSP level for managing E2E NSes (e.g. **QoE Optimizer as PDP**). A Policy Manager entity is also instantiated at the DSP level to manage the domain-level policies. It is precisely this combination of Policy Manager and PDPs at DSP level, for which the QoE Optimizer is central, which brings to fruition the SliceNet vertically-informed actuation framework.

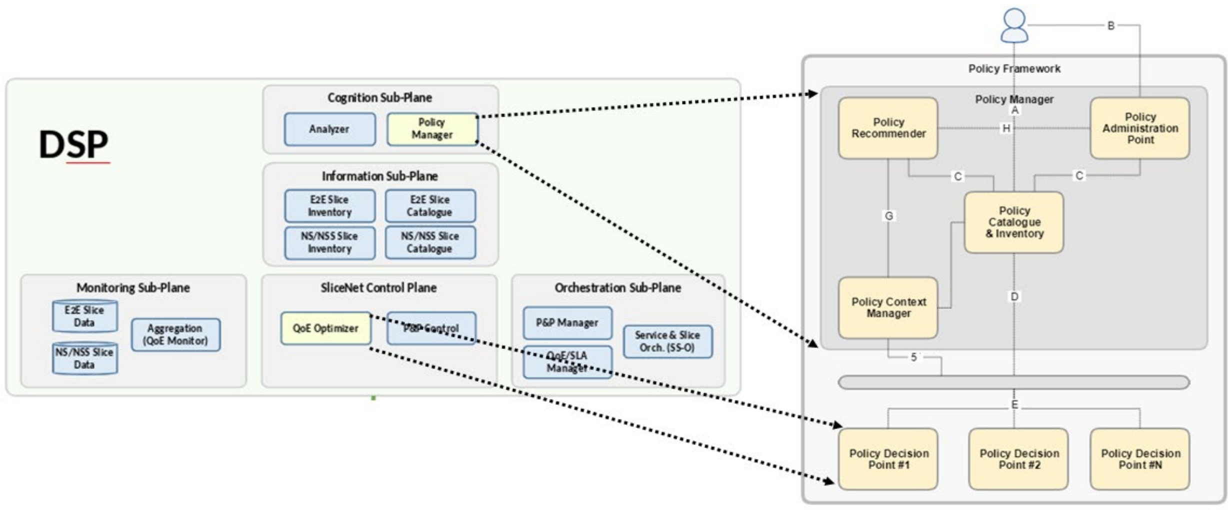


Figure 12 SliceNet PF architecture (instantiated at DSP)

2.4.2.1 Policy administration operations

Given the aforementioned PF architecture, this section elaborates on the main operations regarding policy administration which is enabled through the interaction with the several PF internal components and PDPs, namely:

1. Policy Creation
2. Policy Deployment
3. Policy Recommendation

The policy creation operation, illustrated in Figure 13, represents the on-boarding of new policies to the SliceNet PF architecture. The system operator designs the policies and uses the PAP APIs to deliver them to the system (step 1). The PAP stores the policies on the PCI (step 2). This operation can also represent the policies' CRUD operations.

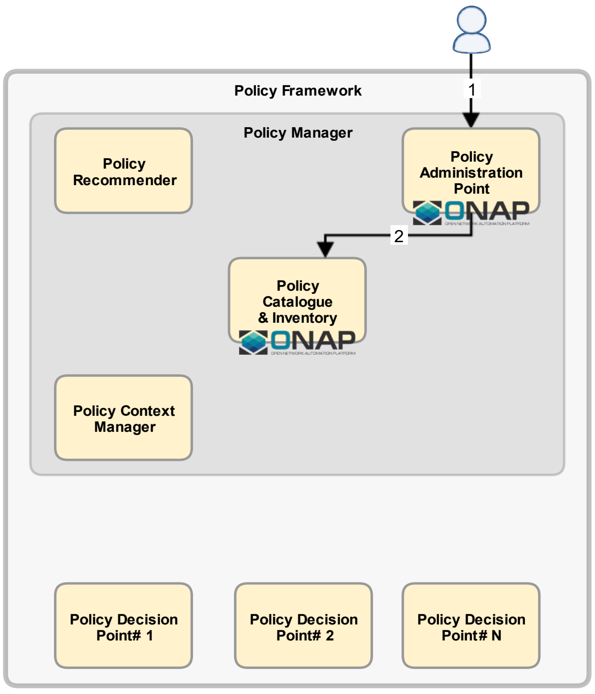


Figure 13 Policy creation/configuration operation

The policy deployment operation, illustrated in Figure 14, describes the deployment of previously on-boarded policies to the architecture's PDPs. Summarizing, the system operator invokes operations exposed by the PAP APIs to instruct the policy deployment (step 1). The PAP retrieves the policy to be deployed from the PCI (step 2) and distributes it to the selected PDP(s) (step 3). After the policies are deployed and running, the PCM collects information from the PDPs about the policies execution (step 4).

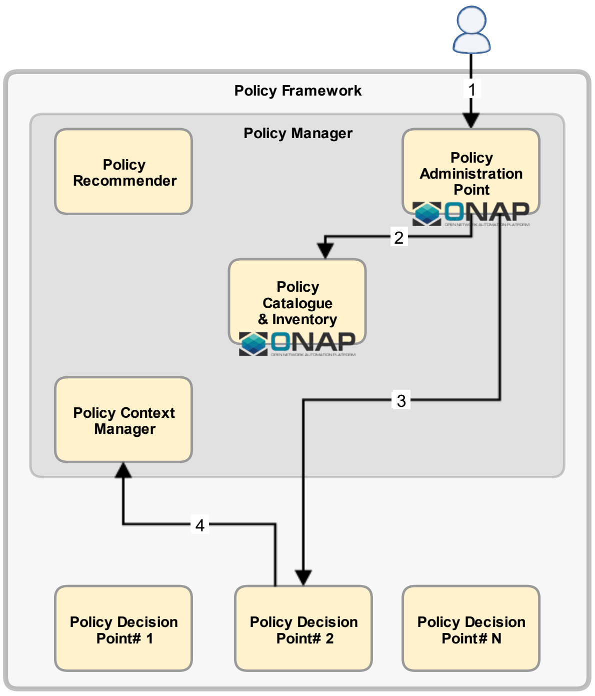


Figure 14 Policy deployment operation

Lastly, the third operation (illustrated in Figure 15) is the policy recommendation. This operation is related with the continuous improvement of the created policies in the system. In summary, after policies are deployed and running in the PDPs, the PR will be continuously receiving performance metrics (step 1) from the analysis part of the Cognition Plane. Additionally, the PR will also collect information about the running policies from the PCM (step 2). With the information collected in steps (1) and (2), the PR will use ML techniques to learn which are the most appropriate policy configurations and, if necessary, update the existing policies to improve their impact on the network (measured through the KPIs). Finally, after a policy update is ready to be recommended, the PR notifies the system operator (step 3). The latter will decide if he agrees with the proposed policy recommendation and, if yes, will update it on the PCI and request its deployment on the PDPs.

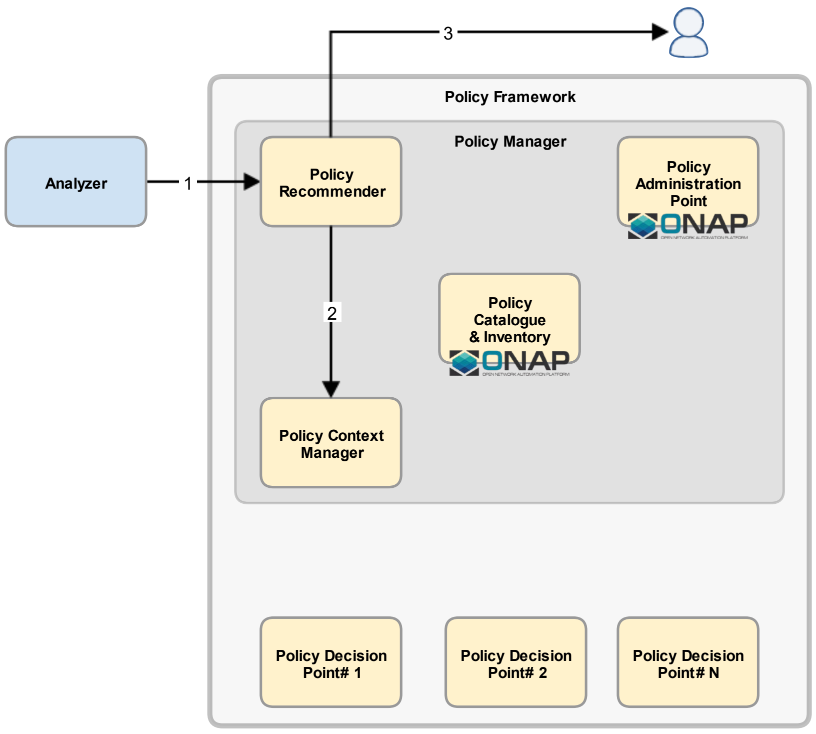


Figure 15 Policy recommendation operation

2.4.3 QoE Optimizer

The QoE Optimizer is responsible for triggering the desired actions (Execution from the MAPE-K loop) within the actuation framework. The actions are meant to maintain the quality of a particular E2E NS deployed on the underlying infrastructure, which may encompass several NSPs/segments/domains. As such, the QoE Optimizer is designed as a module that will be instantiated per E2E NS. A QoE Optimizer instance will have a specific actuation scope tailored to its NS, at the DSP level, since it is necessary to gain visibility of all elements/NSSes that intervene and may affect the quality of the delivered NS.

To achieve this goal, the QoE Optimizer follows a simple approach. On the one hand, it **listens** to monitoring data that carries information relevant to the quality of the NS under its responsibility. On the other hand, **actions** are directed to different parts of the SliceNet ecosystem to enable **(re-)configurations** at the underlying NS. These actions may be applied at concrete parts (thus, enforced to specific NSPs) within the E2E NS or affecting the totality of the slice as a whole. These two levels of actuation are both contemplated at the QoE Optimizer, for which the actual **scope** is determined given the received monitoring data and the **rules** to be applied. As such, it covers the several operations exposed by the main enforcement points (CP and orchestrator) and their abstracted functions (e.g. QoS control).

The QoE Optimizer behaviour is rule-based, with rules being the policies disseminated by the PF. The aforementioned ECA model is applied in this context. For each of the monitoring **events** that the concrete instance of QoE Optimizer is subscribed to, the policies define a specific **condition** that should be checked periodically. When a condition is violated, a corrective **action** is applied. This action is specified by the policy and the QoE Optimizer is responsible for coordinating the actuation workflow that fulfils the desired (re-)configuration operation. As such, the QoE Optimizer assumes the role of one of the PDPs in the context of the PF, giving the necessary means to decide and apply the actions stated by the distributed policies.

Figure 16 depicts the functional architecture of the QoE Optimizer and its main logical components. The specific software implementation of the QoE Optimizer will be discussed later on in Section 4.2.4 in the context of the QoS Modification actuator.

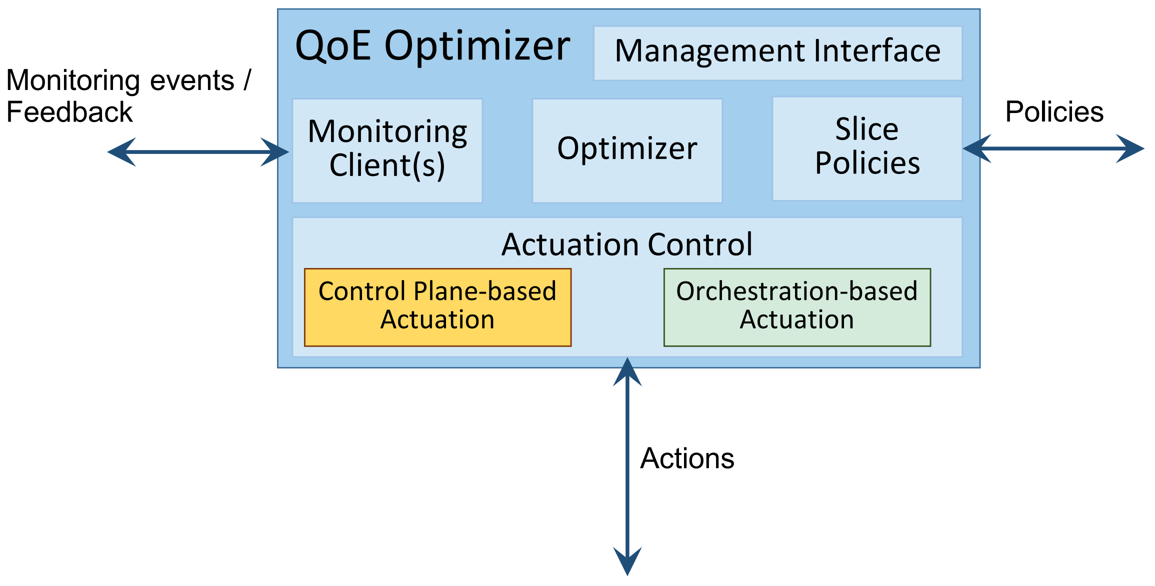


Figure 16 QoE Optimizer logical architecture

The **Optimizer** functional block is responsible for determining when and how conditions at the policies are being violated. It also has some limited intelligence to determine the best way to enforce the policy actions (e.g. the values of the parameters of the actions).

The **Monitoring Clients** are responsible for fetching the external information sources, i.e. sensors of some type or feedback from the vertical, for consumption by the QoE optimizer as a whole.

The **Slice Policies** are the logical representation of the policies being disseminated from the PAP at the PF. The Slice Policies dictate the actions that are to be triggered in the context of a managed NS.

The **Actuation Control** is responsible for triggering the operations (including its parameters) at their enforcement points (**Control Plane-based Actuations** and **Orchestration-based Actuations**).

The **Management Interface** enables the deployment, activation and termination of the QoE Optimizer instance.

2.4.4 Short/cross-entity actuation loop

Although the core of the actuation loops take place at the DSP level, there are some scenarios in which performance related problems that affect the quality of the NS, thus, the E2E perceived QoE, may be resolved at the NSP level. In this regard, the approach followed at the DSP is also applied for the several NSPs involved. As such, a MAPE-K loop is also being exercised at NSP systems, in which a monitoring stack gathers performance information of the different NSSes and resources being deployed. Then, an Enforcer engine is responsible for enforcing actions at the infrastructure (e.g. via the CP). This Enforcer engine also acts as a PDP in the context of the PF, which, as described previously, may cover both DSP level, for E2E wide rules, and NSP level, for more NSS-oriented/local rules. If the detected anomaly/underperforming situation can be resolved at the NSP level, the Enforcer will take charge of the necessary actions to correct the situation. In this case, the DSP QoE Optimizer would not be conscious of the anomalous situations, since corrective measures have already been taken. On the other hand, if the short actuation loop present at a given NSP is not able to overcome the situation, two options arise: (1) The underperformance situation is reflected in the collected and aggregated monitoring information at the DSP level, for which the DSP level QoE Optimizer will react to. (2) Otherwise, the DSP level QoE Optimizer is notified of the specific anomaly and will take the desired corrective action. Note: the notification mechanism may be enabled with a plugin at the P&P controller at NSP level.

Figure 17 depicts the logical architecture of the elements involved in the cross-entity actuation loop, ranging from the MAPE-K loop elements (e.g. Monitoring, QoE Optimizer, PF, ...) to elements that facilitate the separation/communication between roles/entities (e.g. One Stop API (OSA), P&P). In this regard, note that separation across DSP and NSP roles may be achieved thanks to an OSA entity or through capabilities exposed by a P&P instance at the NSP, for which the figure depicts an example. Specific interfaces between roles may also be implemented to achieve this purpose. The concrete design and implementation of operations, as well as functional elements is an ongoing work, for which the current logical and functional design is being presented, with some examples to highlight the possible separation among roles. It will be further addressed in future iterations of the actuation framework in WP5, as well as in WP6 and WP7, due to their close interaction with the actuation framework as a whole.

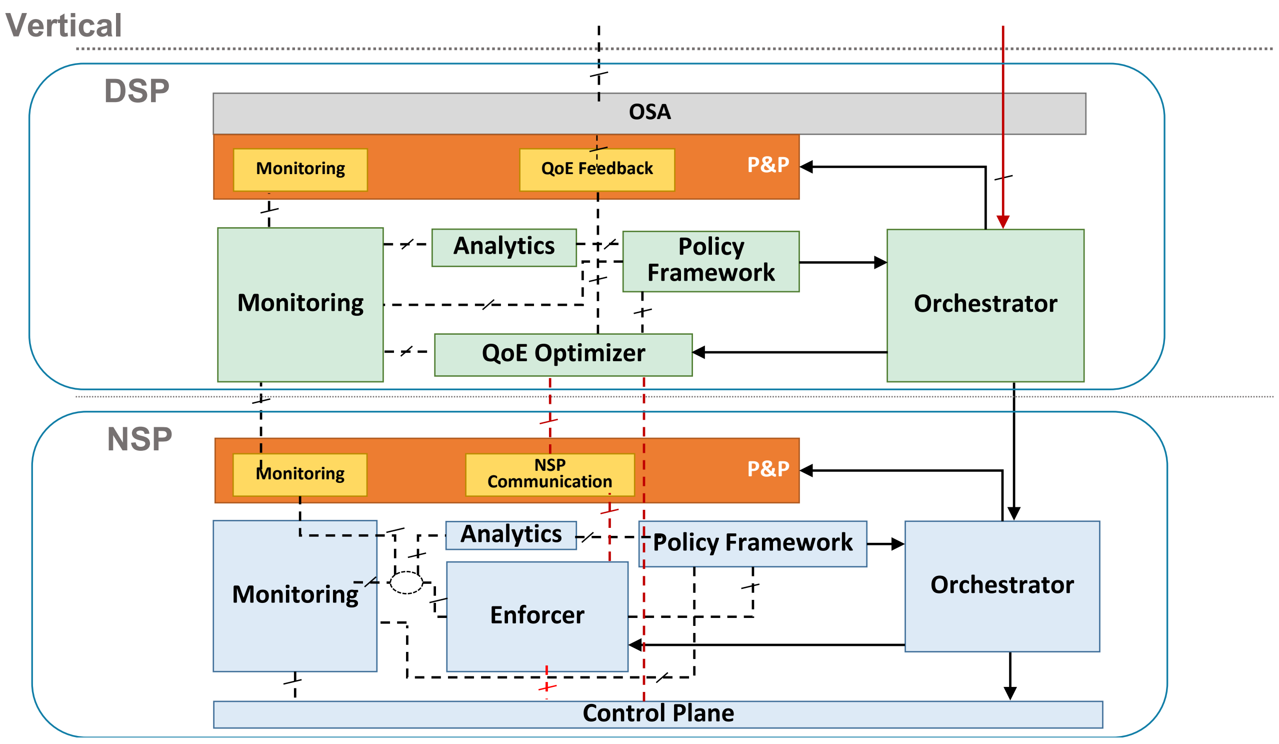


Figure 17 Logical design and architecture of the short/cross-entity actuation loop

2.5 Cognition workflows

To summarize the capabilities and phases that have been explained during this section, herein are described two high-level scenarios that exercise the cognition-related workflows, more precisely the creation of the ML models by the Cognition Plane. The first scenario, illustrated in Figure 18, describes the cognition procedures at the NSP level - for example, predicting a RAN slice imminent failure. In summary, the steps can be grouped into three phases:

1. **Network (Raw) Data Monitoring**: during this phase, network and infrastructure information (counters) is collected from the network and infrastructure sensors, respectively. This includes traffic/flows-related information. Push and/or pull models are supported and implemented through dedicated monitors and thereafter persisted at the SliceNet NSP data store.
2. **Network Slice Metrics/Indicators Calculation**: in this phase the collected and persisted data is pre-processed (cleaning, normalization, transformation, etc.) for further analytics. Additionally, raw counters are aggregated to produce metrics about the network's slicing performance.
3. **Network Data Collection & Learning**: finally, in this phase, data is fetched from the data store and further processed for the ML algorithms. The outputs of this phase are the predictive models to be deployed in the system architecture.

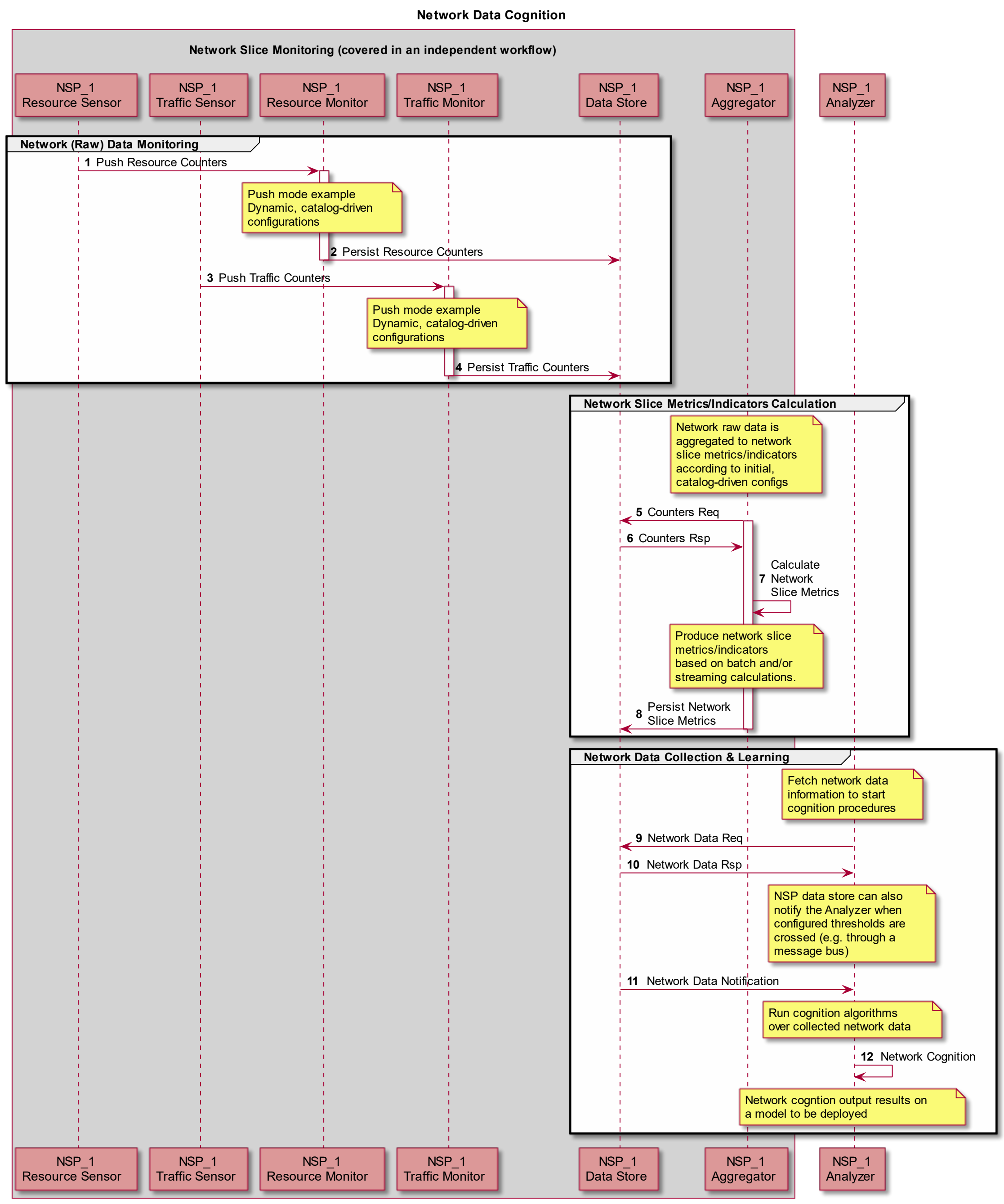

Network Data Cognition
Network Slice Monitoring (covered in an independent workflow)
NSP_1 Resource Sensor
NSP_1 Traffic Sensor
NSP_1 Resource Monitor
NSP_1 Traffic Monitor
NSP_1 Data Store
NSP_1 Aggregator
NSP_1 Analyzer
Network (Raw) Data Monitoring
1 Push Resource Counters
Push mode example Dynamic, catalog-driven configurations
2 Persist Resource Counters
3 Push Traffic Counters
Push mode example Dynamic, catalog-driven configurations
4 Persist Traffic Counters
Network Slice Metrics/Indicators Calculation
Network raw data is aggregated to network slice metrics/indicators according to initial, catalog-driven configs
5 Counters Req
6 Counters Rsp
7 Calculate Network Slice Metrics
Produce network slice metrics/indicators based on batch and/or streaming calculations.
8 Persist Network Slice Metrics
Network Data Collection & Learning
Fetch network data information to start cognition procedures
9 Network Data Req
10 Network Data Rsp
NSP data store can also notify the Analyzer when configured thresholds are crossed (e.g. through a message bus)
11 Network Data Notification
Run cognition algorithms over collected network data
12 Network Cognition
Network cogntion output results on a model to be deployed

Figure 18 NSS cognition workflow (at NSP)

The second scenario, illustrated in Figure 19, describes the cognition process at the DSP level - for example, predict the E2E NS latency degradation. The workflow phases in this scenario are similar to those described in Figure 18, except that the nature of the information flowing is different. In summary, the process is the following:

1. **Network Slices Data Monitoring**: during this phase, metrics are collected from each one of the NSPs NSSes that compose the E2E NS managed by the DSP. The retrieved information is persisted at the SliceNet DSP data store;
2. **E2E Slice Metrics/Indicators Calculation**: herein the collected information is further processed (cleaning, normalization, transformation, etc.) and aggregated to produce E2E NS metrics and evaluate its performance.

3. **E2E Slice Data Collected & Learning**: in this phase, the E2E NS data is collected from the data store and used in the ML algorithms. As a result, the E2E NS prediction models are created and ready for deployment.

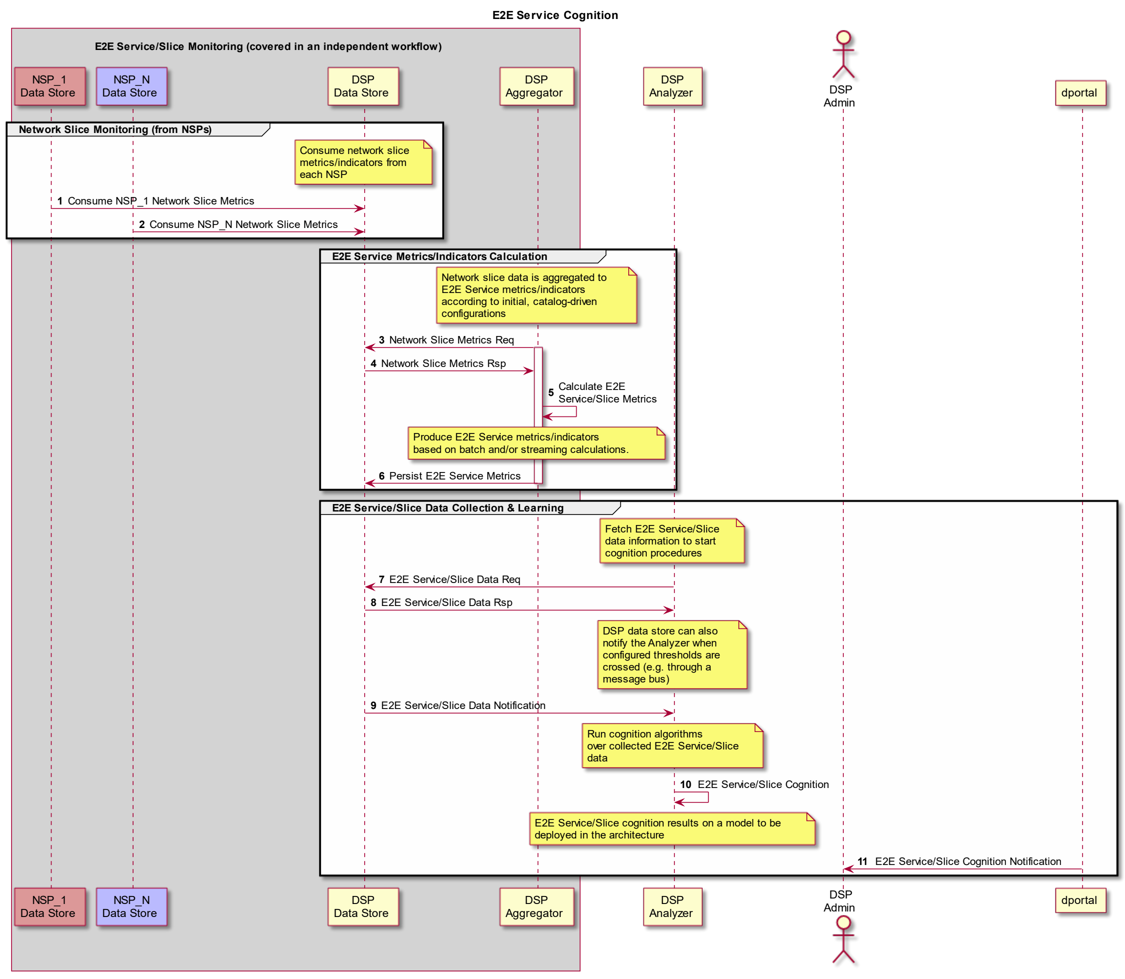

E2E Service Cognition
E2E Service/Slice Monitoring (covered in an independent workflow)
NSP_1 Data Store
NSP_N Data Store
DSP Data Store
DSP Aggregator
DSP Analyzer
DSP Admin
dportal
Network Slice Monitoring (from NSPs)
Consume network slice metrics/indicators from each NSP
1 Consume NSP_1 Network Slice Metrics
2 Consume NSP_N Network Slice Metrics
E2E Service Metrics/Indicators Calculation
Network slice data is aggregated to E2E Service metrics/indicators according to initial, catalog-driven configurations
3 Network Slice Metrics Req
4 Network Slice Metrics Rsp
5 Calculate E2E Service/Slice Metrics
Produce E2E Service metrics/indicators based on batch and/or streaming calculations.
6 Persist E2E Service Metrics
E2E Service/Slice Data Collection & Learning
Fetch E2E Service/Slice data information to start cognition procedures
7 E2E Service/Slice Data Req
8 E2E Service/Slice Data Rsp
DSP data store can also notify the Analyzer when configured thresholds are crossed (e.g. through a message bus)
9 E2E Service/Slice Data Notification
Run cognition algorithms over collected E2E Service/Slice data
10 E2E Service/Slice Cognition
E2E Service/Slice cognition results on a model to be deployed in the architecture
11 E2E Service/Slice Cognition Notification

Figure 19 E2E service/NS cognition workflow (at DSP)

3 Analytic workflows

3.1 Scope and preliminary implementation

The implemented analytic workflows demonstrate the logical SliceNet workflows responsible for predicting NS QoE KPIs; including, obtaining the required data, learning, processing, and providing the necessary information for actuation, to allow NS QoE management. As described above, these workflows enrich the raw data from multiple sources (including the vertical), adding insights and predictions to support Vertically-Informed Actuation (see Section 4 for actuation workflows).

Implementing ML analytics requires data. As most of the data source of SliceNet are not implemented at this point of the project; the analytic workflows can only provide PoC implementations. Following an agile approach, the analytic methods will be refined as data is being gathered and the vertical UCs mature. In their current state, the workflows rely on external data (when available) and simulated data (e.g., for modelling the vertical perceived QoE).

The implementations exercise several E2E logical workflows of SliceNet that interact with the Cognition Plane (see Table 2). We demonstrate monitoring and cognition at both NSS and E2E NS/service levels, providing the building blocks for the implementation of Vertically-Informed QoE Sensors. The focus at this stage is on the learning phase of the cognitive pipeline, validating the applicability of the analytic methods employed using external and simulated data.

Table 2 Summary of analytic prototype workflows

Name	**Short description**
Reliable RAN slicing using NSP alarm data	Demonstrate processing of external data sources to support slice reliability; optimizing resource selection during slice creation and predicting imminent failures
Noisy neighbour detection	Demonstrate the ability to provide a QoE Sensor that monitors slice-level metrics and applies cognitive methods to predict service level degradation and pinpoint its origin (application vs. provider)
QoE classification from QoS metrics	Demonstrate the usage of vertical feedback at the training stage to develop a model for predicting E2E QoE from measured QoS KPIs
RAN optimization	Demonstrate the application of cognitive method for managing the RAN effectively and optimizing its capacity to maintain high QoS for multiple slices and meet their desired service-based performance objectives
Anomaly detection	Demonstrate a QoE sensor that predicts anomalies in a Long-Term Evolution (LTE) RAN and a model that realizes network slicing with guaranteed network-layer QoS (latency)

3.2 Reliable RAN slicing using NSP alarm data

3.2.1 Demonstrated functionality

The goal of this PoC is to demonstrate the ability to ingest external, processed data into SliceNet's Data-Lake and use it for creating and maintaining an E2E NS with the agreed SLAs, with regards to reliability. This section details how MEO's external data source is leveraged to define a UC closer to a real-world scenario, where the network slicing provides a solution to deliver a continuous service to the vertical. This capability will be integrated into the SliceNet's SmartGrid vertical UC scenarios.

3.2.2 Description of data

SliceNet is fortunate to have a reasonable amount of network operations data upon which it can base its studies. The data was granted by MEO, a Portuguese nation-wide telco operator, which belongs to the Altice group. It spawns a complete 4-month period from May to September 2018, and it is directly exported from their network operations production databases. It contains an aggregation of all kinds of alarms generated by the network resources, with 31 million occurrences across over 8000 different locations. The origin of the dataset is the AlarmManager, a tool that receives and collects events from heterogeneous sources (Simple Network Management Protocol (SNMP), log parsers, signals, etc.), managing them through parsing and aggregation. This layer abstracts the different types of events (e.g. alarms) coming from network elements and keeps track of which faults are new, active or closed. In Figure 20, the red square represents the AlarmManager role.

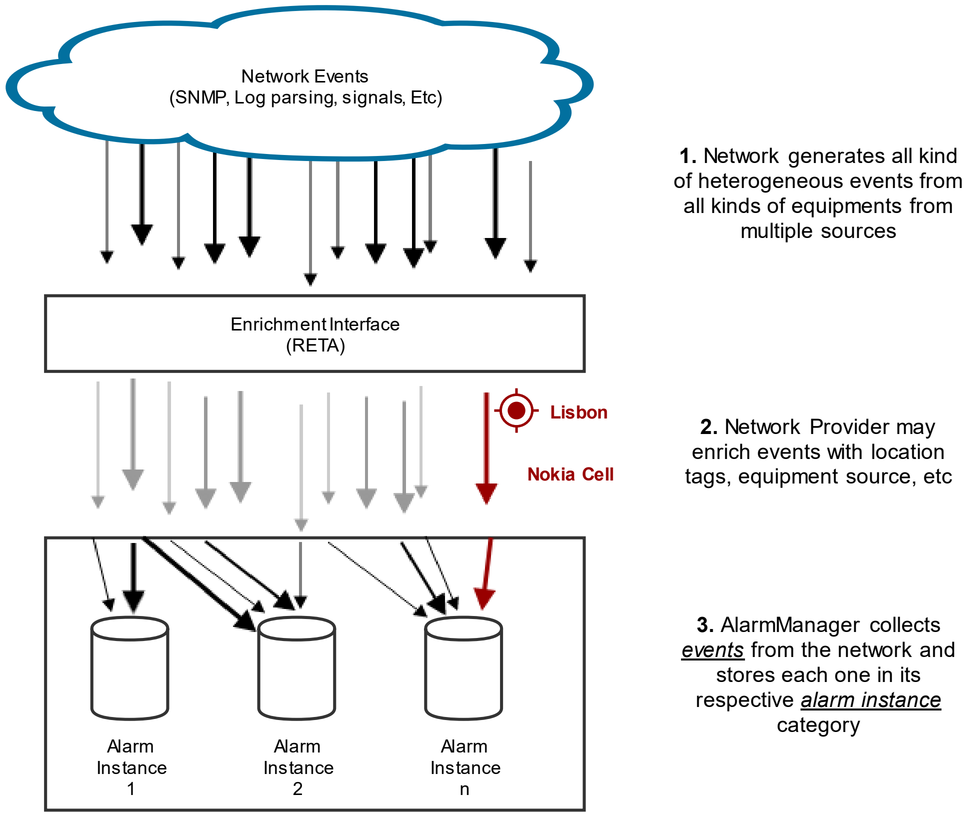

Network Events
(SNMP, Log parsing, signals, Etc)
Enrichment Interface
(RETA)
Lisbon
Nokia Cell
Alarm Instance 1
Alarm Instance 2
Alarm Instance n
1. Network generates all kind of heterogeneous events from all kinds of equipments from multiple sources
2. Network Provider may enrich events with location tags, equipment source, etc
3. AlarmManager collects events from the network and stores each one in its respective alarm instance category

Figure 20 AlarmManager role in the Altice MEO OSS

Each alarm instance has a location, a start and end time, the number of times the signal was emitted to the alarm manager, alarm severity and so forth. The dataset does not contain any type of correlation between the alarms and has different kinds of inconsistency and noise (which is common in these kinds of datasets). However, it is relatively well structured, allowing an easier transition for data exploration with data mining and ML. Further details of the attributes are available at [9].

The purpose of using such datasets is to represent as close as possible the signals that come from a real in-service network. We expect to extract two types of knowledge from it:

1. **Site reliability**: considering the history for each location, what is the most probable one to keep the expectations of the required SLA. The final output is a slice reliability metric, a score that impacts the decision when deploying a slice. This metric acknowledges the overall ability of a location being able to deliver the requested requirements.
2. **Imminent fault prediction**: considering the patterns on the network, given the actual running state, how close is a site (composition of network resources) to failure. This is extracted via past sequences, using ML as a tool to extract such hidden patterns. According to this analysis, the network can emit a signal, but the current situation is prone to failure. This signal is used by the NSaaS provider (NSP or DSP) in order to avoid a fault in the slice (NS or NSS).

The first objective is achieved by pure classical statistical analysis, which by definition is a required step before the second objective. For the second objective, the data needs to be in a format compatible with the algorithms, a process explained in Section 2.3.1, so that the models can learn the underlying patterns. Currently, despite the fact that datasets can very efficiently keep track of the current active alarms, it can barely distinguish which ones are just network noise from the ones that are real failures impacting the network (Figure 21).

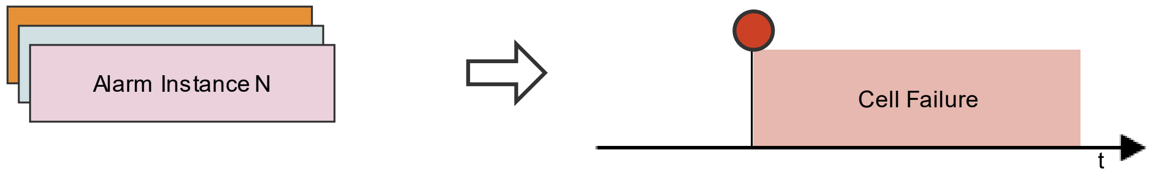


Figure 21 Relationship between alarms instances and the recognition of a failure

The intention is to discover the sequence or combinations of events associated with the failure. The improved solution of the second objective leads to the ability to deliver the requested SLA to the clients, hence transmitting a more precise situation awareness, as so:

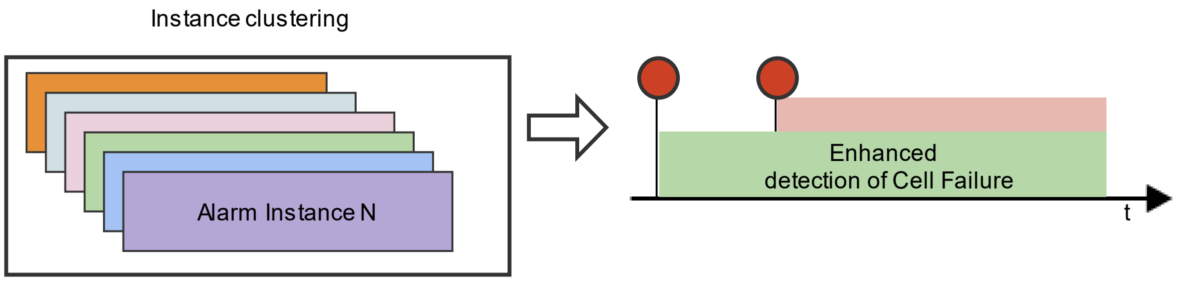


Figure 22 Recognizing what alarms lead to failure allows a more accurate and faster detection of a failure (or even preventively)

Consequently, closing the loop is done pre-emptively. Changes to the slice can be from simple resource reconfiguration through to the deployment of a completely new slice, and can be performed before having an impact on the service. The next section contains the current conclusions, made statistically over the data, which already fulfilled the first objective. Despite the fact the second objective is still under research and development, the analysis performed so far holds very promising expectations to the second objective.

3.2.3 Scenario

The Altice MEO dataset, as described in Section 3.2.2, passes through a process of discovering and understanding what type of information the dataset contains. Typically, data ingestion and data analysis work together in an incremental loop. For example, to know what type of data inconsistencies exist in the dataset we need to understand in detail the dataset's semantics and actual intent, and to do the first pass over the data we also must know the actual mechanics of how to ingest it. Prior to any model creation, there's a vicious cycle of exploration, measuring and characterization.

In the case of the MEO Alarm dataset, we started to learn what each field contains and represents. Per alarm instance (as explained above), the alarm metadata also contains the network resource domain (if it is from the RAN or the core), its technology (if it is 2G, 3G, 4G, etc.), and most importantly, the local site code. The last one allows us to aggregate the alarms (and its corresponding network elements) not only by a location, but also by an installation point. This means that all the devices under the same local code work together to fulfil a service. To build the NSP NSS reliability metrics, we need to know which locations are more prone to failure. In this perspective, we need to build a new metric, a failure-rate for each installation point. Since we are targeting RAN failures for the SmartGrid UC, we first need to identify how to get this knowledge from the dataset. To do so, we studied how sparse the alarms are by domain. Statistically speaking, we have segmented and enumerated the attributes, to map how many different kinds of samples exist for each one. The results are shown in Figure 23.

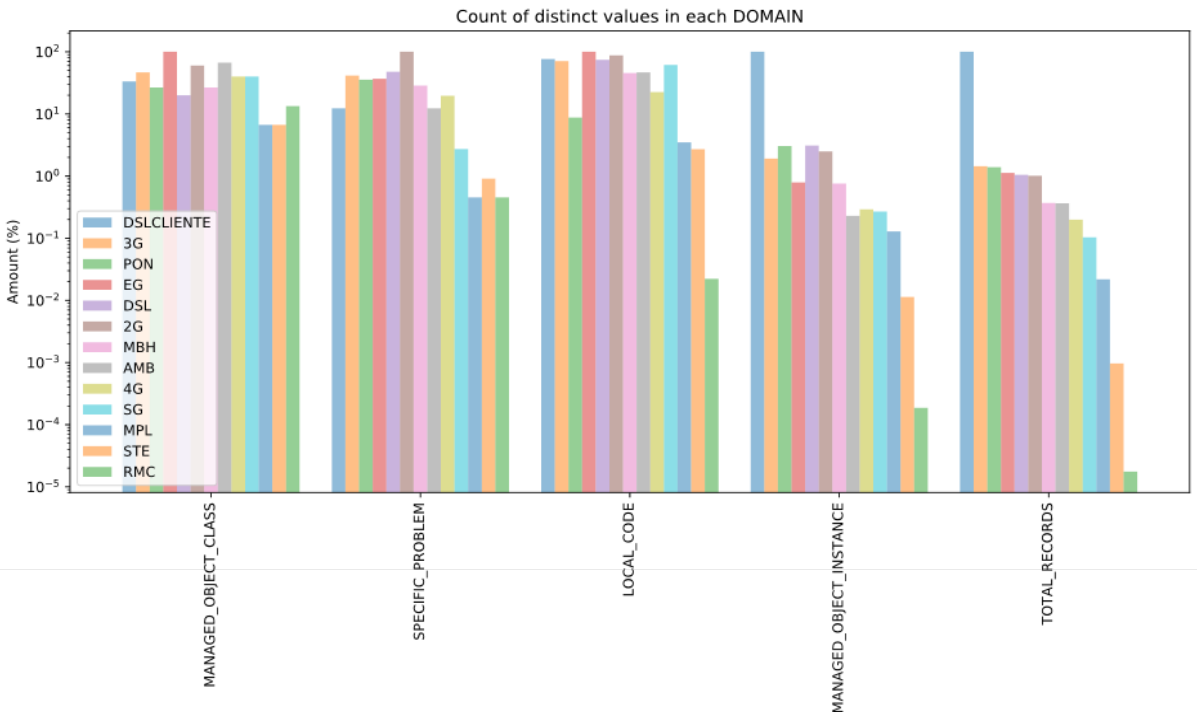


Figure 23 Dataset distribution over the different kind of alarm domains

Notice that the scale is logarithmic, meaning that most of the records are mostly from the DSLCLIENT (core) domain, meaning that the RAN alarms (2G, 3G, and 4G) - the number of entities per Local Code is under ten in average. The number of distinct problems is way less than the number of records, suggesting that there are a lot of observations per domain, which is a good indicator: we have many samples to each of them to learn from. As for the possible alarm types, as expected some of them are limited to a domain or technology. The dataset is largely composed of faults coming from the core and not the RAN, indicating that probably we can better evaluate an NSP reliability as a whole (all its core elements) and not just by considering the RAN reliability performance. A full analysis is available at [10]. We expect to discover correlations and causality between past and future alarms, and link the correlation to a prediction. This means that despite EFACEC IEDs in the SmartGrid UC solely connect via RAN, the core alarm data can also signal an NSP fault. Having real-world OSS data granted from Altice MEO operator is very useful for evaluating the scenario, where slice deployments can secure service availability, like so:

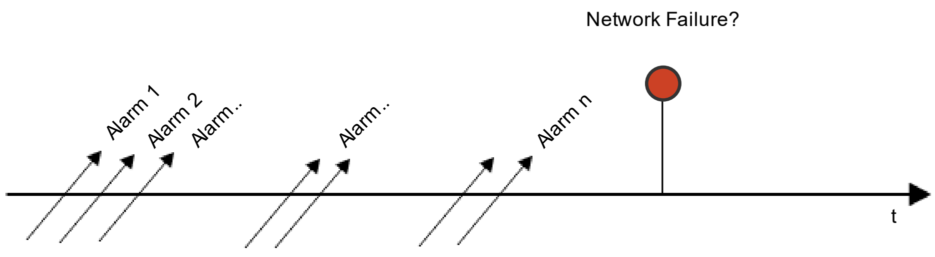


Figure 24 Relationship between the intended imminent fault prediction and the alarm data

Using ML, we believe that the pattern of the alarms preceding the network failure can be recognized, therefore the fault can be predicted, and a new slice can be provisioned to maintain the E2E service. Figure 25 is a high-level flow representation triggered by fault prediction.

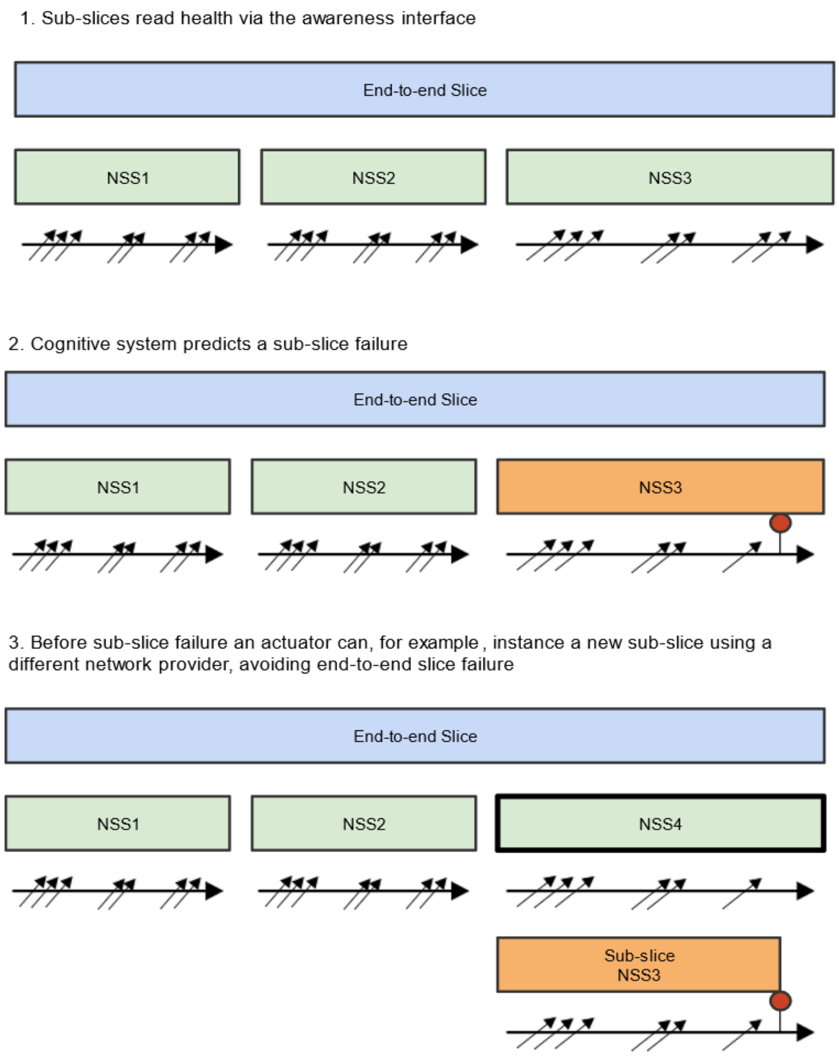

1. Sub-slices read health via the awareness interface
End-to-end Slice
NSS1
NSS2
NSS3
2. Cognitive system predicts a sub-slice failure
End-to-end Slice
NSS1
NSS2
NSS3
3. Before sub-slice failure an actuator can, for example, instance a new sub-slice using a different network provider, avoiding end-to-end slice failure
End-to-end Slice
NSS1
NSS2
NSS4
Sub-slice NSS3

Figure 25 Interrupted E2E service enabled through fault prediction and network slicing

Currently, the slice reliability score is retrieved directly from the statistical analysis performed so far. For online on-demand predictions, the cognition pipeline (discussed in Section 2.3.1) closes the loop by reading events directly from the NSP network resources, as the dataset implies. The presented PoC can be mapped onto the components of SliceNet's logical architecture as follows (Figure 26): (1) In the Monitoring sub-plane: data is gathered from the Altice MEO's dataset, an emulation of the Resources Monitor and Aggregation. These two modules combined would generate internally the same kind of data stream; (2) In the Cognition Plane: the data flow up to the Analyzer, where the algorithms according to the current events generated from the network resources asset the current slice condition.

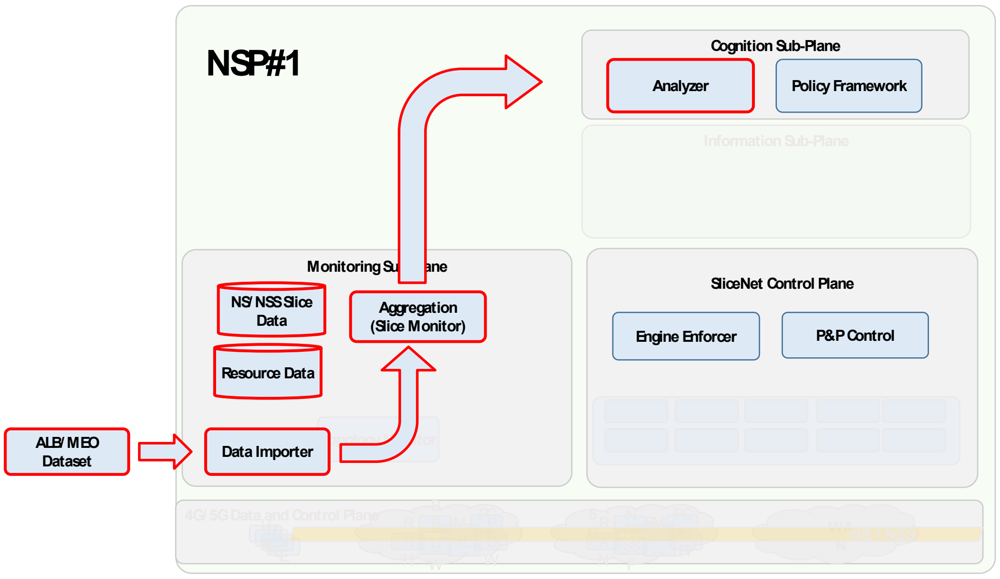


Figure 26 Altice MEO dataset inclusion into SliceNet's logical architecture

In other terms, the first half of the loop is supported by the dataset and ML pattern recognition. The latest developments so far, including the alarms analysis source code, are available at the shared cognition development environment.

3.3 Noisy neighbour detection

3.3.1 Demonstrated functionality

The goal of this PoC is to demonstrate the ability to apply cognitive methods to analyse QoS metrics and to predict the status of the VNF running in an NFV environment. Our objective is to create noise in virtualized environment in order to monitor the infrastructure and collect enough data, which can help us to detect and predict the VNFs/VMs suffering from noise using supervised ML algorithms. The problem of 'noisy neighbour' detection was described in the SliceNet deliverable D5.2 (Section 6.4.2) [3]. In this deliverable, we will describe the process to generate data and produce the 'noisy neighbour' detection model, the main components of the experiment setup, and foreseen interactions with the QoE Optimizer to enable actuations to remedy the noisy situation. Note that the 'noisy neighbour' approach is intended to be integrated within the SliceNet Smart City UC. Moreover, the actuation part, although explained, will be elaborated in future iterations of the Actuation Framework thus, as such, its implementation is not considered in Section 4.

The Smart City UC will implement the Intelligent Public Lighting system in the city Alba Iulia [11], in Romania. This use case will be deployed as a NS composed by a set of network elements (VNFs), components and specific applications. This NS ensures the connection of an enormous number of devices, simultaneously across the network. On the other hand, the network functions composing the NS will be deployed in virtualized environments.

In this context, the ‘noisy neighbour’ detection model will be used to detect if the perceived Quality of Information (QoI) (that, is the quality of information received for the machine-to-machine communication) is degraded due to the ‘noisy neighbour’ problem or not, since most elements of the network interconnecting the devices will be deployed in a virtualized environment.

3.3.2 Scenario

The ‘noisy neighbour’ model aims to detect the status of the supervised VNF by observing a set of QoS measurements and then applying ML algorithms to predict the VNF status. The process to generate the ‘noisy neighbour’ model includes the following steps:

1. **Measuring QoS**: in order to generate and collect the dataset, we set up an environment on the OpenStack platform where we measure QoS metrics of a specific VNF (e.g. a Voice over IP (VoIP) application) using a monitoring system (i.e. Prometheus [12]).
2. **Generating data of different experimentations:** we experiment a situation where the VNF is stressed by another VM located in some server in order to simulate the noisy case. Then we experiment the situation where the VNF is stressed by another VM hosted in another server to simulate the overload case. The normal case corresponds to the situation where the VNF is not stressed.
3. **Training Set:** QoS metrics collected from each experiment were labelled according to the fixed situation (e.g. noise, normal…). The training set was extended with new data by applying the oversampling technique to avoid having an unbalanced dataset.
4. **Learning and validation:** a classification ML algorithm was applied on a learning dataset and we validated the generated model using test data.

3.3.3 Architecture and description of components

To apply our approach, firstly we need enough data to achieve reliable predictions. To generate and collect the dataset, we set up an environment on real cloud infrastructure running OpenStack software [13]. For our testbed, we used two servers with three VMs. In one server, we deployed an open source VoIP VNF-based application in a single VM with one core and 2 GB of memory. Another VM acting as a noise generator was deployed in the same server as the VNF to stress and to generate noise to the VoIP VM. The noise generator VM was deployed occupying the rest of the CPU of the physical server. In the second server, a load generator [14] is deployed and configured to initiate calls to the VoIP application (see Figure 27). The Prometheus server collects metrics from the QoS sensor (node exporter) deployed in the VNF, and then stores them locally in a QoS metrics database. The observed data will then be used, after applying pre-processing techniques, as a training data for the training phase. Note that the pre-processing step used in our case consists in removing incorrect data formats, missing data and errors while capturing the data. The pre-processing includes also the oversampling to create synthetic observations of the minority class.

To evaluate the performance of the trained model, we use the following metrics: accuracy, classification error, sensitivity, specificity, false positive rate and precision. The generated model is then used for the run-time phase to predict the VNF status in real time. As depicted in Figure 27, the ‘noisy neighbour’ detection model retrieves periodically QoS metrics from the Prometheus server and then requests for correction actions (actuation), based on the predicted VNF status. In this regard, VNF scaling or migration are contemplated as potential actuations, for which the QoE Optimizer would serve as the central point of coordination, contacting the Orchestration Plane to perform the desired action.

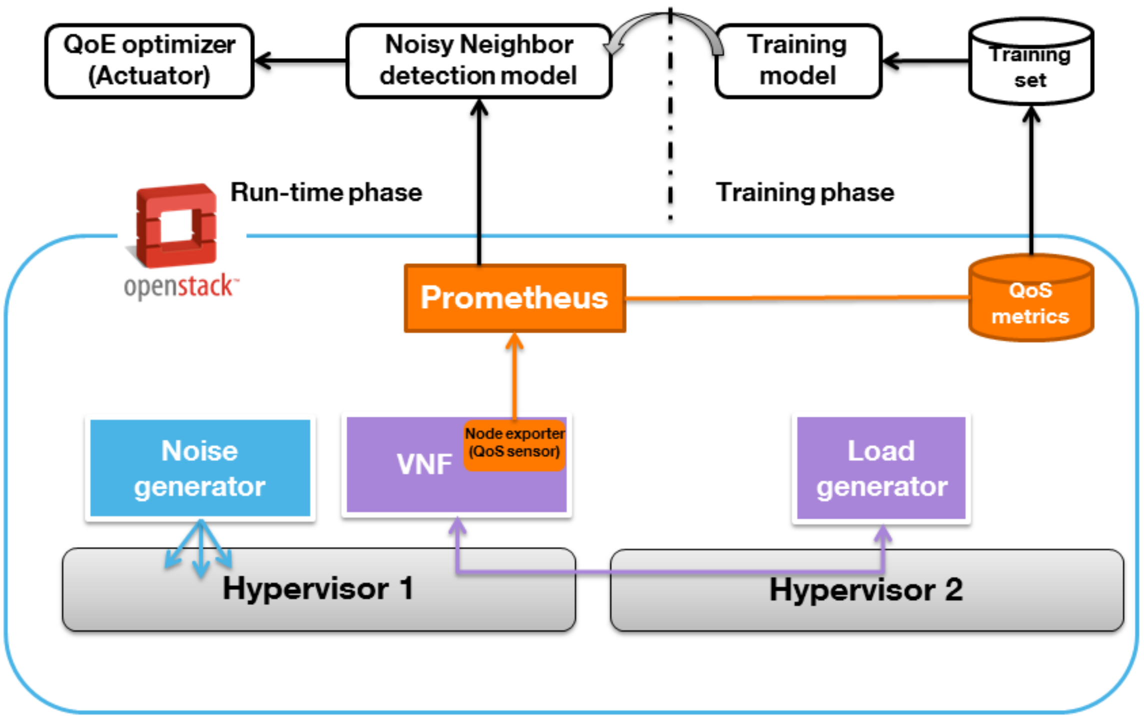


Figure 27 Experimental setup for the 'noisy neighbour' PoC

Figure 28 provides a potential workflow for this, highlighting the interactions between the QoS sensor, the 'noisy neighbour' model, the QoE Optimizer, and the orchestrator. The trained model for the 'noisy neighbour' detection makes regular requests to obtain a sample of the QoS metrics from the QoS sensor through the monitoring system. This data will be used as an input of the ML model to predict the status of the VNF. Three cases are possible:

- If the predicted status is **normal**, the 'noisy neighbour' model continues to request samples from the QoS sensor;
- If the predicted status is **noise**, the model requests, through the QoE optimizer, to migrate the VNF to another server.
- If the predicted status is **overload**, the 'noisy neighbour' model asks the QoE optimizer to perform the scale -up action and subsequently the actuator executes this action through the orchestrator.

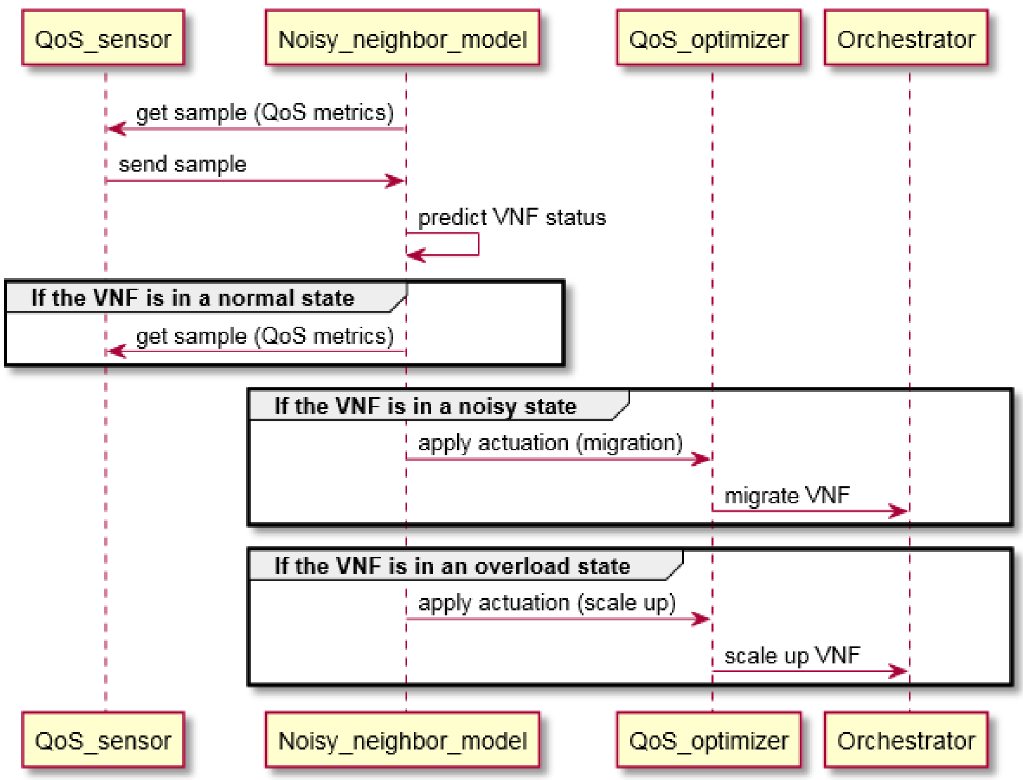


Figure 28 Actuation workflow for noisy neighbour detection

3.4 QoE classification from QoS metrics

3.4.1 Demonstrated functionality

The objective of this PoC is to demonstrate the ability to estimate QoE, as perceived by the vertical NS user, by applying cognitive methods to analyse network-level metrics collected by the slice provider. The assumption is that the slice (service) provider can measure various QoS metrics; however, does not have full information on the actual QoE the vertical is getting. Therefore, it must *estimate* the QoE from the measured QoS metrics. We employ ML to learn the QoS relationship to QoE through training with labelled examples. The learned data can be utilized at slice run-time to predict probable SLA violations and trigger corrective measures. Following the SliceNet data-driven operations approach, the model is deployed as part of the slice to generate an *Estimated-QoE* metric from monitored QoS KPIs. This metric is then consumed by slice control functions to trigger control and/or management actions required for proactively maintaining the service-level QoE, before any degradation affects the vertical.

This approach is intended to be integrated into the SliceNet eHealth UC and exercises several SliceNet workflows; including, E2E service cognition, monitoring, and QoE feedback. The PoC is focused on the QoE KPI of E2E latency.

3.4.2 Scenario

The scenario for this PoC includes the following steps:

1. **Measuring QoE –** in order to develop our approach, we created a controlled environment, where we can measure both network QoS parameters and application-level QoE, e.g. response time. We use a web service (WordPress) and measure the service level from the client perspective. The QoE is defined as a threshold on the average download time of the desired WordPress content. The application is created on two Kubernetes (K8s) container clouds deployed through the IBM ICP

service [15]; the client is on one cluster and the application on the other. The QoE was measured using JMeter [16].

2. **Generating different quality of experiences –** we generate "other" network traffic on both clusters using a stresser application that we have implemented. The stresser creates multiple containers and generates controlled traffic between them using iperf [17].
3. **Measuring QoS –** under our simulation model assumptions, the slice service provider can only measure local metrics within its slice. In our environment, we limited the QoS measurements to the K8s cluster that runs the WordPress service; namely, there are no metrics from the client cluster. The measurements are performed through SkyDive [18] flow capturing.
4. **Training Set –** QoS metrics from each experiment iteration are matched against the QoE measurement by time (wall-clock). For each iteration, we collect measurements only in the middle of the experiment window to remove the effect of end conditions (such as noise associated with creating the stresser containers). A validation set is created in a similar way.
5. **Learning –** We apply classification ML to infer the measured QoE class from the collected SkyDive metrics.
6. **Model Validation –** The model is validated against the validation set.

3.4.3 Architecture and description of components

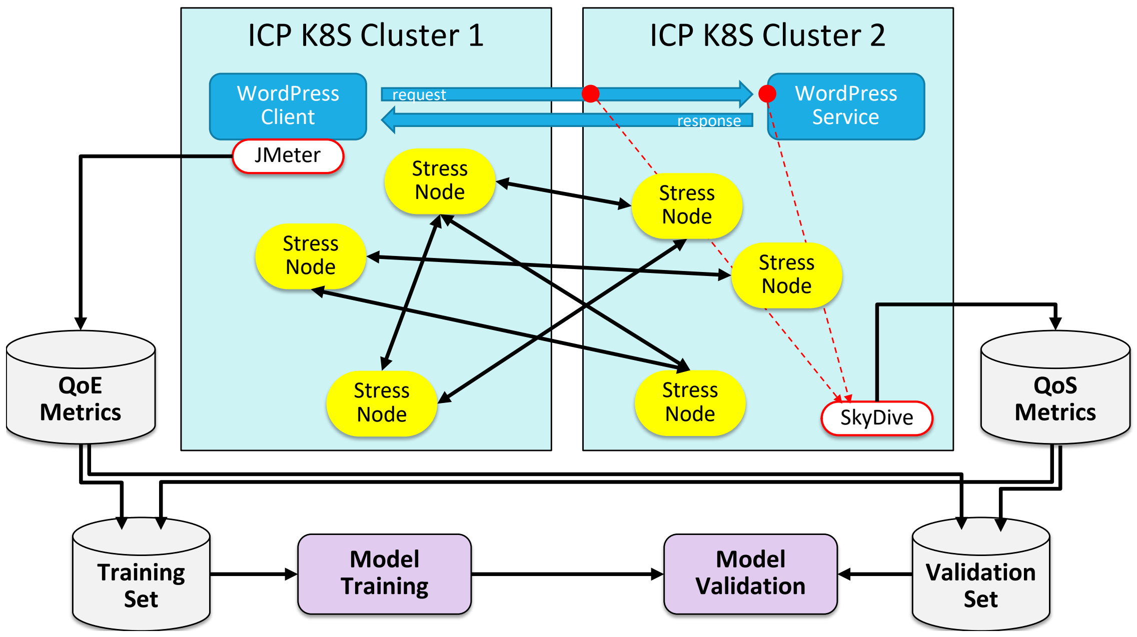
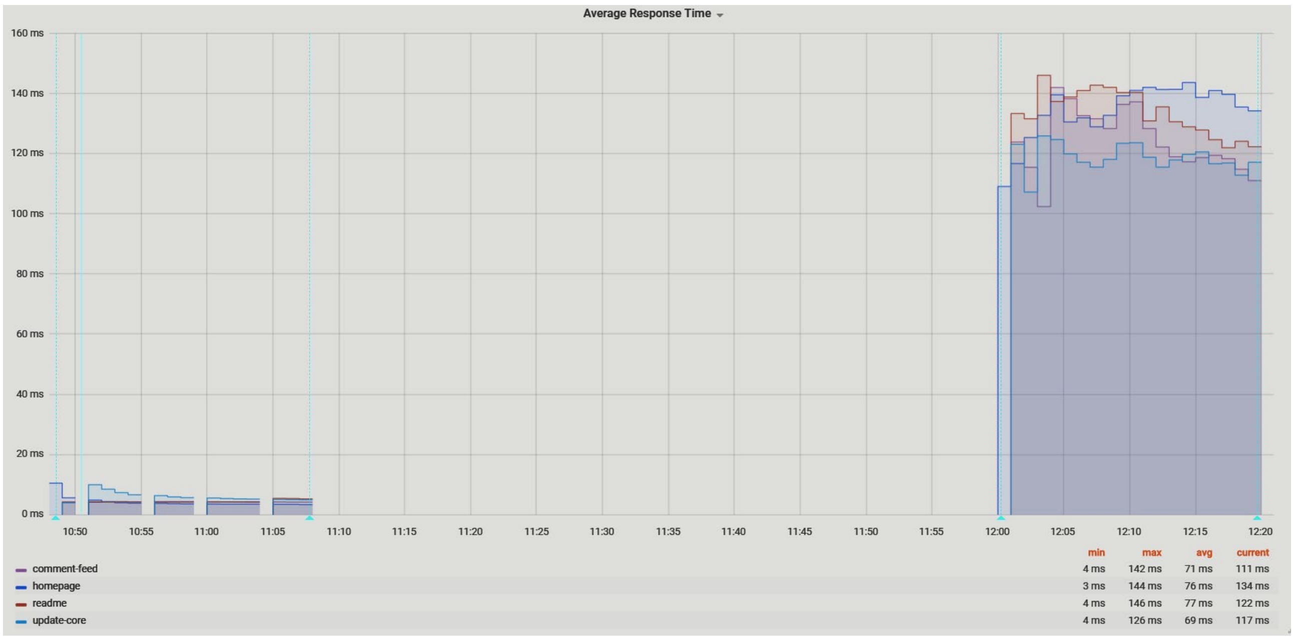


Figure 29 Experiment setup for QoS to QoE classification

The PoC experiment setup is described in Figure 29. K8s Cluster 2 corresponds to a managed slice, where the slice provider collects QoS metrics (through SkyDive) in order to manage the slice QoE. Stresser nodes generate controlled traffic to vary the E2E service behaviour; namely the QoE of the client. Actual service-level E2E QoE metrics are collected (by JMeter) and provided by the vertical (service user) for model training and validation. A ML classification process learns a QoE sensor model that estimates E2E QoE from measured QoS metrics. The model is then validated.

3.4.4 QoE derived from Client Application Response Time



	min	max	avg	current
comment-feed	4 ms	142 ms	71 ms	111 ms
homepage	3 ms	144 ms	76 ms	134 ms
readme	4 ms	146 ms	77 ms	122 ms
update-core	4 ms	126 ms	69 ms	117 ms

Figure 30 QoE derived from Response Time

The POC experimental set up derives the application level QoE metrics by monitoring the Wordpress client's response time with/without stress. Figure 30 illustrates that there is significant degradation to the download time of Wordpress pages when the background stress is applied.

3.5 RAN optimization

3.5.1 Demonstrated functionality

The performance of sliced services significantly depends on the RAN capabilities and functionalities. Managing the RAN effectively and optimizing its capacity to maintain high QoS for the slices requires cognitive control and management decisions, customized to the network slices for meeting their desired service-based performance objectives. The goal of the RAN optimization is to demonstrate how to provide such cognitive control and management over SliceNet's RAN to optimize its functionalities. To this end, we apply the DDCM approach, described in Section 2.3.4, over the SliceNet's RAN to actuate cognitive control and management to the RAN domain, while it can be extended to other domains too.

3.5.2 Architecture and description of components

To apply the cognitive plane containing the DDCM over the RAN, demonstrating how cognitive network control and management can be achieved in the RAN control domain, a paradigm architecture is depicted in Figure 31.

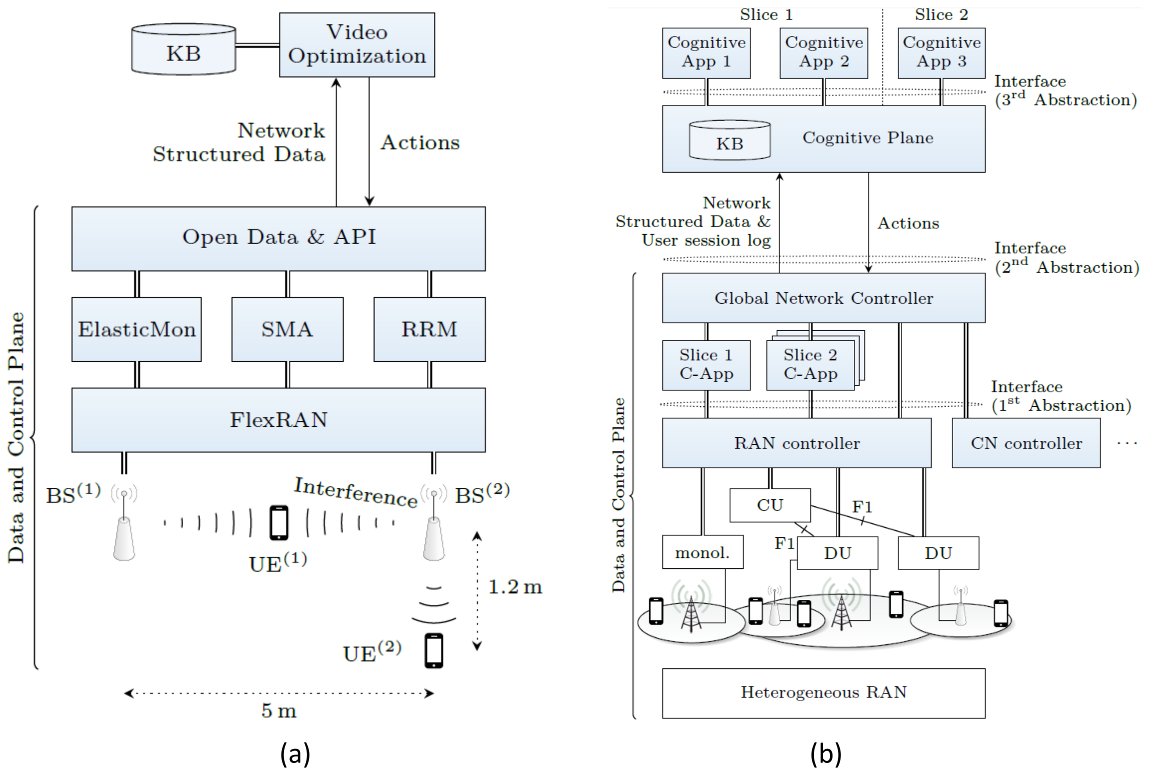


Figure 31 Integration of the DDCM with a RAN. (a) Architecture of cognitive RAN control and (b) Example of DDCM integration in SliceNet testbed

It can be seen that the NG-RAN [19] run-time operations and control decisions are instructed by the RAN controller, which provides a first platform-dependent abstraction for a base station. This abstraction is used by domain-specific C-Apps to perform slice-specific control logic (e.g. radio resource allocation) and expose the monitored or measured RAN status (e.g. link quality). In order to expose cross-domain C-Apps, a second-level of abstraction is exposed through a global network controller to unify the monitoring, control and management operations across different technologies and domains (e.g., RAN and CN). As a result, any single-domain C-Apps can open their data and API towards the Cognition Plane via the global network controller, based on those the cognitive C-Apps operate. Using the Cognition Plane as a third high-level of abstraction, cognitive C-Apps autonomously derive monitoring and control decision actions to meet slice-specific optimization goals, and actuate such actions in the network elements by means of local domain controllers, e.g., Software Defined Networking (SDN) controllers.

3.5.3 Scenario

We analyse the described cognitive RAN by considering a video streaming use-case on the top of the OpenAirInterface (OAI) [20] and Mosaic5G [21] platforms (Figure 31 (b)). We will show how the combination of knowledge of the radio resource and spectrum management information can be used in a video optimizer that enforces the objective to maximize a user's video quality while running in an eNB in low-power mode. The policy is to maintain the SLA, i.e., a downlink stream of at least 10 Mb/s. For this, we use information from the Spectrum Management Application (SMA) [22] (namely transmission power and operating frequency, and bandwidth) and Radio Resource Management (RRM) C-Apps (namely downlink throughput). We also use real-time link quality parameters which are monitored by using the ElasticMon (the monitoring framework described in Section 6.1). Especially the first parameters from the SMA can easily conflict with other domains, like administrative (the operating frequency of cells of another operator) or regulatory (maximum allowed transmission power). The throughput represents the data reflecting user satisfaction and on which network decisions are based, whereas the spectrum management parameters are control parameters to influence the network state. The aforementioned data and control parameters are fed into the knowledge base. Through it, the video optimizer is able to track the network state and predict new events. In particular, it will control the RAN through appropriate actions in order to satisfy the active users.

3.6 Anomaly detection

3.6.1 Demonstrated functionality

In Deliverable 5.2 (Section 6.4) [3], a ML model for anomaly forecasting has been presented. The model was proposed as an intelligent QoE sensor that allows prediction of future QoE using QoS metrics. The model has proven its efficiency on external network data in order to predict future anomalies in an LTE RAN. Apart from QoE prediction, the model also allows to guarantee a QoS-aware slicing, which attempts to realize network slicing with guaranteed network-layer QoS.

As a next step, the forecasting model will be applied to the SliceNet eHealth UC. Within the eHealth scenario, patient data are continuously collected and streamed once the emergency ambulance paramedics arrive at the incident scene. In order to enable more intelligent decision tools for the paramedics, the availability of real-time video streams to the emergency department is a requirement. These real-time streams may enable clinical professionals to remotely monitor the patient for conditions that are not easily sensed, such as skin pallor and patient demeanour. For this reason, the videos to be delivered need to have an ultra-reliable low latency communication across the network (30 ms to 100 ms latency E2E).

In order to maintain the low latency condition, the anomaly prediction model can be used. By forecasting the latency, the network maintainers may be alerted in advance and the issue could be solved before it occurs. For example, the QoE Optimizer may receive the outputs from the model and trigger corresponding actuations to remedy the situation. On the one hand, this will allow maintaining the perceived quality of the sliced services, while on the other hand, it can be used as an intelligent QoE sensor for the eHealth UC since, in Deliverable 5.2, the perceived latency is defined as a QoE indicator.

3.6.2 Scenario

The model aims to forecast the latency in the prediction horizon by observing a set of QoS measurements. The latency prediction scenario for the eHealth UC is summarized by the following steps:

1. Labelling the data according to a threshold on the latency attribute. Since in the requirements, the latency should be between 30 ms and 100 ms, the threshold should be fixed to 100 ms. Thus, latency that is bigger than 100 ms in the prediction horizon should generate an alert. The latency is considered normal otherwise. The prediction horizon depends on the data and the requirements for actuation.
2. Dealing with missing values by replacing each missing value by the mean of the corresponding QoS metric over all the dataset.
3. Applying a standardization on data in order to change ranges of values for the different metrics.
4. Over-sampling the dataset in order to have more balanced data.
5. Applying smoothing for each QoS metric in order to transform them to functional data.
6. Applying a principal component analysis for multivariate functional data for the resulted smoothed dataset in order to reduce the dimensionality of the data.
7. Classifying the training data composed of the principal components of each QoS metric by using a random forest algorithm.
8. Validating the model by using with two techniques: (1) validation over a test data and (2) validation with a cross-validation.

The implementation of the anomaly prediction model is achieved using R language [23].

3.6.3 Architecture and description of components

Figure 32 describes the architecture of the latency prediction model for the eHealth UC. The UE collects measurements about the network. These measurements serve as training data. After the labelling, the anomaly prediction model will learn the patterns of normal and of suspicious behaviours of the QoS metrics regarding the latency. A trained model is produced. This model is usable for the exploitation phase in order to forecast the latency for new measurements collected by the UE. Depending on the prediction, the QoE Optimizer, as a client of the prediction data, may solicit to the orchestrator or the CP corresponding re-configurations in order to correct the latency issue in advance, which allows avoiding service degradation and a bad QoE perceived by the medical multi-disciplinary team involved.

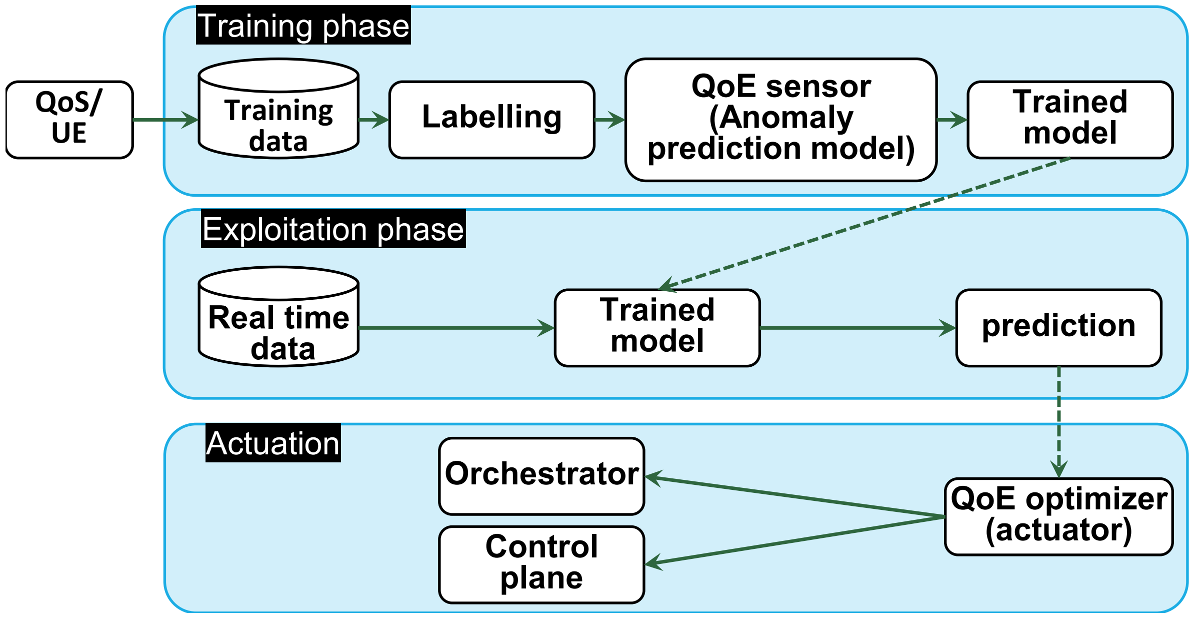


Figure 32 Architecture of latency prediction model for eHealth UC

Such an actuation will take place only if the model has predicted a high latency in the prediction horizon and the video to be transmitted in this prediction horizon is categorized as urgent content, such as telemedicine. Normal video streams, such as a regular video call, may be more tolerable to latency. Once the model predicts a high latency for an urgent video stream, three scenarios for actuation are possible as illustrated by the workflow presented in Figure 33. If the RAN is congested, some non-essential layers of the video may be dropped (for instance, re-configuring the video encoding VNF through the CP functions). Hence, by consuming fewer resources, the QoE can be maintained. However, if the RAN is not congested, the actuator may scale relevant VNFs in order to enhance their resources allocation. Note that these describe potential actuations that may be exercised thanks to the capabilities exposed by the QoE Optimizer module of the Actuation Framework (see Section 2.4.3). Hence, a high-level design and a workflow are presented.

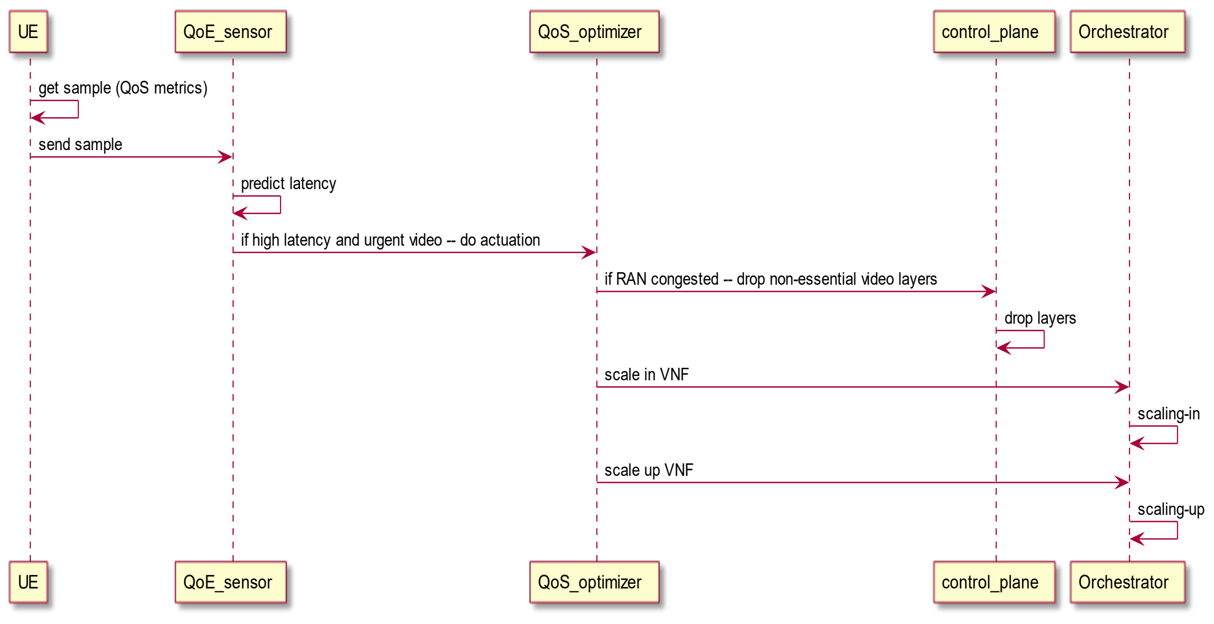


Figure 33 Actuation workflow for latency prediction for eHealth UC

4 Vertical-informed actuators workflows

4.1 Scope and Preliminary Implementation

As mentioned previously, the scope of the actuators within the overall SliceNet architecture is the maintenance of the QoE of deployed NSes. To achieve so, the actuators framework relies in information gathered from both the monitoring sub-system and feedback from the verticals. Then, after having analysed the gathered information, it triggers the necessary (re-)configurations of the provisioned NS to achieve the desired quality levels. In this regard, two main levels of actuation have been considered: actuations at the E2E level (E2E NS) and at the segment/NSP level (NSS). Thus, depending on the monitoring data gathered, the actuation framework may enforce (re-)configurations on specific NSSes within the E2E NS or at the provisioned NS as a whole.

To enforce the desired actions, the actuation framework follows a workflow-based approach, ruled by a policy system, for which the actions are achieved through the collaborative efforts of multiple components of the SliceNet architecture across multiple planes This approach is followed at both DSP and NSP levels, with WP5 efforts being focused in the DSP level (E2E perspective). As such, only in the case of actuation at traffic level, which relates to more NSP-oriented actuations, a specific actuator has been developed, which directly operates with the traffic flowing through Open Virtual Switch (OVS) instances at the provisioned data-path for the slices. Hence, while the generic actuator framework works in a workflow-based approach, since actions need to cover multiple aspects across the layered SliceNet architecture, for specific actions that require tight interactions with lower level elements of the slice, dedicated functions working as actuators are being deployed.

In this regard, for the first preliminary implementation of the actuation framework, three different actuators have been developed, exercising the different aspects of the actuation framework. While for the generic workflow-based approach, two different actuators have been considered: QoS modification and NSP sequence modification. For both of them, the QoE Optimizer and the PF are the main components that enable their operations. Meanwhile, to consider specific function-based actuators, an OVS-based traffic classification actuator has been designed and prototyped by extending the functions in the standard OVS. In such a case, the actuator enforces specific traffic control rules to the slices at the flow level thanks to the extended functions ingrained directly onto the OVS instances along the provisioned data-paths of the concerned slices. Table 3 provides a summary of the considered actuators, their scope and targeted scenario as well as the main involved components across the logical architecture.

Table 3 Summary of developed actuators

Name	**Short description**	**Approach**	**Scope**	**Main involved components**
QoS modification	Modify QoS parameters associated to the E2E NS or specific NSSes	Workflow -based	Slice; sub-slice	QoE optimizer; Policy Framework; QoE plugin; QoS Control CP function
NSP Sequence Modification	Modify the sequence of NSPs (thus, the NSSes) in which E2E NS is being supported over	Workflow -based	Slice	QoE optimizer; Policy Framework; QoE plugin; Service/Slice Orchestrator
OVS-based traffic classification	Control the user traffic on the data plane, based on extensions to OVS to enable the handling of 5G and multi-tenancy traffic	Function -based	Flow; (sub-)slice (based on aggregated flows)	QoS Control CP function

The following subsections will provide more details about the specific workflows and functionalities exercised through these actuators, specifying also their architecture and current preliminary implementation and how they relate to the overall SliceNet architecture and UCs.

4.2 QoS Modification

4.2.1 Demonstrated functionality

One of the main contributors to unsatisfactory QoE levels is the quality of the underlying virtual/physical infrastructure supporting the NS, that is, the QoS of NSSes composing the E2E NS. For instance, an insufficient amount of bandwidth provisioned in one or multiple segments spanning across the E2E NS can degrade the perceived QoE. Under such circumstances, it is essential to re-configure underlying QoS parameters to regain desired quality levels in the NS as a whole.

Given this goal, the QoS Modification actuator provides the necessary mechanisms to enforce QoS modifications in both NSSes residing at different NSPs (in selected groups or all of them as a whole) as well as the interconnectivity between the different NSSes. More specifically, the QoS Modification actuator exploits the capabilities offered by the QoS Control SliceNet CP function [24] at the NSP level, which allows for modifying the provisioned bandwidth and the configured priority of deployed slices in specific network segments (e.g. modify the bandwidth allocated to a RAN NSS).

Such functionality is triggered on-demand by the QoE Optimizer, which acts as the central piece for all workflow-based actuators. The actual trigger for the actuator relates to the conditions stated by the policies disseminated from the PF, which tie the monitoring information received by the QoE Optimizer (i.e. the event and condition) with the desired actuation (i.e. the action). When the stated condition is met, the QoE Optimizer request for a QoS modification operation in the affected NSPs/NSSes through the API exposed by the QoS Control at the SliceNet CP, prior retrieving the specific instances of the function tied to each NSS through the CP Service Registry (CPSR). Another contemplated source for the actuator trigger is the feedback provided by the vertical user, which then becomes another source of monitoring to be contrasted with the conditions stated by active policies. In this case, such feedback is gathered through a plugin developed in the context of the P&P controller [25]. It allows the vertical to report its perceived QoE and, as a consequence, trigger the actuator if any of the policies present in the QoE optimizer states as a condition a bad vertical's QoE and a QoS modification as action.

The QoS Modification actuator is intended to be integrated in the overall SliceNet framework as an enabler for the E2E service/NS optimization loop, exercised across the layered management and control architecture. Such loop will be then exploited for the vertical UCs that require modifications on the provisioned QoS at the E2E NS or NSS level as means to guarantee optimal QoE levels for the supported service. A preliminary PoC exercising such functionality has been presented in [26], also demonstrating the interaction with an Opens Source MANO (OSM) orchestration framework.

4.2.2 Scenario

The general scenario that exercises the workflow of the QoS Modification actuator includes the steps depicted in Figure 34. In summary, the steps are:

1. **Network slice monitoring**: the different counters and metrics that relate to the quality of the multiple NSSes deployed across the different NSPs are gathered from their corresponding monitoring systems and persisted at the DSP level data-store. Although not depicted, the feedback gathered from the vertical through the QoE plugin follows the same treatment, that is, it will be stored at the DSP data-store for its later consumption by other applications/modules.
2. **End-to-end slice/service metrics/indicators calculation**: following the Data-Lake approach presented in Section 2.2, the persisted monitoring data coming from the different NSPs is collected by an aggregator application (DSP aggregator), which utilizes the multiple single metrics

to calculate E2E service/NS quality metrics. Then, the elaborated E2E metrics are stored back at the Data-Lake (DSP data store). These metrics are the ones that will be later consumed by the QoE Optimizer for deciding if an actuation is needed or not.

3. **End-to-end QoS parameters modification**: this third macro-step is the one involving the QoE Optimizer as central point of the actuation workflow and thus, the QoS Modification action. Once the metrics related to the quality of the E2E NS are received/consumed, the QoE Optimizer checks the obtained metric against the events and conditions of currently configured policies for the NS. If it matches with one of the events, the condition is being violated (e.g. bandwidth lower than certain value) and the configure action for the policy is a modification on the QoS configured for the NS, the QoS Modification actuator is triggered. This requires first to retrieve the information of the NSPs which are affected/will need to enforce a QoS modification from the DSP Orchestration system. More specifically, given the gathered information, the QoE Optimizer requests for the contact point (i.e. the CPSR) and the identifier for the NSSes configured at the NSPs for which the conditions of the policies are not met. Then, it requests in a loop, for each NSP, the instance of the QoS Control CP function of the NSS. Once obtained, it requests for the actual modification of the desired QoS parameter (bandwidth or priority).

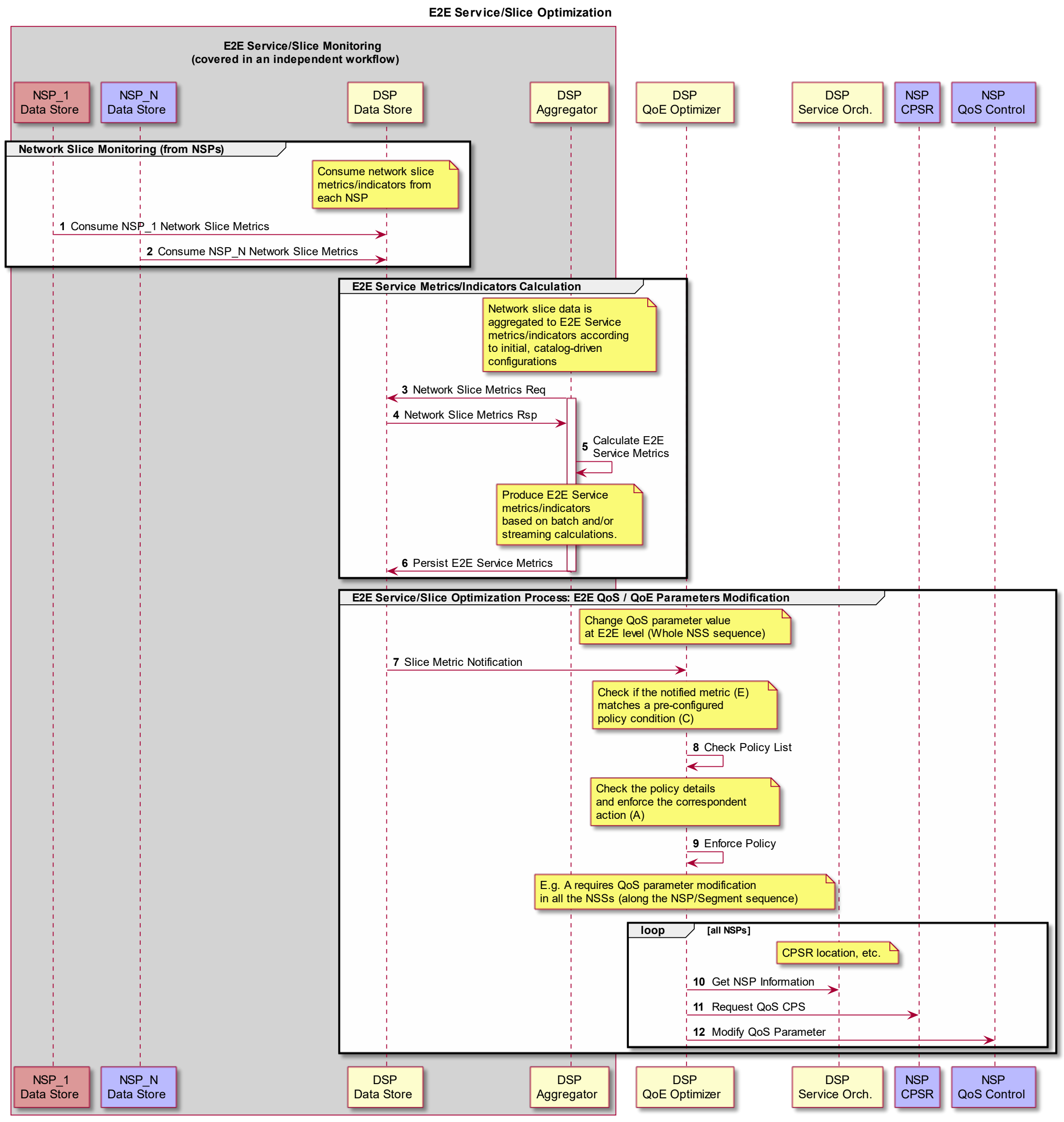

E2E Service/Slice Optimization
E2E Service/Slice Monitoring (covered in an independent workflow)
NSP_1 Data Store
NSP_N Data Store
DSP Data Store
DSP Aggregator
DSP QoE Optimizer
DSP Service Orch.
NSP CPSR
NSP QoS Control
Network Slice Monitoring (from NSPs)
Consume network slice metrics/indicators from each NSP
1 Consume NSP_1 Network Slice Metrics
2 Consume NSP_N Network Slice Metrics
E2E Service Metrics/Indicators Calculation
Network slice data is aggregated to E2E Service metrics/indicators according to initial, catalog-driven configurations
3 Network Slice Metrics Req
4 Network Slice Metrics Rsp
5 Calculate E2E Service Metrics
Produce E2E Service metrics/indicators based on batch and/or streaming calculations.
6 Persist E2E Service Metrics
E2E Service/Slice Optimization Process: E2E QoS / QoE Parameters Modification
Change QoS parameter value at E2E level (Whole NSS sequence)
7 Slice Metric Notification
Check if the notified metric (E) matches a pre-configured policy condition (C)
8 Check Policy List
Check the policy details and enforce the correspondent action (A)
9 Enforce Policy
E.g. A requires QoS parameter modification in all the NSSs (along the NSP/Segment sequence)
loop [all NSPs]
CPSR location, etc.
10 Get NSP Information
11 Request QoS CPS
12 Modify QoS Parameter

Figure 34 QoS Modification actuator general workflow

For the specific PoC being reported, the described workflow still applies, some simplifications to the whole actuation workflow are made. First, the employed source of monitoring to trigger the actuation is not gathered from any database/store but directly fetched through the QoE plugin. In this regard, the trigger for the actuation is the feedback reported by an external user (i.e. the vertical) through the capabilities exposed by said plugin. Note, however, that for the generic actuation workflow, such feedback could be an integrated part to the DSP Data Store, serving as input metric for the QoE Optimizer. A predefined policy stating that when the vertical's feedback reports a bad quality feedback (the feedback is reported in a String format), an increase of the bandwidth of the underlying configured NS (i.e. a QoS modification) is needed has when already on-boarded onto the QoE Optimizer instance. Lastly, a mock-up version of the QoS Control CP function is utilized (no CPSR is assumed), which contacts directly with the controller of the segment in which the NSS has been deployed. Aside from these differences, the generic workflow is exercised, achieving an increase of the bandwidth on the NS, thus, modifying its configured QoS.

4.2.3 Architecture and description of components

The general architecture of the QoS Modification actuator follows up the functional/logical architecture explained previously in Section 2.4, combining both the QoE Optimizer and the PF. To demonstrate the functionality of the actuator, a simplified experimental set-up has been deployed, for which Figure 35 depicts the main components and technologies employed.

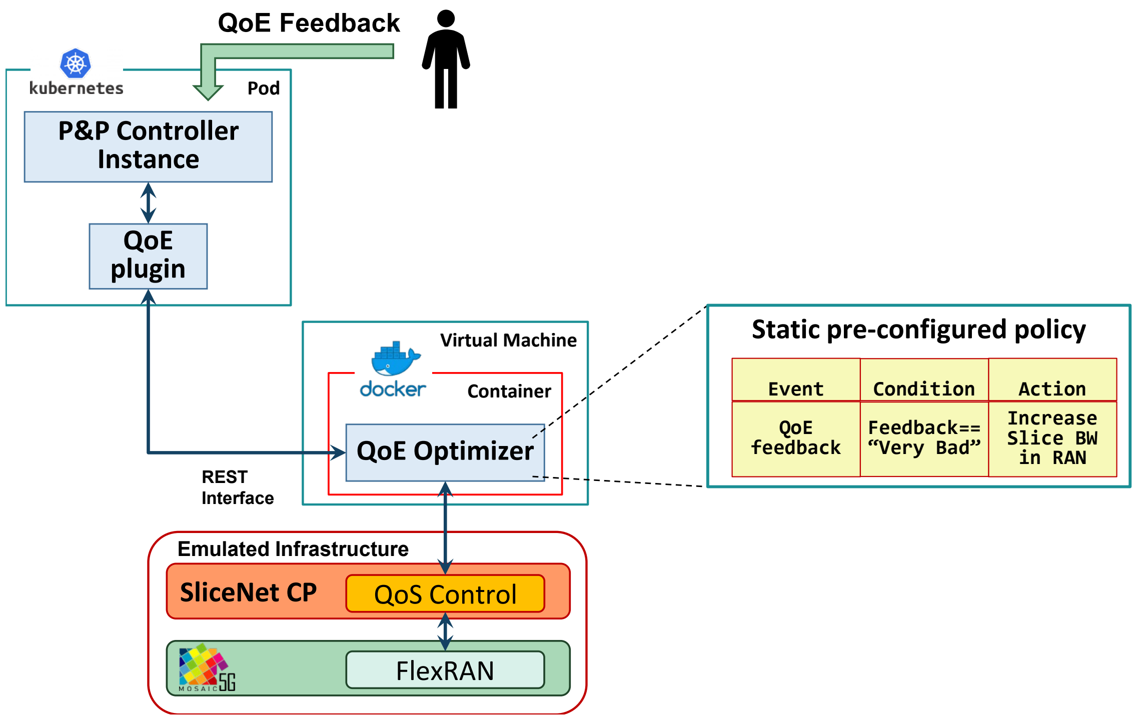


Figure 35 Experimental set-up for QoS Modification actuator

In essence, the experimental set-up consists in a Kubernetes pod in which an instance of the SliceNet P&P controller has been deployed, together with an instance of the QoE plugin. Thanks to the combination of both, the capacity to express the experimented QoE in regards of the NS deployed is exposed towards an external user, which may indicate said quality ranging from "Very Bad" to "Very Good". This feedback is collected by an instance of the QoE Optimizer, which has been containerized in a Docker container inside a VM. AS said previously, for the experimental evaluation of the actuation workflow, a static pre-configured policy is present in this instance of the optimizer, with the concrete details of the policy being depicted in the figure. The policy follows the aforementioned ECA model explained in Section 2.4, dictating that if the received feedback matches a "Very Bad" value, an increase on the NS bandwidth at the RAN segment (NSS) must be performed. In this regard, an emulated infrastructure consisting in a mock-up version of the QoS Control CP function plus a RAN network segment based on Mosaic 5G FlexRAN [27], has been deployed. More specifically, in order to emulate the behaviour of the Northbound API of FlexRAN, the Software Development Kit (SDK) provided by Mosaic 5G Store has been employed. Through this emulated infrastructure, an initial configuration of a RAN slice is performed. Then, once the external user provides a feedback and, in the case that it matches with the value stated by the policy, the QoE Optimizer enforces an increase on the RAN slice bandwidth contacting the QoS Control CP function.

4.2.4 Design of components

Having presented both the overall QoS Modification actuator workflow and the architecture employed to exercise it, in this section we will elaborate about the internal design and implementation of the QoE Optimizer module. Note that the implementation of the module is common for all actuators that are exercised through the QoE Optimizer, with differences resting on the source of the external stimulus that triggers the actuation, the concrete instance of the policies that have been disseminated and are active and the type of actuation as well as its main enabler (CP functions or orchestrator). Having said that, Figure 36 depicts a diagram of the software architecture and design followed in the implementation of the QoE Optimizer module.

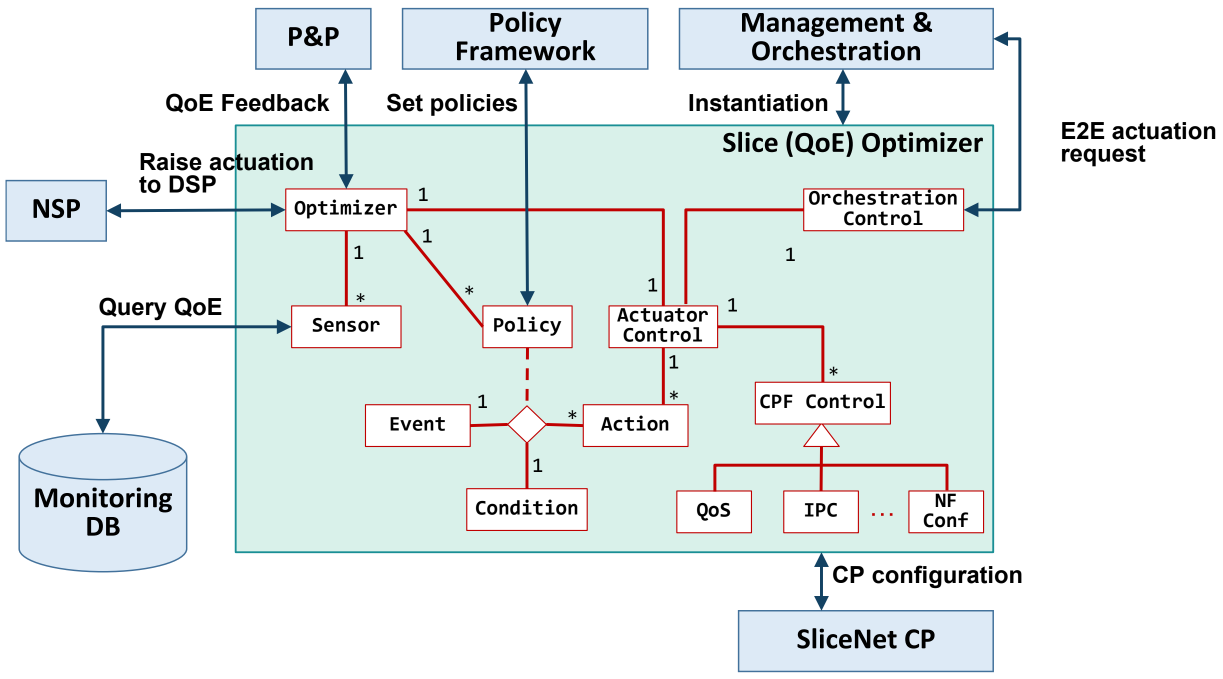


Figure 36 Software design and class diagram of the QoE Optimizer

The module has been developed as a Java application, with several classes implementing its main functionalities. The core of the QoE Optimizer is the **Optimizer** class, which ties together the current active policies for the NS, the actuation control and the external stimulus which trigger the actuations. Specifically, a list of policies is stored in this class and in the case that an external stimulus is received, the **Optimizer** will loop through all stored policies to determine if an actuation is needed. In this regard, the **Sensor** class is the one that models such external stimulus, which may come from three distinct sources: 1) monitoring information from one or multiple QoE/QoS sensors; 2) feedback from the vertical; and 3) request/petition from one of the NSPs in which the E2E NS is sustained. For each active policy, an instance of the **Sensor** class is created, which is the responsible to gather the values associated to the stimulus, either being subscribed to a Data-Lake source or via a direct interface, depending on the implementation approach followed by the external sources of information. For every gathered value, the **Sensor** generates an event which is captured by the **Optimizer** class, passing down the value and the type of event. This becomes the source for the comparison with the list of policies mentioned before.

Each policy is modelled by the **Policy** class, which, in turn, employs an instance of the **Event**, **Condition** and **Action** classes to represent the elements of the policy, following the ECA model mentioned previously during Section 2.4.2. Hence, when an event generated by a sensor is captured by the **Optimizer**, this is checked by the corresponding **Event** in the **Policy**. If the type matches and the value violates the condition stated by the **Condition**, then the **Action** contacts with the **Actuator Control**, which is the responsible to coordinate all the actuations raised from triggered policies. Each **Action** carries the type of actuation that is needed as well as its parameters. According to the type, the **Actuator Control** then contacts the **Orchestration Control**, responsible for all actuations that require the intervention of SliceNet Orchestration plane (e.g. modify the sequence of NSPs), or the **CPF Control**, responsible for all actuations that require the intervention of SliceNet CP, as is the case of the QoS Modification actuator. For this last group of actuations, several classes are employed to model the different CP functions available (e.g. **QoS**, **IPC**, **NF Conf**). Each one of them has a list of the possible operations that the corresponding CP function can implement as well as the types of parameters that are expected. Moreover, they are also the responsible to contact the CPSR in order to retrieve the CP function instance of the slice to enforce the desired action.

In terms of interfaces design, Figure 36 also depicts the expected interfaces with the main components for which the QoE Optimizer requires a communication with them. In this regard, we have several interfaces to gather the information related to the stimulus that trigger the policies as well as an interface between the QoE Optimizer and the PF to allow for the dissemination/update/deletion of policies. Moreover, interfaces between the QoE Optimizer with the main points of enforcement for actions (i.e. the Orchestrator and the CP) are also designed. Lastly, an interface between the QoE Optimizer and SliceNet management plane (particularly, with the SLA/QoE Manager) is also designed in order to allow for the life cycle management of the QoE Optimizer instance. Having said that, current version of the QoE Optimizer has implemented the interfaces between the QoE plugin and the Optimizer as well as the interface with the control plane, which are reported in Section 5. The rest of the interfaces will be developed in further iterations following the progress of all involved components.

4.3 NSP Sequence Modification

4.3.1 Demonstrated functionality

An E2E NS, offered by a DSP, is usually composed by several NSSes which, in turn, may be supported by different NSPs, each one having its own control and management system. Thus, each NSP is responsible to maintain the quality and reliability of the NSSes deployed in its infrastructure. Nevertheless, depending on the scenario, it may happen that despite the corrective measures employed by the NSP to overcome a faulty or underperforming situation, it is not possible to maintain the desired quality levels or the reliability of the NSS, thus affecting the DSP E2E NS. In such an event, if the actual NSP cannot resolve the faulty/degraded NSS, it becomes necessary to replace the actual NSP by another available NSP that can replace that faulty NSS by a healthy NSS, thus maintaining the SLA of the E2E NS offered to the vertical. Such replacement should consider the presence of important/sensitive data that was at the previous NSS before the abnormal situation. As such, a transfer of this information (together with the re-provisioning of the NSS at another NSP), may be also necessary to remedy the situation.

The NSP Sequence Modification actuator is intended to execute the necessary actions to enforce changes on the sequence of underlying NSPs in which the E2E NS sustained. The trigger for such action may come from monitoring information or an explicit trigger from a degraded NSP. It may be also the action associated to a policy related to the feedback received from the vertical, gathered thanks to the QoE Optimizer. Regardless of the source, once triggered, the actuator will contact SliceNet Orchestration plane requesting for a change on the sequence of employed NSPs, indicating concrete NSPs to avoid or desired NSPs to be employed in the new sequence. Such parameters can be inferred directly from the monitoring data or NSPs triggers; alternatively, can be the result of some optimization algorithm, which determines the new sequence to be followed according to a stated goal. The need to involve the Orchestrator in this actuator rises from the necessity of having a detailed view of the overall picture, since changes on the sequence of NSPs are delicate actions which can require the intervention of multiple elements at different layers in a coordinated/orchestrated way. Note that such changes of involved NSPs may also happen in scenarios which require mobility across the E2E NS (e.g. the eHealth UC). In such scenarios, it is also necessary to involve the Orchestration plane so as to successfully enable the cross-over between domains in an (almost) seamless way, rearranging the sequence of NSPs for the E2E NS.

The NSP Sequence Modification actuator is intended to be integrated in the overall SliceNet framework as an enabler for the E2E service/NS optimization loop, exercised across the layered management and control architecture. Such loop will be then exploited for the vertical UCs that require high-reliability on the provisioned NS or the ones which require mobility of the NS end-points, thus requiring on modifications of the NSPs that are being crossed by them.

4.3.2 Scenario

The general scenario that exercises the workflow of the NSP Sequence Modification actuator includes the steps depicted in Figure 37. Since the principles of operation are the same in all workflow-based actuators which have the QoE Optimizer as central piece, the workflow is also divided in three macro-steps, similar to the QoS Modification actuator, for which the first two are in essence the same, that is, monitoring from multiple NSPs and calculation of E2E service/NS metrics. Only the third macro step has some slight differences, as the action to be enforced in this actuator is different. Particularly, in this case, once a metric notification stating that a specific NSP is unreliable or that is violating some performance requirements, the actuation is to modify the sequence of involved NSPs on the E2E NS delivery, avoiding the NSPs which have been marked as faulty/underperforming. As said before, to enforce such action, the QoE Optimizer contacts the Orchestrator at the DSP level, which will take charge of orchestrating the sequence modification, taking charge of all cross-layer dependencies.

As for the first iteration of the actuation system being reported, the overall functionality is not being exercised, since the SliceNet Orchestration sub-system is currently being developed. In this regard, the focus has been on the design of an algorithm to determine the choice of NSPs to be selected for the modified NSP sequence. The designed algorithm will be integrated in next iterations of the QoE Optimizer, with the final goal to be integrated in SliceNet Orchestrator, as to have an overall system view instead of the per-NS view provided by the QoE Optimizer. Given that, next we detail the main scenario, scope and goal of the designed algorithm.

In a scenario in which a reliable NS must be provided, it is essential to first determine the reliability of each of the NSPs. The designed algorithm assumes that a reliability score is ready to be consumed from the DSP inventory, with said metrics/score being gathered directly from the NSP or calculated/predicted through some fault models (similar to techniques reported in Section 3.2). Given such reliability scores, and the targeted reliability when the slice was requested, the algorithm determines the set of NSPs for which the product of their reliability scores **r** (defined as the percentage of how reliable is the NSP, e.g. 90%), is equal or greater than the targeted reliability, subject to excluding the NSPs deemed as unreliable. This may be subject to some cost of choosing a concrete NSP over another as well as neighbouring information among NSPs to guarantee reachability when substituting an NSP by another. Lastly, the selection of the set is also restricted to the number of NSSes that must be provisioned to deploy the E2E NS. All of this defines a knapsack problem, for which the algorithm finds the solution, which is the new NSPs to be employed in the E2E sequence.

Aside from the abovementioned algorithm, note that the implementation of the core functionalities of the QoE Optimizer which will allow for the execution of the NSP Sequence Modification actuator has been carried out (as described in Section 4.2.4), with later iterations focusing on the design and implementation of the interaction between the QoE Optimizer and the Orchestration plane.

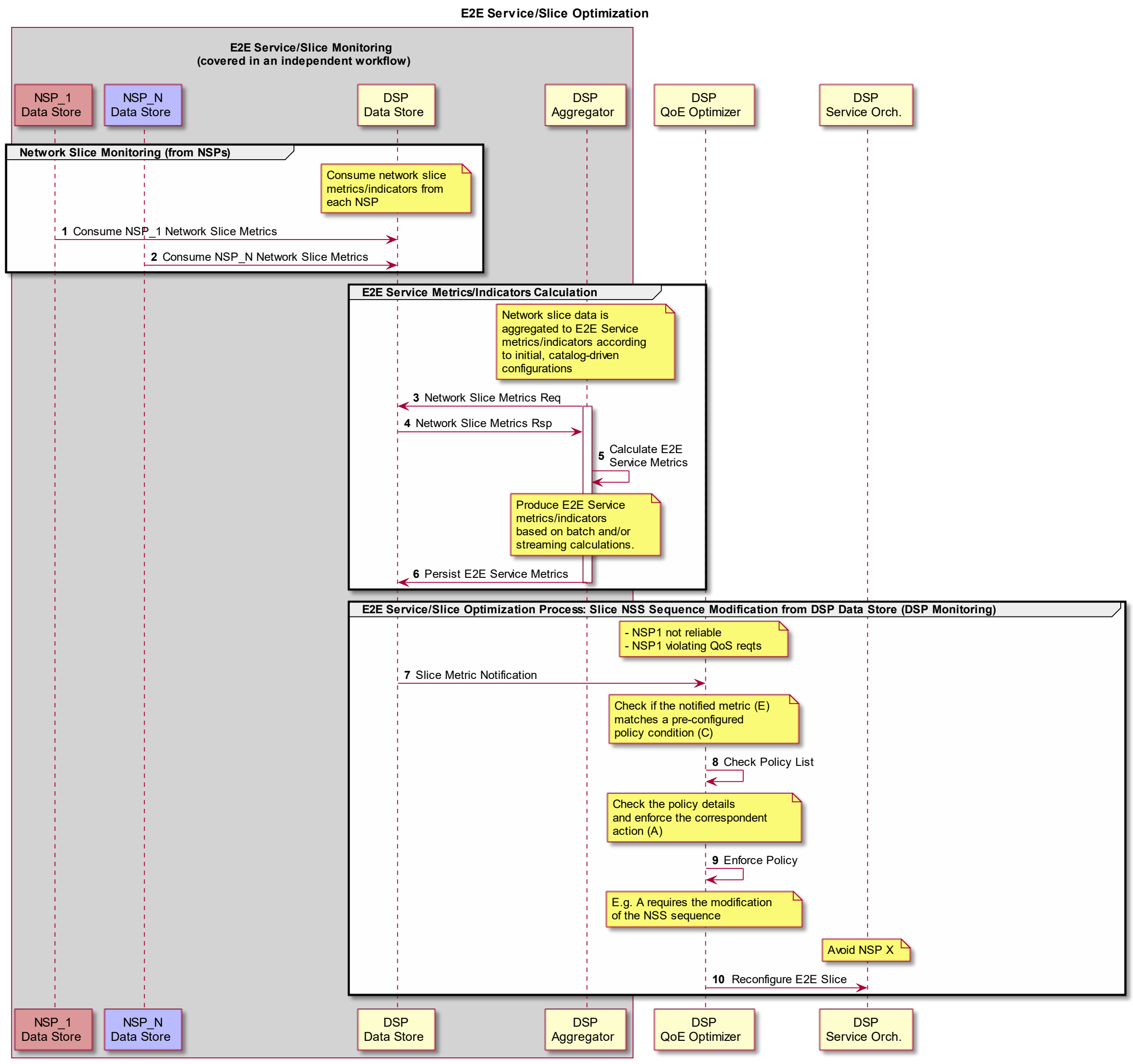

E2E Service/Slice Optimization
E2E Service/Slice Monitoring (covered in an independent workflow)
NSP_1 Data Store
NSP_N Data Store
DSP Data Store
DSP Aggregator
DSP QoE Optimizer
DSP Service Orch.
Network Slice Monitoring (from NSPs)
Consume network slice metrics/indicators from each NSP
1 Consume NSP_1 Network Slice Metrics
2 Consume NSP_N Network Slice Metrics
E2E Service Metrics/Indicators Calculation
Network slice data is aggregated to E2E Service metrics/indicators according to initial, catalog-driven configurations
3 Network Slice Metrics Req
4 Network Slice Metrics Rsp
5 Calculate E2E Service Metrics
Produce E2E Service metrics/indicators based on batch and/or streaming calculations.
6 Persist E2E Service Metrics
E2E Service/Slice Optimization Process: Slice NSS Sequence Modification from DSP Data Store (DSP Monitoring)
- NSP1 not reliable
- NSP1 violating QoS reqts
7 Slice Metric Notification
Check if the notified metric (E) matches a pre-configured policy condition (C)
8 Check Policy List
Check the policy details and enforce the correspondent action (A)
9 Enforce Policy
E.g. A requires the modification of the NSS sequence
Avoid NSP X
10 Reconfigure E2E Slice
NSP_1 Data Store
NSP_N Data Store
DSP Data Store
DSP Aggregator
DSP QoE Optimizer
DSP Service Orch.

Figure 37 NSP Sequence Modification actuator general workflow

4.3.3 Architecture and description of components

In this section we will focus on detailing the logical structure and steps of the aforementioned algorithm as well as the simulation setup employed for its initial validation. Figure 38 depicts the approach followed by the algorithm to solve the aforementioned knapsack problem. In this regard, for the first implementation, it is assumed that the goal of the algorithm is to select from a list of available NSPs, a subset that, when combining their reliability scores, it reaches the targeted overall reliability, without any consideration in terms of reachability between NSPs. Thus, the only constraints are to avoid faulty NSPs and to match the size of the list of selected NSPs with the size of the initial list.

Given such constraints, the steps of the algorithm are summarized as:

1. The algorithm reads from an external source the information related to available NSPs. The assumed information is their identifiers, reliability score and a cost value associated to them. In this regard, less costly NSPs are more desirable than others. Additionally, other considered inputs of the algorithm are the targeted overall reliability score, the number of NSPs to be selected as well as the ones to be avoided. With such information, the algorithm eliminates from the read set the NSPs marked as unreliable. In general terms, the information related to the NSPs and other inputs would be accessed from inventories and data-stores at the DSP level, but for the PoC, the information is accessed by reading a text file where the information has been manually introduced.
2. The algorithm then computes the ratio between the reliability of each NSP by their cost, since more reliable options are more desirable as well as less costly ones.
3. The algorithm selects the first NSPs from the ordered set without surpassing the targeted size. In this regard, the algorithm is designed as a pure greedy algorithm, selecting the solution elements that have the largest contribution towards the desired goal.
4. As a last step, the algorithm checks if the product of the reliability scores of the selected NSPs is equal or above the targeted overall reliability. If so, the details of the selected NSPs are exported in a file text. Specific interfaces to pass the solution towards the Orchestrator for its consumption will be designed in following iterations.

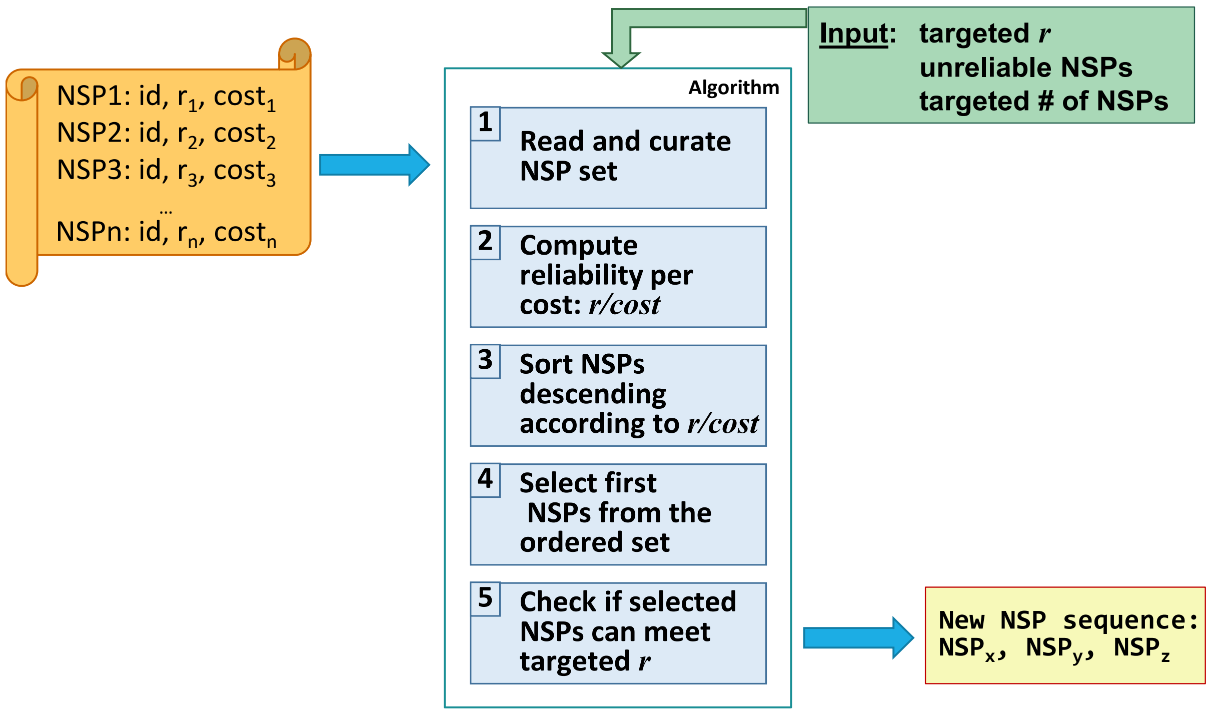

NSP1: id, r_1, $cost_1$
NSP2: id, r_2, $cost_2$
NSP3: id, r_3, $cost_3$
...
NSPn: id, r_n, $cost_n$
Input: targeted r
unreliable NSPs
targeted # of NSPs
Algorithm
1 Read and curate NSP set
2 Compute reliability per cost: $r/cost$
3 Sort NSPs descending according to $r/cost$
4 Select first NSPs from the ordered set
5 Check if selected NSPs can meet targeted r
New NSP sequence: NSP_x, NSP_y, NSP_z

Figure 38 Logical design for reliable NSP selection in NSP Sequence Modification actuator

Future iterations of the algorithm will also refine the selection of the NSPs (e.g. multi-start, metaheuristics) to provide a more optimized selection of NSPs, while also accounting for limitations on the reachability between NSPs or maintenance of still reliable NSPs in the original sequence, that is, only replacing the ones deemed as faulty.

4.4 OVS-based User Traffic Classification

4.4.1 Demonstrated functionality

A “Programmable Software Data-Path” together with a flexible definition of network slices, will allow controlling traffic in 5G infrastructure, as a means to control the QoS of the slices. This is especially important in architectures where user mobility and tenant isolation are essential requirements and have to be supported both at the same time. From the NSP point of view, these requirements are key aspects and they imply the use of nested encapsulation. It is also necessary to isolate the performance of different 5G users belonging to different tenants and to isolate the performance of the different tenants using the same physical infrastructure. In summary, in a 5G scenario, a “Programmable Software Data-Path” has to offer fine-grained control over 5G flows, allowing slicing at 5G user-level to provide simultaneous control of users, tenants, and infrastructure.

There are a number of "Programmable Data Paths" available such as FD.io [28], XDP [29], PACKET_MMAP [30], PF_RING [31], SNABB switch [32] or DPDK [33] but none of these enable fine-grained capabilities required in a 5G environment such as 5G flow queue isolation and tenant queue isolation, among others. Meanwhile, OVS has become a de facto standard in SDN/NFV environments and is the most widely used currently in 5G environments [34]. For this reason, we have chosen OVS as the data-path software, and enhanced it through extending the source code of the OVS official Git repository.

Given the above mentioned 5G requirements, the "OVS-based User Traffic Classification" provides novel mechanisms designed and prototyped in the context of this project to contributing to controlled performance in 5G traffic:

1. **5G flow specification:** Low-level implementable and flexible specification of a 5G Network Slicing definition according to different criteria such as Tenant or User identifier, and others.
2. **QoS requirements:** For each defined 5G Network Slice, QoS control is enforced to allow performance tenant isolation and 5G user isolation.
3. **Dynamic configuration of a slice:** Change in real time any of the parameters of a 5G Network Slice definition and its QoS associated, through actuations including setting new bandwidth, redirecting traffic, dropping traffic, etc.

All these new 5G OVS functionalities have been designed, prototyped and empirically tested. The OVS software component is ready to be integrated into SliceNet testbeds.

4.4.2 Scenario

This OVS extended software can be controlled by the control plane of the SliceNet framework, specifically the QoS Control component. Therefore, the workflow for it until that point is similar to what is presented in Section 4.2.2. The following sequence diagram shown in Figure 39 further depicts the interactions between the OVS-based technology and primarily the CP modules in particular the QoS Control for the specific technical UC of creating a new slice definition. As it can be observed, the communication between the QoS Control (QoS for brevity in the figure) and the OVS technology is implemented using the interface provided by the Flow Control Actuator (FCA) [24] for such a purpose. All slice definitions inserted in the OVS actuator remain in the user space as OpenFlow 1.2+ rules and only become kernel space flows when necessary, thereby achieving a great improvement in performance in this way. When rules are implemented in the kernel space, they enforce the slice in the software data path.

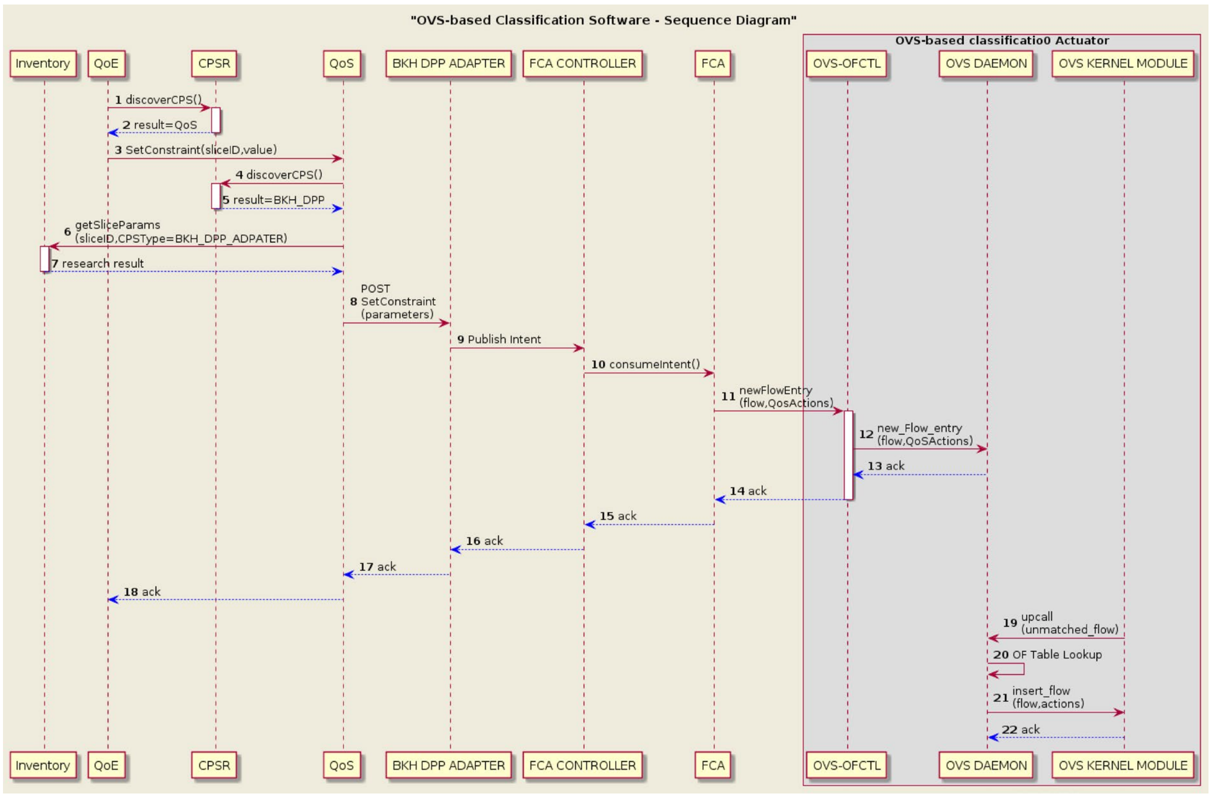

"OVS-based Classification Software - Sequence Diagram"
OVS-based classificatio0 Actuator
Inventory
QoE
CPSR
QoS
BKH DPP ADAPTER
FCA CONTROLLER
FCA
OVS-OFCTL
OVS DAEMON
OVS KERNEL MODULE
1 discoverCPS()
2 result=QoS
3 SetConstraint(sliceID,value)
4 discoverCPS()
5 result=BKH_DPP
6 getSliceParams (sliceID,CPSType=BKH_DPP_ADPATER)
7 research result
8 POST SetConstraint (parameters)
9 Publish Intent
10 consumeIntent()
11 newFlowEntry (flow,QosActions)
12 new_Flow_entry (flow,QoSActions)
13 ack
14 ack
15 ack
16 ack
17 ack
18 ack
19 upcall (unmatched_flow)
20 OF Table Lookup
21 insert_flow (flow,actions)
22 ack

Figure 39 Workflow sequence for the OVS-based traffic classification

4.4.3 Architecture and description of components

The OVS software has been extended to create this new actuator that allows classification of 5G traffic. Architecturally, the standard components and interfaces in the standard OVS remain the same; however, almost all the components in the standard architecture have been extended in order to allow the new 5G traffic classification capabilities introduced. Figure 40 depicts the extended functionalities of the different OVS architectural modules and which components are extended with patches to add the new 5G features.

The new 5G features have been added by extending code, data structures and protocols in each of the modules required. The OVS architecture is composed of 3 layers: Kernel space, User space and Management space.

- **Kernel Datapath:** It keeps both flow and action tables. This module receives packets and performs actions associated with each flow. If a flow does not match in the flow table, it is sent to user space.
- **User Space Layer:** This module keeps OpenFlow Tables. The purpose of this module is to convert OpenFlow tables into kernel-level flows to achieve high scalability, high throughput and high performance.
- **Management Layer:** OVS provides a set of applications to perform the management tasks inside the different OVS components. Each of these applications offers a command line using which it manages an OVS switch software.

As shown Figure 40, more than 8 different extensions have been designed and implemented in order to provide the new capabilities.

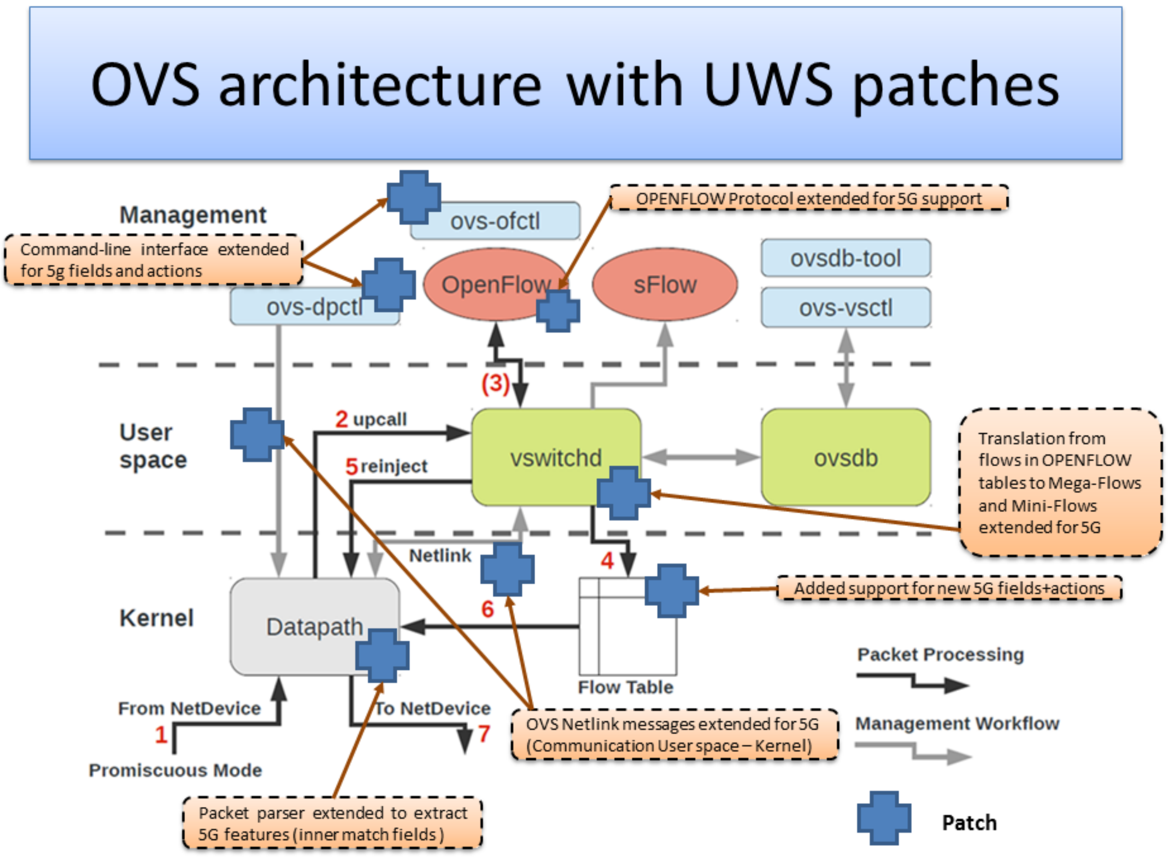

OVS architecture with UWS patches
Management
Command-line interface extended for 5g fields and actions
ovs-ofctl
OPENFLOW Protocol extended for 5G support
ovs-dpctl
OpenFlow
sFlow
ovsdb-tool
ovs-vsctl
(3)
User space
2 upcall
5 reinject
vswitchd
ovsdb
Translation from flows in OPENFLOW tables to Mega-Flows and Mini-Flows extended for 5G
Netlink
4
Added support for new 5G fields+actions
Kernel
Datapath
6
Flow Table
Packet Processing
From NetDevice
1
To NetDevice
7
OVS Netlink messages extended for 5G (Communication User space – Kernel)
Management Workflow
Promiscuous Mode
Packet parser extended to extract 5G features (inner match fields)
Patch

Figure 40 Architecture and components of OVS-based traffic actuator

4.4.4 Design of components

As indicated in Section 4.4.3, the basic architecture of OVS was not redesigned. On the contrary, the new requirements for an integration of OVS-based User Traffic Classification actuator into a 5G architecture have been added by extending functionality of existing modules, modifying data structures and extending protocols (OpenFlow and Netlink). The innovations made to OVS are briefly described below:

1. **Extract 5G-Key:** When a new packet enters the OVS switch, in addition to the current parsing performed by OVS, a new packet parsing has been designed and implemented. This parsing allows obtaining information from the most inner headers of the packet, including information about the stack of nested encapsulations (Protocol and Tunnel identifier).
2. **Flow table in Kernel Space (Mini-Flows and Mega-Flows):** New fields have been added to the standard flow definition of OVS. These new fields refer to the information of the inner headers of each packet as well as the encapsulation stack (Type of encapsulation and Tunnel identifier). These new fields define what we call the network slice definition in the data path and allows performing actions over the defined slice.
3. **New Actions:** For example, a new action intended to copy the Differentiated Services Code Point (DSCP) value from the inner IPv4 header into the outer DSCP value in order to allow a vertical control signalling between NSP and DSP.
4. **Netlink Protocol:** OVS uses the Netlink protocol to communicate between kernel and user spaces. Netlink messages have been extended to allow the new fields and actions supported.
5. **OpenFlow Protocol Extension:** The OpenFlow protocol has been extended with new fields and actions.

6. **ovs-ofctl (OpenVSwitch OpenFlow Control):** This module provides a command line through which OpenFlow tables can be handled. In this module, it has been necessary to implement new functionality to:
    a. Be able to read, via command line, an OPF specification with the new fields and actions supported.
    b. Be able to build OpenFlow messages containing the new fields and actions to send them to the daemon in the OVS user space.
7. **ovs-dpctl (OpenVSwitch Data Path Control):** It communicates directly with the OVS kernel module, using the Netlink protocol. Therefore, it should be used only in controlled environments and for experienced purposes. It is not recommended to use ovs-dpctl in real use cases. However, we have adapted the ovs-dpctl command line interface to add support to the new fields.

5 Interfaces and APIs

In this section, we report the main interfaces across functional modules of the MAPE-K loop as well as external interfaces towards other components of SliceNet architecture which intervene on the operations described across the deliverable. The focus here is on reporting the interfaces that have been designed or implemented, leaving the documentation of other interfaces and APIs for future developments on the overall Cognition Plane framework.

5.1 Policy Framework Interfaces

As referred previously in Section 2.4.2, the ONAP Policy Subsystem will be adopted and integrated in the project as its PF. The management of the policies is done through a Representational State Transfer (REST) API and its full documentation can be found on the ONAP's official documentation [35]. However only a subset of the API's endpoints is relevant for this context, as described below:

- **createPolicy**: creates a policy based on given policy parameters;
- **pushPolicy**: pushes the specified policy to a PDP (group);
- **updatePolicy**: updates a policy based on the given policy parameters;
- **deletePolicy**: deletes the specified policy from the PDP (group);
- **getConfig**: gets the configuration of a policy based on the given policy name.

In order for the PDPs to receive the policies, they must subscribe policy notifications from the PAP. The notifications are sent through a web socket in Java Script Notation Object (JSON) format following a structure as shown in Figure 41.

```
{
  "removedPolicies":[
  ],
  "loadedPolicies":[
    {
      "policyName":"com.Config_BRMS_Param_BRMSParamvFirewall.1.xml",
      "versionNo":"1",
      "matches":{
         "ONAPName":"DROOLS",
         "ConfigName":"BRMS_PARAM_RULE",
         "guard":"false",
         "TTLDate":"NA",
         "RiskLevel":"5",
         "RiskType":"default"
      },
      "updateType":"NEW"
    }
  ],
  "notificationType":"UPDATE"
}
```

Figure 41 PAP notification example

Upon receiving a notification, the PDP must evaluate the retrieved information and remove or load the policies in question. If a policy must be loaded by the PDP, its configuration must be retrieved from the PAP using the *getConfig* endpoint of the REST API and based on the specified policy name.

5.2 QoE Optimizer CP Interface

One of the main endpoints in which the QoE Optimizer enforces the desired actions is SliceNet control plane. In this regard, two main groups of operations are required to be performed by the QoE Optimizer. First, when the actual actuation has been determined, also identifying the function that needs to be involved, the QoE Optimizer needs to retrieve the instance of the corresponding control plane function. For this, it needs to contact the CPSR, which will return the instance of the function associated to the NS. Next, the QoE Optimizer needs to contact the retrieved function to execute the desired operation (e.g., modify QoS parameters), employing the API exposed by the function.

In this regard, the QoE Optimizer follows the APIs reported in deliverable D4.3 for both retrieving control plane functions instances from the registry as well as the APIs for each of the functions, mainly, QoE Control, IPC and NF Conf. The focus of the current implementation has been on the API enabling the communication with the CPSR and the API exposed by the QoS Control function. Additional APIs will be implemented in later stages following the development of other planned actuators.

5.3 QoE Optimizer - QoE plugin Interface

As said previously, one of the main sources of external stimulus that trigger an actuation from the QoE Optimizer is the received feedback from the vertical. From the QoE Optimizer perspective, this indicates a bad experience from the user, which may not be exactly reflected on the received monitoring data in regards of the NS, due to the subjective nature of the perception of the vertical. In such a case, this mandates for a run-time actuation to remedy potential bad situations (i.e. negative feedback from the vertical), for which rules (policies) will dictate the action to be applied.

The element that enables this dynamic run-time feedback and, as a consequence, the reaction from the QoE Optimizer is the QoE Plugin developed in the context of the P&P controller. In the original development reported in D4.1, the incipient functionalities of the QoE Optimizer where integrated within the QoE plugin as a single monolithic module. In the current implementation being reported, both elements (plugin and optimizer) have been separated and an interface to allow for sending the feedback towards the QoE Optimizer has been developed. The interface is based on REST, with Table 4 summarizing its main details.

Table 4 QoE feedback API details

Endpoint	http://<IP>:<PORT>/ IP: address of the QoE Optimizer instance PORT: port in which the QoE Optimizer is listening for QoE feedbacks
HTTP operation	PUT
Description	It enables the reception of QoE feedback from the vertical by the QoE Optimizer for run-time actuations
Caller	QoE plugin
Request body	JSON, see example in Figure 42
Response body	None
Response code	200 OK - Feedback received correctly 400 Bad Request - A generic problem happened with the feedback operation

The QoE feedback is sent in the form of a **QoEObject**, for which Figure 42 details its structure and main fields.

```
QoEObject:
type: object
  properties:
    id:
      type: string
      example: “3b3888dc-3502-11e9-b210-d663bd873d93”
    value:
      type: string
      example: “VERY BAD”
```

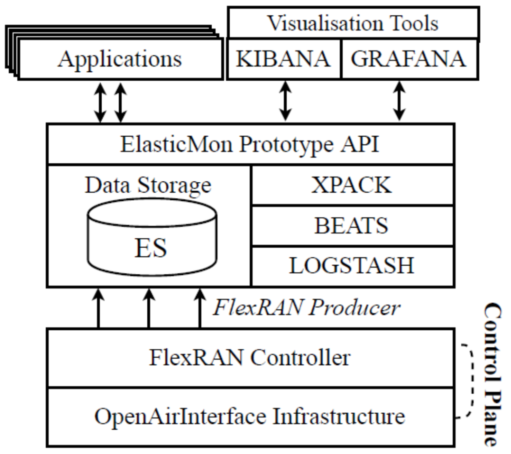
Figure 42 Schematic of a QoEObject

Essentially, a QoEObject carries two fields with necessary information. On the one hand, the **id** field relates to the identifier of the end-to-end NS for which the feedback is being sent. On the other hand, the **value** field carries the actual value of the feedback as expressed by the vertical through the capabilities exposed by both the P&P controller and the OSA.

6 Summary of software components

In this section, we provide a summary of the main software components presented along the deliverable through the multiple exercised analytical and actuation workflows (Sections 3 and 4). Links to their codebase can be found for each one of them. These preliminary software developments will serve as foundation for later iterations of the Cognition Plane implementation.

6.1 ElasticMon

Name	**ElasticMon**
Description	ElasticMon is a monitoring framework crafted especially for the needs and monitoring challenges of 5G mobile networks following the principles described in Deliverable D5.2. It serves as a pipeline for storing, retrieving and exchanging a rich set of data between monitoring applications, allowing flexible data flows between monitoring applications that can be developed to collect, process and/or consume monitoring data. ElasticMon supports the monitoring of massive and (quasi-) real-time data streams per slice for the purposes of network control and management, as well as both flat and hierarchical deployments to match the network architecture and to allow to respect the data-ownership model across different players ranging from infrastructure and service providers.
License	Apache v2.0
Version	1.0
Design	 Visualisation Tools Applications KIBANA GRAFANA ElasticMon Prototype API Data Storage ES XPACK BEATS LOGSTASH FlexRAN Producer FlexRAN Controller OpenAirInterface Infrastructure Control Plane *Figure 43 Prototype implementation of ElasticMon monitoring Framework* The current prototype version of ElasticMon works as a framework extension on top of the ElasticSearch search engine[2] and the FlexRAN. Figure 43 portrays the modules of ElasticMon's prototype implementation and their interaction with ElasticSearch, the FlexRAN hierarchical controller, and the OAI. FlexRAN's controller runs over an OAI user plane network infrastructure, whereas ElasticSearch is used to store the control plane data from the southbound API (the FlexRAN producer). The FlexRAN Producer API is a dedicated lightweight application deployed over FlexRAN that stores the gathered raw monitoring data to the data store provided by ElasticSearch. Note

[2] https://www.elastic.co/

	that an arbitrary number of UE devices can be indexed by the FlexRAN producer application and that monitoring data/UE aggregation is also supported, including aggregation per slice. But apart from supporting multi-slice monitoring, ElasticMon v1.0 can also work with different data stores in the background, whereas different ElasticMon instances can be used together over one or more datastore instances. These features enhance privacy via the isolation of data stores between different stakeholders (slices, NSPs and DSPs) alongside the filtering mechanisms that are native to ElasticSearch. A special **wiki manual** with a detailed description of the architectural design and prototype implementation is available here: https://gitlab.eurecom.fr/mosaic5g/mosaic5g/wikis/tutorials/elasticmon-manual
Codebase	• **Code available here:** https://gitlab.eurecom.fr/mosaic5g/elasticmon/tree/develop • **A tutorial wiki** with all necessary details for downloading, deploying, configuring and running ElasticMon is available here: https://gitlab.eurecom.fr/mosaic5g/mosaic5g/wikis/tutorials/elasticMon-tutorial Note: *To gain access to the above GitLab sources about ElasticMon, reviewers will be provided with a special guest account on demand*[3] *by Eurecom.*

6.2 SkyDive

Name	**SkyDive**
Description	Skydive is an open source real-time network topology and protocols analyser providing a comprehensive way of understanding what is happening in your network infrastructure.
License	Apache v2.0
Version	0.21.0
Design	http://skydive.network
Codebase	https://github.com/skydive-project/skydive

6.3 Stresser for QoE-QoS experiment

Name	**Network_stresser test suite for Kubernetes**
Description	This test suite consists in Helm charts for network bandwidth testing a Kubernetes cluster.
License	NA
Version	NA
Design	iperf based K8s stresser
Codebase	https://github.com/cognetive/network_stresser

6.4 Noisy neighbour experiment

Name	**Noisy neighbour detection model**
Description	The model classifies the status of the VNF to one of the following statuses based on the consumed CPU, memory, and network bandwidth: • Normal status

[3] Please, contact Prof. Navid Nikaein with an email sent to navid.nikaein@eurecom.fr, mentioning "SLICENET reviewer access to GitLab request" in the subject of your email, to gain access to the codebase and wiki pages.

	• Noise status • Overload status
License	Apache v2.0
Version	NA
Design	Section 3.3
Codebase	https://gitlab.com/slicenet/noisy-neighbor

6.5 Machine learning pipeline

Name	**Cognition pipeline**
Description	The ML pipeline consists of several processing steps that normalize the data, create several event windows (of different types) and "classify" them. It is composed of several components: • **Gold**: consumes the four stream data inputs, normalizes and enriches them; • **Windows**: takes JSON events and build time, time and/or sequence windows; • **Classifier**: tags event windows with intervention tickets stream data; • **Kafka2Elastic**: converts JSON stream data and publishes it into ElasticSearch.
License	GNU GPLv3
Version	1.0.1
Design	Section 2.3.1
Codebase	https://gitlab.com/slicenet/cog-pipeline

6.6 Cognition Notebooks

Name	**Cognition Notebooks**
Description	Jupyter notebooks repository that contain data exploration and ML code.
License	NA
Version	NA
Design	-
Codebase	https://gitlab.com/slicenet/cog-notebooks

6.7 QoE Optimizer

Name	**QoE Optimizer**
Description	The QoE Optimizer is the responsible to gather E2E NS QoE/QoS-related monitoring data from DSP datastores/monitoring sources and trigger necessary actuations if some corrective actions are needed to maintain quality levels. The actions are ruled thanks to a policy system provided by the PF. As for the actions, they may entail to engage with the orchestration sub-system at the DSP level or Rule Enforcer/Control Plane at the NSP level. Current prototype includes the general functionality of the QoE Optimizer as well as the communication with the QoE plugin and SliceNet CP
License	NA
Version	1.0
Design	Sections 2.4.3 and 4.2.4
Codebase	https://gitlab.com/slicenet/qoe-optimizer

6.8 OVS-based user traffic classification actuator

Name	**OVS-based user traffic classification software**
Description	This software-based traffic clasifier able to parse and control 5G multi-tenant user traffic in the data plane according to the traffic control rules received. Different traffic control rules are allowed to be applied to specific flows/sub-slices at a non-RAN

	network segment, including traffic dropping, setting new bandwidth, traffic redirection etc. The prototyping is based on extending the capabilities of OVS.
License	Apache v2.0
Version	OpenVSwitch version 2.9.2 patched with 5G extensions
Design	Section 4.4
Codebase	https://gitlab.com/slicenet/openvswitch

7 Conclusions

This deliverable has presented the design of the overall Cognition Plane within SliceNet's management layer and provided the first iteration for the design and implementation of the several elements that will constitute the phases of the full cognition MAPE-K-based loop, with special emphasis on the Analytics and Actuation parts. The main goal of the Cognition Plane is to enable the automated and QoE-aware management and control of E2E NS as offered by DSPs entities towards vertical customers. To this end, it is essential to provide means for analysis of the underlying NS and its components to determine its quality levels as well as for applying (re-)configurations when needed to maintain the desired quality levels, all in all keeping a QoE-aware/E2E NS context.

In this regard, SliceNet's Cognition Plane provides a multi-layer/level approach, in which several cognition loops are exercised at different stratums of the full architecture to allow for a holistic and automated management of E2E NS at DSP level and NSSes at NSP level. The focus of the design and implementation of the Cognition Plane resides at the E2E level. Nevertheless, the means to allow for cross-entity NS management are also designed as it is a crucial part for fully automated management of E2E NS. In this regard, SliceNet has designed a novel policy-ruled Actuation Framework, which determines when and how (re-)configurations should be made on the underlying NS/NSSes, engaging with the different points of execution at different levels to enforce the desired actions. Moreover, the policy-based approach allows for a unified way to dictate how the overall system and its individual components (both at DSP and NSP levels) should react to specific events. This is enabled thanks to the data-centric and analysis approach followed by the whole Cognition Plane.

SliceNet follows a Data-Lake approach, in which all source of monitoring (telemetry, traffic level, topology level, etc.) are logically centralized into a single source for the data, allowing for external elements, such as the analytics, to gather desired data, elaborate on that, and insert it again in the unified Data-Lake for later consumption by other elements of the Cognition Plane (such as the Actuation Framework). This Data-Lake approach is also enriched by feedback data provided by vertical customers thanks to the unique capabilities of the P&P controller, which allows to integrate the verticals' view in the loop and provide a more accurate QoE perspective of NSes. A novel cognition pipeline has been designed to ingest the data found at the Data-Lake, preparing it for several learning algorithms that have been specially designed to cover multiple aspects of SliceNet vertical UCs.

In this regard, the deliverable reports several proof-of-concepts, with specific software implementations, exercising the multiple phases and workflows of the Cognition Plane. These workflows successfully demonstrate the ability to process data sources to serve as foundation for the Data-Lake approach. Moreover, several analysis cases are presented, both in order to determine QoE levels from monitored data as well as to determine anomalous, faulty or underperforming situations which would then require an actuation. Lastly, several actuation cases are presented, highlighting both actuations based on workflows (for E2E/DSP level actuation) and based on specific functions (for more NSP-oriented level actuation). The presented designs and implementations will be consolidated and expanded in future iterations of the overall Cognition Plane.

References

[1] SliceNet Deliverable D2.4, “Management Plane System Definition, APIs and Interfaces”, SliceNet Consortium, May 2018.

[2] 5GPPP Network Management & Quality of Service Working Group, “Cognitive Network Management for 5G”, white paper, March 2017.

[3] SliceNet Deliverable D5.2, “Modelling and Design of Vertical-Informed QoE Sensors”; SliceNet Consortium, December 2018.

[4] Andrew Lerner, “AIOps Platforms”, Gartner Blog, August 2017. [online]: https://blogs.gartner.com/andrew-lerner/2017/08/09/aiops-platforms/

[5] N. Miloslavskaya, A. Tolstoy, “Big Data, Fast Data and Data Lake Concepts”, Elsevier Procedia Computer Science, vol. 88, pp. 300-305, October 2016.

[6] ETSI, “Network Function Virtualization (VNF); Management and Orchestration”, December 2012.

[7] SliceNet Deliverable D4.2, “Network Slicing in 5G RAN-Core”, SliceNet Consortium, December 2018.

[8] ONAP Policy Framework [online]: https://wiki.onap.org/display/DW/The+ONAP+Policy+Framework

[9] Altice Labs Alarm Manager wiki [online]: https://wikis.ptinovacao.pt/display/SLICENET/AlarmManager+dataset+description

[10] Altice Labs Data Analysis [online]: https://wikis.ptinovacao.pt/display/SLICENET/3+-+Data+Analysis

[11] SliceNet Deliverable D2.1: “Vertical Sector Requirements Analysis and Use Case Definition”, SliceNet Consortium, November 2017.

[12] Prometheus - Monitoring system & time series database [online]: https://prometheus.io/

[13] OpenStack [online]: https://www.openstack.org

[14] Traffic generator for the SIP protocol [online]: http://sipp.sourceforge.net/

[15] IBM Cloud Private Service [online]: https://www.ibm.com/cloud/private

[16] Apache JMeter [online]: https://jmeter.apache.org/

[17] Iperf TCP/UDP traffic generator [online]: https://iperf.fr/

[18] Skydive – Real-time network analyser [online]: http://skydive.network/

[19] ETSI Technical Specification TS 138.401, “NG-RAN; Architecture description”, ETSI, July 2017.

[20] OpenAirInterface [online]: http://www.openairinterface.org/

[21] Mosaic 5G platform [online]: http://mosaic-5g.io/

[22] C.-Y. Chan et al., “Spectrum management application - A tool for flexible and efficient resource utilization,” in Proceedings of IEEE Global Communications Conference 2018 (GLOBECOM 2018), Abu Dhabi (UAE) 9-13, December 2018.

[23] R language – The R project for statistical computing [online]: https://www.r-project.org/

[24] SliceNet Deliverable D4.3, “Single-Domain, Multi-Tenant Network Slicing Control”, SliceNet Consortium, December 2018.

[25] SliceNet Deliverable D4.1, “Plug & Play Control Plane for Sliced Networks”, SliceNet Consortium, October 2018.

[26] R. Montero et al., “Supporting QoE/QoS-aware end-to-end network slicing in future 5G-enabled optical networks”, Proceedings of SPIE Photonics West 2019 (PW 2019), San Francisco (USA), February 2019.

[27] X. Foukas et al. “FlexRAN: A Flexible and Programmable Platform for Software Defined Radio Access Networks,” in Proceedings of 12th International Conference on Emerging Networking Experiments and Technologies, 2016.

[28] Fd.io project [online]: https://fd.io

[29] eXpress Data Path (XDP) [online]: https://www.iovisor.org/technology/xdp

[30] Packet_mmap howto [online]: on https://sites.google.com/site/packetmmap

[31] PF_RING [online]: https://www.ntop.org/products/packet-capture/pf_ring
[32] SNABB project official git repository [online]: https://github.com/snabbco/snabb
[33] Data Plane Development Kit [online]: https://www.dpdk.org
[34] F. Z. Yousaf et al., "NFV and SDN - key technology enablers for 5g networks," IEEE Journal on Selected Areas in Communications, vol. 35, num. 11, pp. 2468-2478, November 2017.
[35] ONAP Policy Engine Platform - Offered APIs [online]: http://docs.onap.org/en/casablanca/submodules/policy/engine.git/docs/platform/offeredapis.html