Deliverable D5.6

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

Iteration 2

Editors:	Salvatore Spadaro, Universitat Politècnica de Catalunya (UPC) Dean Lorenz, IBM Research Haifa (IBM)
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	04/10/19
Actual delivery date:	04/10/19
Suggested readers:	Network Administrators, Vertical Industries, Telecommunication Vendors, Telecommunication Operators, Service Providers
Version:	1.1
Total number of pages:	89
Keywords:	Network Slicing, AIOps, Network Slice, 5G, SDN, Cognitive management

Abstract

This document reports the second iteration of the SliceNet Cognition Plane, expanding the design and implementation of its components. SliceNet Cognition Plane enables the Quality of Experience (QoE)-aware management of network slices. 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). To this end, the components reported in this document provide the implementations of both analytical and actuation frameworks of the Cognition Plane, all governed through a data-centric approach. In addition to the design and implementation of the several components, the current document provides the results obtained through several experimental set-ups as well as integration tests that demonstrate the achievement of the integration of the Cognition Plane components. The developed Cognition Plane will be exploited by both SliceNet's FCAPS (Fault, Configuration, Accounting, Performance and Security) management and Orchestration systems to aid in the management and orchestration of 5G slices, paving the path towards the final integration of the overall SliceNet framework.

Disclaimer

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 2
[Editor: Name, company]	Salvatore Spadaro, Universitat Politècnica de Catalunya (UPC); Dean Lorenz, IBM Research - Haifa (IBM)
[Work-package leader]	Dean Lorenz, IBM Research - Haifa (IBM)

Copyright notice

Executive summary

The provisioning of network slices (NSes) 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 is the second iteration of the overall Cognition Plane design and implementation. It demonstrates work package-level integration of the different Cognition Plane components and initial integration with other SliceNet components such as Monitoring, FCAPS (Fault, Configuration, Accounting, Performance and Security) management, and Plug and Play (P&P) control. It provides means to validate SliceNet's approach and architecture for QoE cognitive management of NSes, including interfaces and prototypes.

To this end, it builds on the foundations reported during iteration I for all the Monitor-Analyse-Plan-Execute governed by a Knowledge-base (MAPE-K) control loop elements and expands them, providing refinements in both the logical design and implementation as well as reporting several results coming from multiple implemented Proof-of-Concepts (PoCs) and integration tests. The obtained results highlight the performance of the several ML models and technical approaches that help in the realization of the multiple cognition use cases (UCs) considered within the context of SliceNet vertical UCs. Thanks to them, the QoE-aware management of NSes is enabled.

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. The architecture will be leveraged during the development of SliceNet FCAPS and Orchestration systems (WP6 and WP7, respectively), to be finally integrated within the vertical UCs defined by SliceNet (WP8). As such, the final iteration of WP5 will revisit the Cognition Plane design and refine the ML algorithms, based on integration with the vertical UCs on the SliceNet testbeds. Our goal is to provide a reference architecture for QoE-driven cognitive management of NSes.

List of authors

Company	Author
Altice Labs SA, Portugal	Rui Pedro, Guilherme Cardoso, Pedro Neves, José Cabaça, José Sousa, João Silva
Eurecom	Nasim Ferdosian
IBM Research – Haifa	Dean Lorenz, Kenneth Nagin
Orange Romania	Marius Iordache, Catalin Brezeanu
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
Ericsson Telecomunicazioni Spa (TEI)	Ciriaco Angelo
University of The West Scotland (UWS)	Qi Wang

Table of contents

List of figures

List of tables

Abbreviations

4G	Fourth Generation (mobile/cellular networks)
5G	Fifth Generation (mobile/cellular networks)
5G PPP	5G Infrastructure Public Private Partnership
AI	Artificial Intelligence
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
DB	Database
DDCM	Data-Driven Control and Management
DSP	Digital Service Provider
E2E	End-to-end
ECA	Event-Condition-Action
eMBB	Enhanced Mobile Broadband
eNB	Evolved Node B
FaaS	Function as a Service
FCAPS	Fault, Configuration, Accounting, Performance and Security
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ICMS	Inter-cloud Meta-scheduling
ILP	Integer Linear Programming
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-PoP Communication
JSON	Java Script Notation Object
KB	Knowledge Base
KPI	Key Performance Indicator
LCM	Lifecycle Management
LTE	Long Term Evolution
MAPE-K	Monitoring, Analysis, Planning and Execution governed by a Knowledge-base
ML	Machine Learning
mMTC	Massive Machine Type Communications
MQTT	Message Queuing Telemetry Transport
NFV	Network Function Virtualisation
NN	Noisy Neighbour
NS	Network Slice
NSP	Network Service Provider
NSS	Network Sub-Slice
OAI	OpenAirInterface
OSA	One Stop API
OVS	Open Virtual Switch

P&P	Plug & Play
PAP	Policy Administration Point
PDP	Policy Decision Point
PF	Policy Framework
PoC	Proof of Concept
PoP	Point of Presence
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RRM	Radio Resource Management
RTT	Round Trip Time
SDK	Software Development Kit
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
SS-O	Service Slice Orchestrator
TICK	Telegraf, InfluxDB, Chronograf, Kapacitor
TTE	Time-To-Event
UC	Use Case
UDP	User Datagram Protocol
UE	User Equipment
URLLC	Ultra Reliable Low Latency Communications
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
WP	Work Package

1 Introduction

1.1 Scope within updated architecture

This report covers the second iteration of the SliceNet Cognition Plane; as such, it builds upon the report provided for the previous iteration and expands its content to include the latest developments and integration test of SliceNet's Cognition Plane. Due to the strong dependences with the rest of SliceNet's architectural components (specially, the Monitoring and Orchestration sub-planes), the developed Cognition Plane and its components are intended to be updated as WP6 and WP7 materialize. The scope and goals of the SliceNet Cognition Plane remain as described during the first iteration of WP5: to define and prototype a ML-aid framework which enable 5G control and management systems with the capacity of QoE awareness in regards of NS provisioning and lifecycle management (LCM). As such, the proposed Cognition Plane covers the functions needed to monitor, estimate and predict relevant metrics that affect the QoE of NSes provisioned in the context of SliceNet vertical UCs as well as the functions and modules that govern runtime (re-)configurations of the underlying physical and virtual infrastructure to maintain optimal QoE levels. Having said that, Figure 1 depicts the logical SliceNet components addressed by WP5 as described in WP2 deliverables (see D2.4, Section 5) [1] and in the updated SliceNet architecture (see D8.4 [2]), putting special emphasis on highlighting for every cognitive management logical component the scope of its implementation (WP5 or WP6).

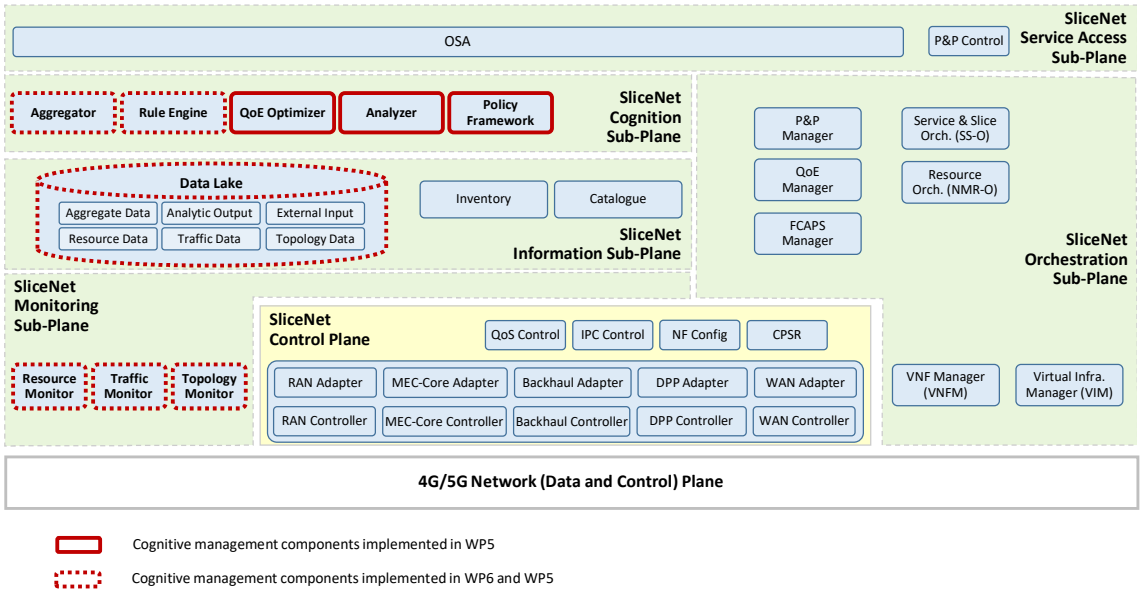

Figure 1 Logical SliceNet components addressed by WP5 within the updated architecture

Compared to the previous iteration, the main update to the SliceNet logical architecture is the emphasis on the role of the Data Lake. SliceNet Cognition Plane design developed the Data Lake as a key component to analytic driven management (as already described in the previous WP5 deliverables). This design has influenced the overall SliceNet architecture, as well as the division of effort between WP5 and WP6. SliceNet follows the same logical architecture at both the Digital Service Provider (DSP) and Network Service provider (NSP) levels, with some noteworthy differences (see Section 2 for more details). One of the implications of these differences is that some logical functions are implemented at both DSP level (in WP5) and NSP level (in WP6). The Data Lake itself is implemented using the same technologies based on the TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) stack [3] at both DSP and NSP levels, with differences on the data that is being monitored,

ingested and processed. Finally, note that some logical components are implemented as analysis and aggregation tasks that run on top of the Data Lake. For example, the different monitoring functions (resource, traffic and topology) at the DSP level are implemented utilizing the data made available by the NSP level Data Lakes. As another example, QoE sensors (as described in D5.2 [4]) are implemented through aggregation and analysis on top of the Data Lake. Lastly, the Actuation Framework has been also designed to operate on top of the DSP Data Lake, employing the data samples stored as triggers for actuations. As such, the described Data Lake approach allows for a better and cleaner separation of monitoring, analysis and actuation functions, loosely coupling them through the shared data, and enabling the possibility to easily update existing functions or add new ones (for example, to consider new cognition UCs or actuation procedures).

1.2 Document structure

In this sub-section, we detail the document structure, referring to the progression of the contents with respect to the content that was already reported during the first iteration (D5.5 [5]). In essence, the new content reflects the latest refinements on the overall Cognition Plane architecture and how the different MAPE-K components have been evolved with respect to iteration I. Moreover, the focus of the current iteration has been on the validation of the several cognition UCs defined in D5.5, for which only the design was proposed. As such, this deliverable presents extensive experimental results which demonstrate the development and integration of the several Cognition Plane components. More specifically, the document is structured as follows:

1. Section 2 provides an update of the MAPE-K loop elements and modules that have been modified following the latest refinements of both SliceNet's Cognition Plane and overall architecture (mainly the Actuation Framework), putting especial emphasis on the role of the Data Lake. Additionally, it provides a section highlighting how the Cognition Plane design is being exercised at the different roles of the overall SliceNet architecture (NSP/DSP), focusing on the interactions between the roles.
2. Section 3 describes the last developments of the analytical functions and workflows that aid on the support of the multiple UCs defined within SliceNet. To this end, concrete experimental results are shown to highlight the performance of the analytical functions as well as the integration with other Cognition Plane components, such as the Actuation Framework. The experimental results not only reflect the validation of the analytical UCs and their integration with other components of the Cognition Plane, but also the updates on their approaches to achieve the desired analysis. For example, different metrics as well as testing applications with respect to iteration I are being exercised in the current iteration by several of the reported analytical functions. The specific details for all of them are reported in their corresponding sections. Specific discussions relating the multiple ML UC with SliceNet's vertical UC are also provided.
3. Section 4 describe the last developments of the Actuation Framework. In concrete, actuators defined during the previous version have been updated while new ones have been introduced, emphasising their usage in some of the analytical cognition UCs (such as the Noisy Neighbour (NN) described in Section 3.3) and experiments performed in this regard. Moreover, we specify the policy implementation for the workflow-based actuations according to the structure being defined in D6.6 [6].
4. Section 5 documents the main interfaces that have been designed and implemented for the second iteration of the Cognition Plane, expanding on the interfaces already detailed during the first iteration.
5. Section 6 summarizes the new implemented software modules for the second iteration, providing the links for their source code.
6. Lastly, Section 7 contains the main conclusions of the deliverable.

1.2.1 Main changes from previous version of D5.6 (v1)

1. The overall document has been revamped in order to avoid redundancy of contents with respect to past iteration I already reported in D5.5 [5]. Following the same document structure as in D5.5 and D5.6 (v1), the current document only reports new contents developed within the second iteration of WP5 or content that has been updated following the latest refinements of both the Cognition Plane architecture and the overall SliceNet framework.
2. Section 3 has been enhanced with the following points:
    - A discussion of the compatibility among all analytical UCs and ML models with the general Cognition Plane and SliceNet framework architectures.
    - More details on the datasets employed for each of the analytical workflows, including their dimensionality, main features as well as their usage for the learning and training phases. In addition, more details about the employed mechanism for feature extraction, annotation and learning are also provided.
    - Finally, new sub-sections are provided to highlight the relationship of the multiple analytical/cognition UCs with SliceNet vertical UCs and 5G services.

2 Cognition Plane architecture and functional components – update

The revised overall SliceNet architecture is presented in Section 1 of this document, which is further explained in deliverable D8.4 [2]. Aside from refinements on the roles of each functional module, the relationships between them and the information flows across modules and layers, one of the main updates of the overall SliceNet architecture and more specifically of the Cognition Plane is the highlight of the Data Lake. In this regard, the Data Lake becomes the main protagonist of the Cognition Plane, not only serving as a centralized source for the information but also articulating the interactions between monitoring, analytical and actuation functions. Indeed, interactions between the Cognition Plane functional elements are done by consuming/inserting data from/to the Data Lake, simplifying operational workflows as well as enabling a more updatable architecture, in which new analysis and actuation functions can be added by properly consuming/inserting the right data from/to the Data Lake.

This cleaner design approach has driven the modifications and updates to Cognition Plane sub-systems, mainly the Actuation Framework, which for the current iteration has been re-designed to work on top of the Data Lake, employing it as the main source of stimulus for actuations. As such, the following sub-section elaborates on the new design and approach followed for the updated functional components.

2.1 Actuation Framework and vertical-informed actuators

Figure 2 depicts a schematic of the updated 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. As explained before, interactions from the analytics at the Cognition Plane are enabled thanks to the centralized Data Lake at the DSP level, which collects all data sources for them to be ready for consumption by the Actuation Framework, more specifically, the QoE Optimizer module. Thus, monitoring data related to the provisioned E2E NSes is inserted to the centralized Data Lake by the multiple monitoring functions. Such data can be then directly consumed by the QoE Optimizer to trigger an actuation or elaborated by analytical functions, which will then insert the enhanced data back to the Data Lake, to serve as stimulus for the QoE Optimizer. Additionally, the Data Lake serves as a point for interaction with the actuation systems at the NSP (the TAL Engine, as depicted in the picture), in which NSP alerts may be inserted to notify that an E2E actuation is required to overcome situations that cannot be resolved at the NSP level. We elaborate on the interaction between actuation systems at both roles (DSP and NSP) later on this section.

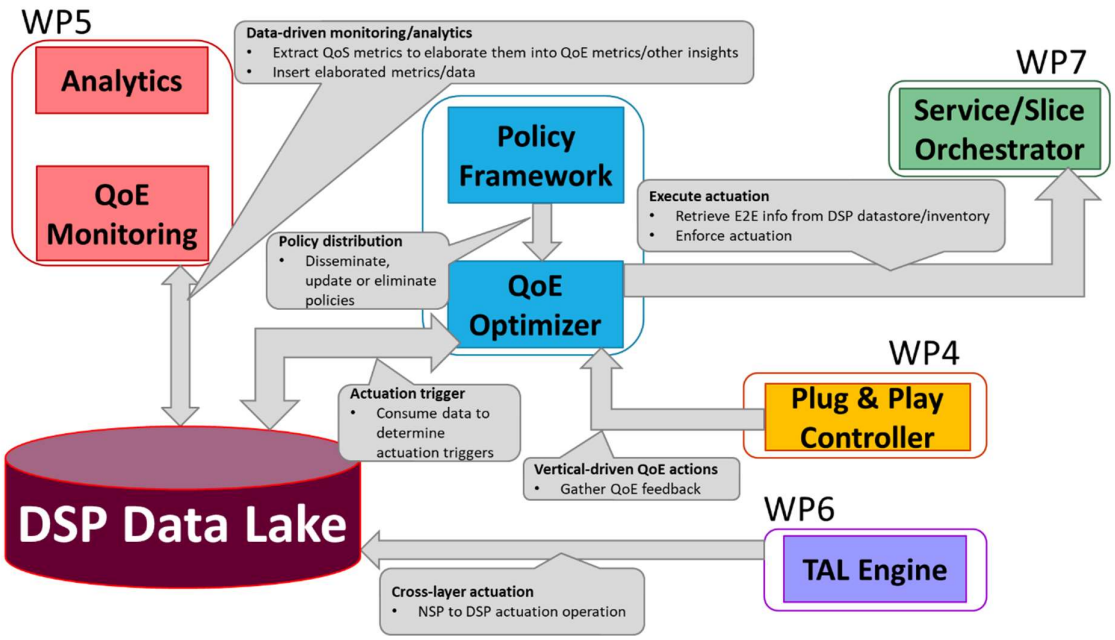

Figure 2 Schematic of the logical architecture of the Actuation Framework

Aside from the introduction of the Data Lake, which has modified and influenced the way in which the actuation loop is being executed, the refined SliceNet architecture has also influenced in the way that the Actuation Framework at the DSP level enforces the desired (re)-configurations to maintain optimal QoE levels for the deployed slices. In previous iterations, the two main enforcement points where both SliceNet Orchestration and Control Planes, with the Orchestration Plane being contacted for actuations that required modifications or new configurations at the E2E NS level while the Control Plane (CP) was contacted for actuations that only required modifications in a concrete network segment (i.e. a single NSP).

Moving away from this design, the current iteration follows a cleaner approach to this regard, in which actuators triggered from the QoE Optimizer are always enforced through the DSP orchestration system. This approach allows for a more coordinated actuation workflow, in which the QoE optimizer specifies the "what" and high-level parameters of the action, but the "how" is determined by the orchestration system, as it has the visibility of all currently instantiated NSes so optimal solutions can be applied to enforce the desired (re-)configurations. Moreover, this approach allows for a better separation of the business roles, with each one having a clear view of its own responsibilities and attributions: the DSP is in charge of coordinating actions for E2E actuations, while the NSP focus is on orchestrating the actions needed to enforce specific actions into the network sub-slices (NSSes) which have been deployed onto its infrastructure segments including Radio Access Network (RAN), Core Network (CN), Network Function Virtualisation (NFV) Points of Presence (PoPs), etc.. Given this premise, Figure 3 depicts the updated logical architecture for the QoE Optimizer module. The main modification relates to the actuation control part. In previous iteration I, the QoE Optimizer had a dedicated **Actuation Control** per main point of enforcement, named respectively **Control Plane-based Actuation** and **Orchestration-based Actuation**. Following the described approach for the current iteration, the presented updated design has only one actuation control (**Orchestration-based Actuation**) which directly speaks with the Orchestration Plane to enforce all actuations resulting from the policies being exercised as part of the actuation loop.

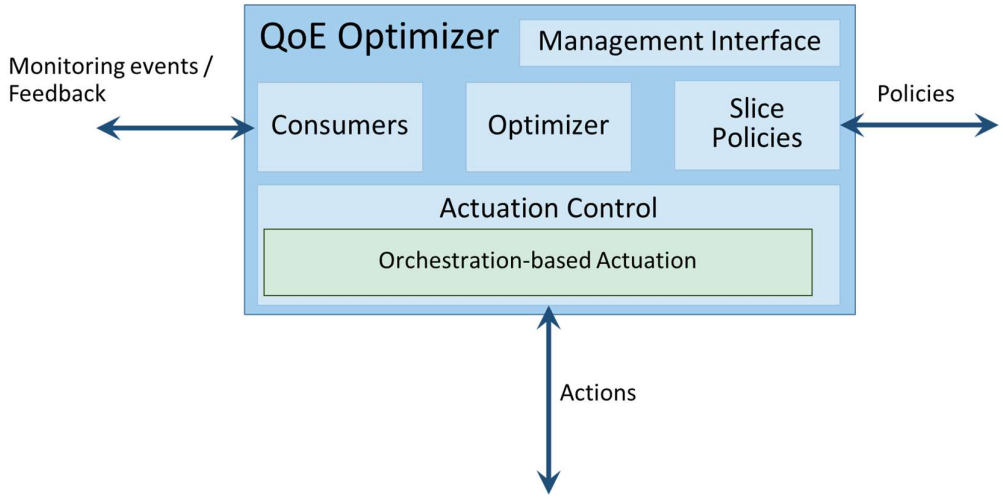

Figure 3 QoE Optimizer logical architecture

Regarding policies, the design of the Policy Framework (PF) has been maintained for the current iteration, in which policies define rules to be followed in the format of Event-Condition-Action (ECA), as explained during iteration I. Then, a Policy Administration Point (PAP) disseminates the policies to the multiple Policy Decision Points (PDPs) across the SliceNet architecture, being the QoE Optimizer one of them. Despite being no changes on the overall design and behaviour of the PF, the current iteration further evolves the instantiation of the framework in regards of the business role in which the architecture is being exercised (DSP or NSP), following the overall efforts to have a more generic system architecture that, while it shares part of the functional blocks across roles, their instantiation may differ due to the different requirements that need to be covered in them.

In this regard, the PF is designed to work at different levels and roles (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. 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 4 for the NSP scenario, where the main PDP is the Rule (TAL) Engine, being developed in the context of WP6.

On the other hand, Figure 5 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 vertical-informed Actuation Framework with QoE awareness.

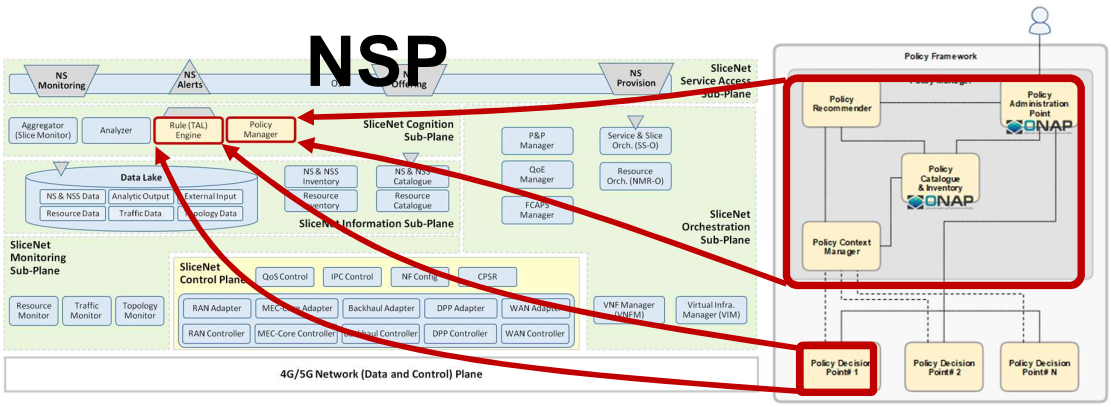

Figure 4 SliceNet PF architecture (instantiated at NSP)

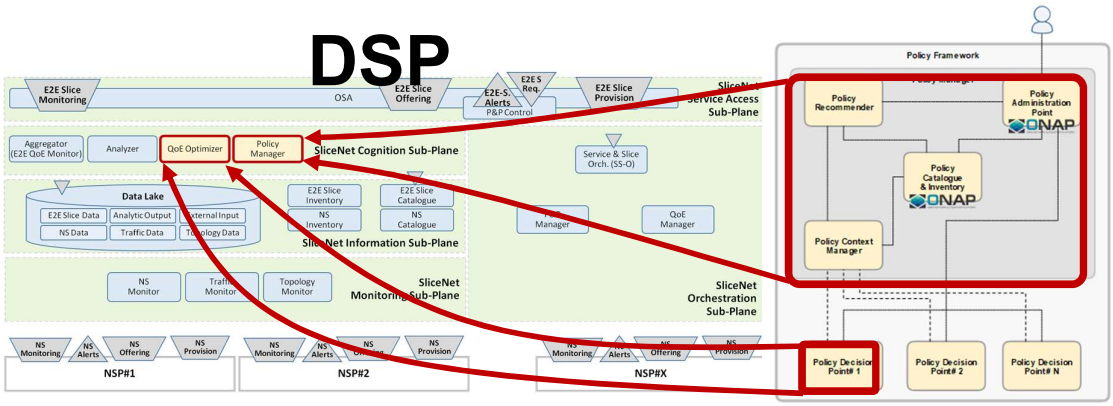

Figure 5 SliceNet PF architecture (instantiated at DSP)

2.1.1 Short/cross-entity actuation loop

In this section, we revisit the cross-entity actuation loop presented during iteration I and update the main functional elements involved in the operational workflow. Although the core of the actuation loops takes 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. The data is then persisted in a Data Lake at the NSP level, following the same reasoning as at the DSP level. Then, an instance of the TAL Engine being developed within WP6 is responsible for enforcing actions at the infrastructure via the enforcement points available at the NSP level (e.g. CP and NSP orchestrator), acting as a PDP in the context of the PF. If the detected anomaly/underperforming situation can be resolved at the NSP level, the TAL Engine 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 NSP level, which then is reflected at the DSP level (to elaborate E2E metrics/insights), resulting in a reaction from the QoE Optimizer once the relevant information has been consumed from the DSP Data Lake. (2) Otherwise, the DSP level QoE Optimizer is notified of the specific anomaly and will take the desired corrective action. In such a case, a specific event will be produced at the NSP level, which then will be collected at the DSP level and inserted at the centralized Data Lake. As such, this event will be ready for consumption by the QoE Optimizer, being another source of monitoring, which may result in an actuator trigger.

Figure 6 depicts the logical architecture of the elements involved in the cross-entity actuation loop, ranging from the MAPE-K loop elements (e.g. Monitoring, Data Lake, QoE Optimizer, PF, ...) to elements that facilitate the separation/communication between roles/entities (e.g. One Stop API (OSA) and P&P). In this regard, the presence of both DSP and NSP Data Lakes is emphasised since they are the elements that enable the information flows across entities.

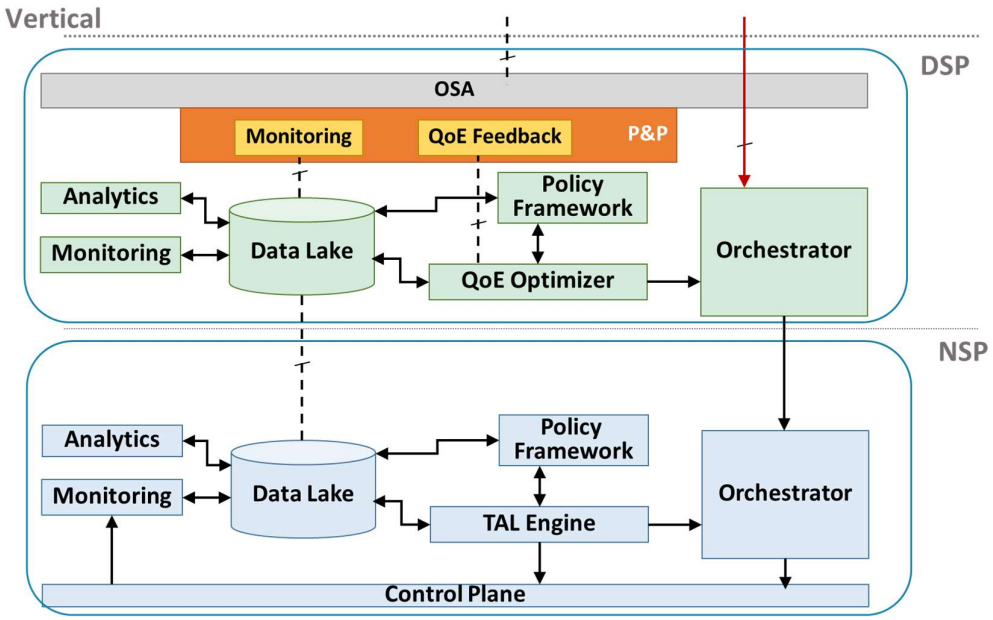

Figure 6 Logical design and architecture of the short/cross-entity actuation loop

2.2 DSP vs. NSP system architecture – cognition perspective

SliceNet's Cognition Plane covers both DSP and NSP business roles, providing concrete solutions to satisfy their needs. In this regard, the followed approach has been to define and develop a common architecture that can serve both roles by an adequate instantiation and particularization of the functional components defined at the generic Cognition Plane architecture, which have been explained in details during iteration I. Following this design principle, within this section is provided a description of the system architecture instantiation at the DSP and NSP business roles. Furthermore, since data management (ingestion/collection, preparation and transformation) and Artificial Intelligence (AI)-based procedures (e.g. training and real-time insights/predictions) are tightly coupled, it is also contextualized the responsibilities of WP5 and WP6 in the DSP and NSP architecture instantiations, highlighting what components are designed and implemented under the scope of each WP. Finally, in order to provide a functional perspective of the data management and AI areas of the system architecture, a very high-level cognition workflow is described, including monitoring, data management, AI-based predictions, policy-driven decision-making and actuation.

2.2.1 NSP architecture instantiation

The NSP architecture perspective is given in Figure 7. The several planes (data, control and management planes) and sub-planes (monitoring, information, cognition, orchestration, and service access) are represented, as well as the interfaces exposed by the SliceNet system towards outside components (mainly, NSes LCM interfaces – NS Offering, NS Provision, NS Monitoring and NS Alerts). Detailed information about the planes, sub-planes, interfaces and components is given in deliverable D8.4 [2]. Nevertheless, for overall contextualization and critical relevance to cognitive-based architectures, we highlight three key aspects of the revised system architecture:

1. **Data-Centric**: this is one of the key aspects of the architecture; cognition-based procedures are dependent on data and how the data is prepared and transformed before it actually reaches the AI/ML algorithms. A Data Lake approach is followed, located within the SliceNet Information Sub-Plane, which is transversal to all the architecture components, enabling any sub-plane component to read, process/transform and write data from/to the Data Lake. The data transformation processes can be used for ML procedures (training and/or prediction), data visualization tools, etc.

The Data Lake is being designed and implemented in WP6 utilizing the TICK stack, with D6.6 reporting the first efforts in these regards [6]. Simple aggregations, filtering and transformations can be done directly on the TICK stack, thus, can be configured through the PF as slice-specific data-processing functions. More complex processing, including inferring ML models, use the TICK stack only to ingest data from the Data Lake and to provide their output back.

2. **Cognition-enhanced**: several types of ML algorithms are available (neural networks, decision trees, etc.) for training and creating models able to perform real-time predictions and/or recommendations according to the vertical UCs. SliceNet cognition can be seen as a service able to create and deploy ML-based models for any type of UC (e.g. alarms prediction, KPIs forecast, KPIs performance degradation prediction, configurations recommendation, etc.). The Analyser component is the key responsible for implementing the SliceNet ML procedures and is located in the SliceNet Cognition Plane.
3. **Policy-driven**: finally, the architecture components behaviour is highly configurable through rules/policies, as much as possible decoupled from the components internal logic, allowing the SliceNet system operators to easily change the behaviour of the system. The Policy Manager located at the SliceNet Cognition Plane, together with the policy decision and enforcement points, which can be located in several architecture locations (e.g. management, control and data planes), are responsible for the policy-driven behaviour of the system.

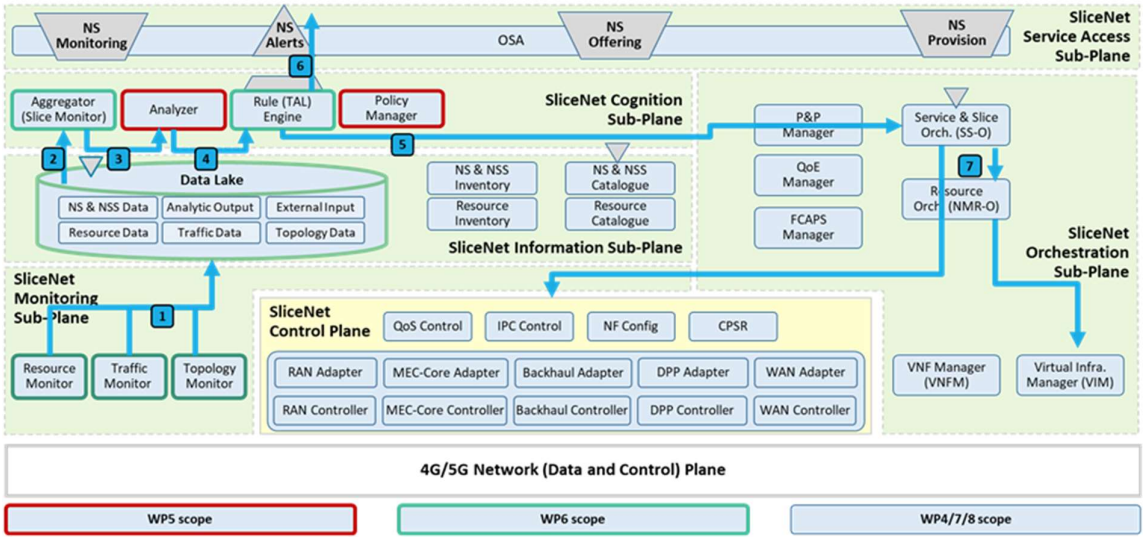

Figure 7 SliceNet system architecture instantiated at the NSP

Besides the logical architectural aspects, Figure 7 also includes a (very high-level) MAPE/cognition loop workflow (in blue arrow), highlighting the role of each component. As for the architectural components' description, also detailed information about the several SliceNet workflows is available in D8.4. Table 1 describes the several steps implemented in the MAPE/cognition workflow.

Table 1 MAPE/cognition workflow (at NSP) steps description

Step	Description
1	Data ingestion and persistence in the Data Lake through the several SliceNet monitors (**resource, traffic and topology**).
2	**Aggregator** reads data from the Data Lake, performs the required preparation/transformation/aggregation procedures and persists the result again on the Data Lake (NS data). Besides persisting the data, the Aggregator also streams the data to be consumed by other applications (e.g. Analyser).

3	**Analyser** (e.g. AI/ML prediction model) consumes the transformed data and runs the model to produce the insights/predictions. The produced insights/predictions are stored in the Data Lake and streamed for real-time consumption and reaction.
4	**Rule (TAL) Engine** consumes the prediction **E**vent (e.g. network fault prediction), checks the policies **C**onditions (e.g. available bandwidth) and decides for the appropriated **A**ction (e.g. increase slice bandwidth) to be applied.
5	When the NSP is able to mitigate the predicted fault through internal procedures, the action plan is sent to the **SliceNet orchestrator**.
6	If the NSP is not able to mitigate the predicted fault internally, a notification/alert may be sent towards the **DSP** to react (as exemplified in Section 2.1.1).
7	**SliceNet Control Plane** and/or the **Orchestrator** receive the actions request and enforce them on the network data plane.

2.2.2 DSP architecture instantiation

The architecture principles adopted at the NSP are identical to the ones adopted at the DSP (data-centric, cognition-enhanced and policy-driven). The main difference between the DSP and NSP is the network related resources – the NSP is the one managing the network resources that compose the offered NS. On the other hand, the DSP is the business entity composing the E2E NS – using the several NSSes at his disposal (offered by each one of the underlying NSPs) – and offering it to the verticals. Therefore, with respect to the SliceNet architecture planes, since the DSP is not managing the NS/NSS per se, it does not need to instantiate the SliceNet Network (data and control) Plane, as well as the SliceNet CP. Furthermore, only part of the SliceNet Orchestration system needs to be instantiated at the DSP – the part responsible for E2E slice orchestration (and not the resources and NSSes orchestration). The remaining management sub-planes (Monitoring, Information, Cognition and Service Access) are similar to the ones instantiated at the NSP. The only difference is the component that is deciding the policies at the DSP side – QoE Optimizer instead of the Rule (TAL) Engine at the NSP side. Figure 8 illustrates the SliceNet system architecture instantiated at the DSP.

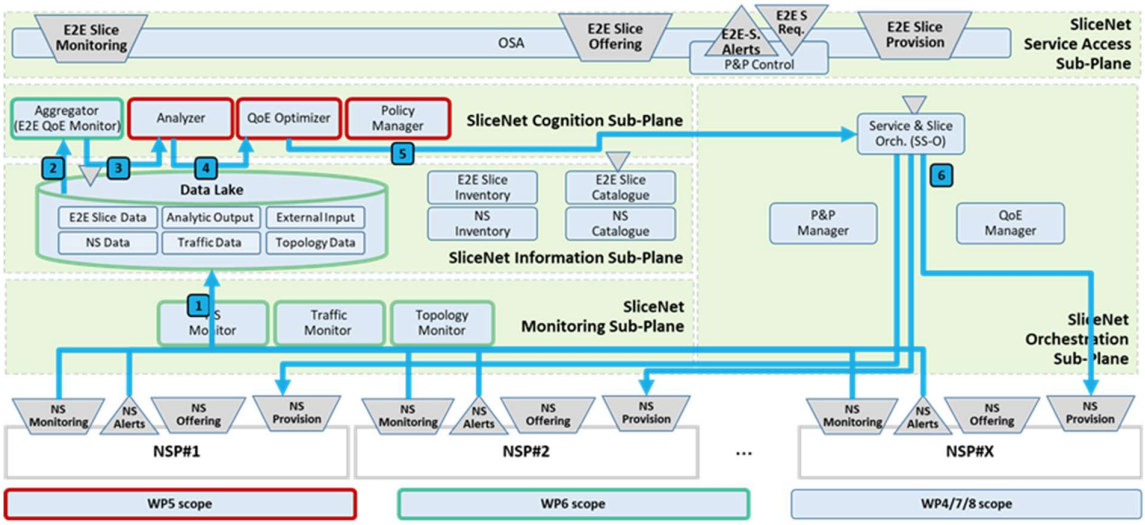

Figure 8 SliceNet system architecture instantiated at the DSP

In terms of responsibilities between WP5 and WP6, the approach is similar to the one followed at the NSP side. WP6 is responsible for the **Monitoring** part (Monitors, Data Lake and Aggregator), whereas the **Analysis** (Analyser) and **Planning** (Policy Manager and QoE Optimizer) parts are on the WP5 scope. Finally, the **Actions** (i.e. the final enforcement point) are handled by WP7.

Figure 8 also illustrates a cognition workflow at the DSP level. The workflow behaviour is very similar to the one explained at the NSP side – the main differences are the following:

- The monitors collect NSes information (and not resource-level information);
- Aggregator produces E2E NS data information;
- Analyser creates E2E NS level insights;
- QoE Optimizer and Policy Manager decide on the best mitigation strategies at the E2E NS level (e.g. recomposing the E2E slice, reconfiguring specific NS parameters, etc.);
- Actuations are enforced by the Slice Orchestrator at NS level through specific requests to the NSPs northbound Application Programming Interfaces (APIs).

3 Analytic workflows

3.1 Scope and preliminary implementation

The scope of the implemented analytical workflows, their details and purpose as well as their scenarios have been already described in iteration I of WP5 (see D5.5 [5]), with Table 2 summarizing the developed analytical workflows. As such, in this section we provide the latest developments of the analytical workflows, providing the implementation details and results for all workflows as well as updates to their design (if applicable) that have happened during the execution of iteration 2 of WP5.

Table 2 Summary of analytic 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 Quality of Service (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)

All the presented analytical workflows and cognition UCs are supported thanks to the execution of analytical and monitoring functions that are being exercised across the whole WP5 architecture as well as the overall SliceNet framework. Indeed, the multiple presented ML models may be exercised following a Function as a Service (FaaS) approach, in which the developed models are being executed thanks to the capabilities of the Analyzer module, which encloses the software framework to support the runtime execution and inference of the models (for example, through OpenWhisk). Then, the necessary data to produce the analysis outputs by the different ML models is provided thanks to the centralized Data Lake. As said before, the Data Lake approach allows loosely coupling any analytical function with any monitoring function, for which the Data Lake is central. As such, the different monitoring mechanisms/technologies within the SliceNet architecture (e.g. Prometheus, Skydive) pour their data onto the Data Lake, after normalization and aggregation processes, for the later consumption of the analytical functions. This Data Lake approach is the one that provides compatibility across all the developed analytical ML models, since the only point of contact across them is the Data Lake, hence, they can run independently of each other. Moreover, this approach allows for the compatibility of all ML models with the Actuation Framework at DSP level or the FCAPS management at NSP level. In this case, it is only necessary to implement the data consumption of the different ML

models by the Actuation Framework (i.e. the QoE Optimizer) and the FCAPS management (i.e. the Rule/TAL) engine. Then the necessary actuations will be carried out at the corresponding level (DSP or NSP) by the corresponding module in a transparent way. Having said this, the following sub-section elaborates the considered analytical workflows and their evolution respect iteration I.

3.2 Reliable RAN slicing using NSP alarm data

In this section, we present the design of components as well as the specific validation tests and experimental results extracted for the analytical workflow of reliable RAN slicing. In a nutshell, as presented during iteration I, the goal of the workflows is to predict potential RAN failures given the ingestion of NSP alarm data by mean of a ML model enclosed in the Analyser context. The predictions of the module serve as basis to either initial RAN slice provisioning with reliability guarantees or as to determine in the runtime phase of slices if a concrete NSP meets desired reliability levels.

3.2.1 Design of components

As in all analytical workflows described across this section, the key cognition element on SliceNet architecture is the Analyser or analytical functions. This component does not work as the typical process we are used to see, where each run usually is linear and deterministic. There are a few key differences that make this component unique. In the first place, it does not run in very well-defined internal states. It is continuously self-updating and adapting as new inputs are read. For example, the way the model in the Analyser operates means that even when the NS is recently deployed, and every component is new and fully working, the model is subject to probability and uncertainty, meaning that, even if very low, there is always a plausible chance of failure. The model adapts incrementally so each moment has different evaluation weights of the last assessment. Therefore, each output is the result of a unique computation - the internal model state is in constant mutation. Secondly, and because of this constant change, the architecture of the components that connect to the Analyser output must take incertitude into account. Despite the fact that some thresholds can be tuned, it is not possible to configure the Analyser to output an imminent fault prediction when it is 100% sure (we remind the reader that this analytical workflow is devoted to reliable RAN slicing given the prediction of failures at the RAN segment). Even if infinitesimal low, there is always a probability of a false positive. As a consequence, the model outputs new current situation prediction on every event and not only when there is an imminent failure, even if the output itself is a very small failure probability – meaning that everything is “healthy”. This ambiguity comes with some concerns typical to the ML domain: the model performance on false positive and false negative predictions. The Actuation Framework must deal with this and should accommodate the fact that the Analyser will eventually at some point in time misbehave, and may not be suitable as the only source to make drastic actions.

To overcome such a challenge, there are a few data manipulation and transformation in order to have a suitable environment to run the prediction model. The steps involved include the following:

1. Read the data from the Aggregator.
2. Convert it to an internal format capable of interpretation.
3. Validate the data.
4. Apply processes, such as unroll lists, flatten data, reduction, normalizations, etc. to facilitate posterior data manipulation.

Figure 9 shows a representation of how the alarms are events that happen through time. Each alarm event is captured by the pipeline into multiple sliding windows depending on their range. Then, each one is processed (data transformation and wrangling), where the pattern is extracted and evaluated by the trained ML model.

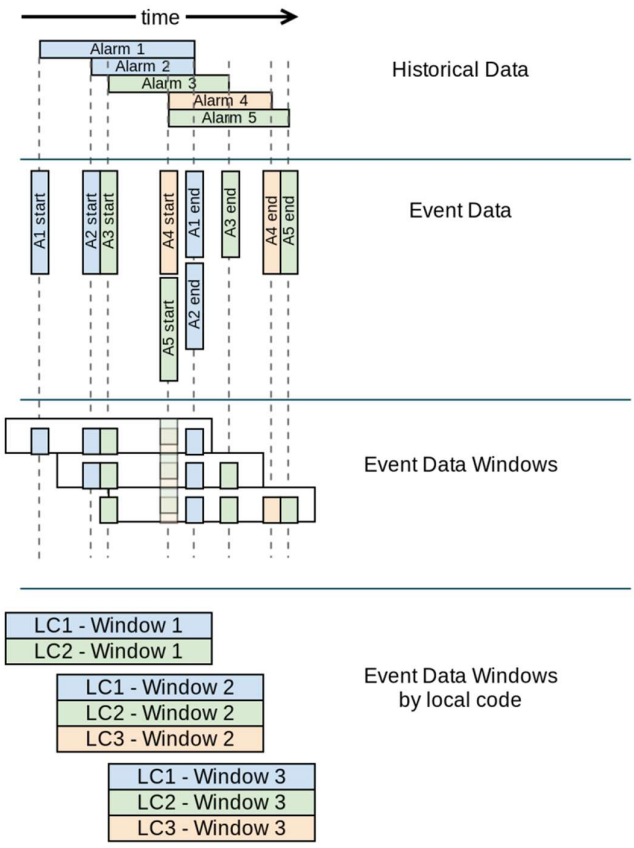

Figure 9 Converting the occurrence of alarms in time to windows that capture those alarm events

The whole process works as a black box to other SliceNet's Cognition Plane components. The Analyser consumes the Aggregator output and then sends the result as messages in a format compatible with the PF input. The advantage is that the Analyser component can be replaced or reinitialized with different internal purposes/models/approaches to the data input without modifying the SliceNet architecture and messages formats. In this scenario, execution of the conversion process from historical data up to into an *Event Data Windows* that lives and dies within the Analyser module. Samples are then segmented by local code where vocabulary files are extracted and then run against the model for both possible situations, either for training or for prediction.

3.2.2 Implementation details

At first sight, the Analyser module has one input and one output. It connects to the Aggregator component from the Monitoring Sub-Plane and outputs to the Data Lake at DSP level. Apache Kafka message broker is used for both input and output. For the input, it expects messages from the Aggregator, which pre-processes and sends the Altice Labs MEO dataset (previously described in D5.5 [5]) in such a way that the subsequent Analyser pipeline can further process it. To meet the expectations, we conducted several technology experimentations to find which one (if any) meets the SliceNet requirements. The results - explained in detail later on - show that there are huge differences in performance and responsiveness if this pipeline implementation is not calibrated for the SliceNet architecture. The macroscopic Analyser process, however, is not just the data-processing pipeline. It requires all the connectors to support the whole Analyser workflow:

1. Read messages from Aggregator in the Apache Kafka message broker.
2. Open and validate the payload, and forward the core data to the data-pipeline.

3. Process the data in the data-pipeline.
4. Collect the prediction result from the data-pipeline.
5. Encapsulate and context the result to the Data Lake.

With this in mind, the Analyser needs a few internal components that join E2E data with the added process of combining two completely different domains: the aggregated data of the network up to the "decision" results to the Data Lake for consumption by the Actuation Framework. The performance of cognition itself in the context of alarm data and imminent fault prediction is described in Section 3.2.3.

3.2.2.1 Machine learning technologies comparison

In the following, we present a comparison between multiple ML technologies and technique that will help on determining the most suitable approach to face the learning procedures required for the reliable RAN slicing scenario, enclosed in the Smart Grid SliceNet vertical UC. To this end, in the following we present the employed methodology as well as the results from the comparison among the tested technologies.

3.2.2.1.1 Methodology

The technologies that are compared are on one side Tensorflow [7] and on the other side PyTorch [8]. In order to perform the comparison, a neural network with the same structure is employed in both cases [9], with Table 3 detailing the concrete parameters.

Table 3 Summary of employed neural networks parameters

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, None, 53)	22684
dense (Dense)	(None, None, 26)	1404
dense_1 (Dense)	(None, None, 53)	1431

The benchmark vector includes: dataset disk read Input/Output (I/O) performance, model training elapsed time, resource usage during training (Central processing unit (CPU)%, Memory%, Graphical Processing Unit (GPU)%, GPU memory usage and GPU power draw). In order to achieve this, a small tool called *BenchmarkUtils* has been developed [10]. It comes in the format of a python notebook and makes use of a mix of bash scripts and python code to measure CPU, memory and GPU usage. The tool works in a simple way: it launches two measuring scripts; one for all GPU counters and another for CPU and memory utilization for a given *Jupyter* notebook kernel identifier. This monitoring is done asynchronously and at a configurable sample rate. Then, measurements are written to their own unique files until the monitoring is stopped. When stopped, the tool will parse those files, delete them and provide the data at the application level.

3.2.2.1.2 Computational platform

In this section, we describe the characteristics of the computational platform employed for the ML technologies comparison test. Table 4 provides a summary of the main characteristics of the physical host in which the test has been performed.

Table 4 Summary of the characteristic of the ML performance test platform

Resource	Characteristics
CPU	8x Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz
RAM	64GB
GPU	1x Nvidia Tesla P100 16GB
Hard Disk Drive	300GB 10000 RPM SAS 6Gb/s

3.2.2.1.3 Dataset

In order to perform the comparison, both tested technologies have to be fed with the same dataset. This sub-section describes the data set employed for the test. The employed dataset has three dimensions, being the number of windows (i.e. windows of data samples in the set), sequences per window and features per sequence. With this structure in mind, the dataset used is based on 928 alarm windows, each window containing 1286 alarms and 53 feature columns that represent the one-hot encoding over the *localcode* field aggregation (first 2 digits). Each window has a single label, representing the *localcode* field aggregation where the fault is located.

To better highlight the structure of the dataset, in the follow Figure 10 we depict an example of a file containing the full data associated to a window.

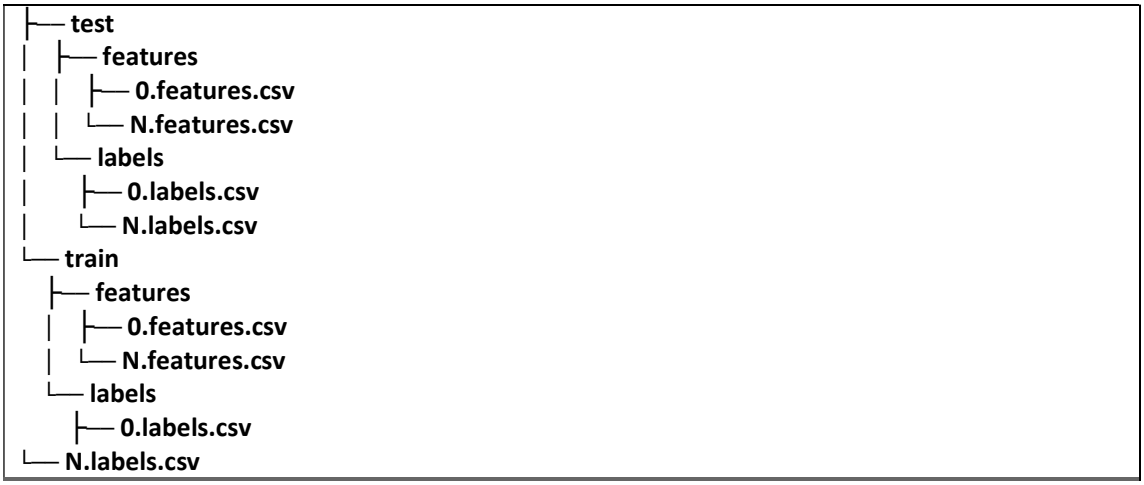

Figure 10 Example of a file containing the data from a window

The train/test set split is 75% and windows order is retained. Consequently, the sets take the following shape:

- Features set → 928 x 1286 x 53
  - Train → 696 x 1286 x 53
  - Test → 232 x 1286 x 53
- Labels set → 928 x 1 x 53
  - Train → 696 x 1 x 53
  - Test → 232 x 1 x 53

Lastly, in order to further explore the data dimensionality impact on training and resource usage, a new enlarged dataset version was created. This version only enlarged the training sets, both features and labels, to 30 times their size and has the following shape:

- Features set → 21112 x 1286 x 53
  - Train → 20880 x 1286 x 53
  - Test → 232 x 1286 x 53
- Labels set → 21112 x 1 x 53
  - Train → 20880 x 1 x 53
  - Test → 232 x 1 x 53

Table 5 provide a summary of the two datasets employed as well as the characteristics of the techniques employed for the training of the ML model, the extraction of the features and the annotation of the sets. Table 6 provides a summary of the main features of the dataset, which consist on a vector that contains the relevant information of the alarm. In this case, the main feature relates

to the location of the alarm, which is stated by a code indicating the geographical location of the alarm. The exact details of the alarm are then found in the rest of fields of the alarm's vector.

Table 5 Summary of employed data and learning for the Reliable RAN Slicing use case

Number of samples	Features per sample	Training /test data splitting	Feature selection/ extraction	Type of learning	Algorithm for training	Source of data	Annotation
928	53	75/25 %	Sequences of one-hot encoding	Super-vised	Recurrent Neural Networks	Simulated following MEO datasets	Alarm location
21112	53	98,9/1,1 %	Sequences of one-hot encoding	Super-vised	Recurrent Neural Networks	Simulated following MEO datasets	Alarm location

Table 6 Summary of the features for the Reliable RAN Slicing use case

Label of the feature	Description	Data type	Values/range
geo_agg_code	Location of the occurring alarms	One-hot vector	[0..52]

3.2.2.1.4 Benchmark results

Having described the employed hardware and data sets for the comparison of the studied ML technologies, in the following we present the obtained results for multiple facets of the comparison.

3.2.2.1.4.1 Dataset read I/O performance

Figure 11 below showcases the read I/O times when reading the data set with different batch sizes. The time spent reading the dataset (disk I/O) does not seem to be affected by the configured batch size. This does not mean that there are not more efficient ways to read the dataset. However, for this case in particular, where the dataset iterator code is the same, for both PyTorch and Tensorflow, the I/O figures remain stable. Thus, it can be concluded that, for this particular benchmark, the I/O times affect all runs in equal matter.

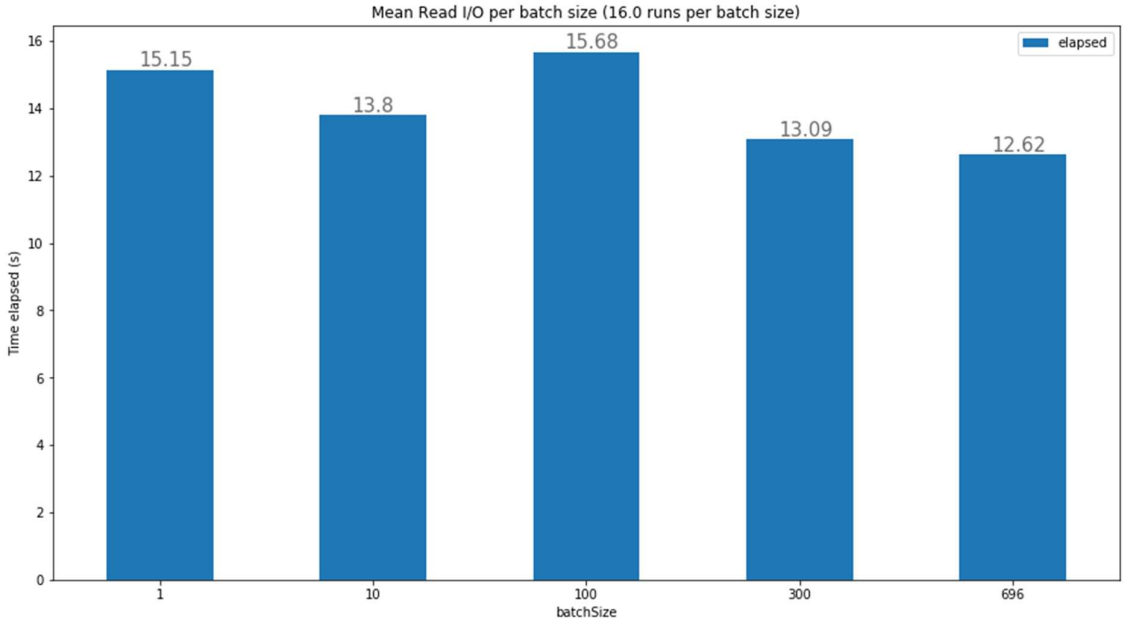

Figure 11 Mean read I/O per batch size

3.2.2.1.4.2 Training times

The following Figure 12 contains two different charts: the first row depicts the mean training per epoch (10 epochs were used per training session) and the second row shows the GPU speedup ratio, relative to CPU for the first row charts. Notice that the last column, highlighted with a red box, uses the enlarged data set in order to explore a scenario where the whole dataset does not fit into GPU memory.

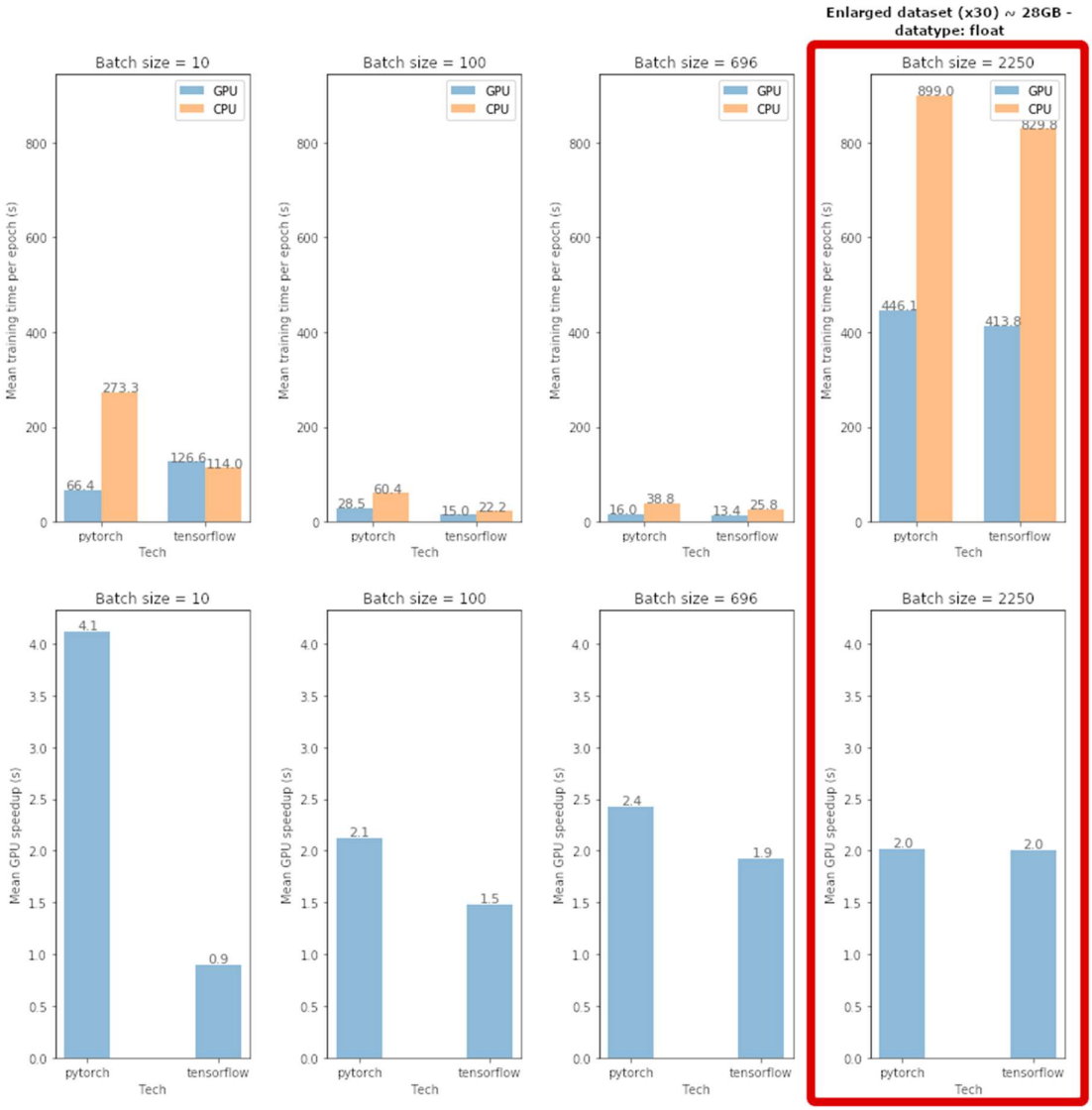

Figure 12 Mean training per epoch (top) and GPU speedup ratio (bottom)

Analysing the obtained result, it can be concluded that small batch sizes have a negative impact on training times, where PyTorch seems to do better in those cases. Moreover, for the employed neural network set up, the GPU speedup seems to stay stable at a ratio of two. This means that GPU makes training faster, as one would expect. Lastly, it can be appreciated how there are no large time differences between technologies. However, Tensorflow is slightly faster than PyTorch for the tested scenarios.

3.2.2.1.4.3 Resource usage during training

Having seen how these technologies perform when training, with different batch sizes, it is important to also take a look into their resource usage. For this matter, Figure 13 below uses the exact same layout to show three different metrics: firstly, the mean CPU usage; secondly, mean GPU usage; and lastly, the mean GPU memory used. Please notice that the last column, highlighted with a red box, uses the enlarged data set in order to explore a scenario where the whole dataset does not fit into GPU memory. Moreover, note that all technologies where configured to allow GPU memory growth, meaning that during training GPU memory was allocated as it was needed, instead of pre-allocating it.

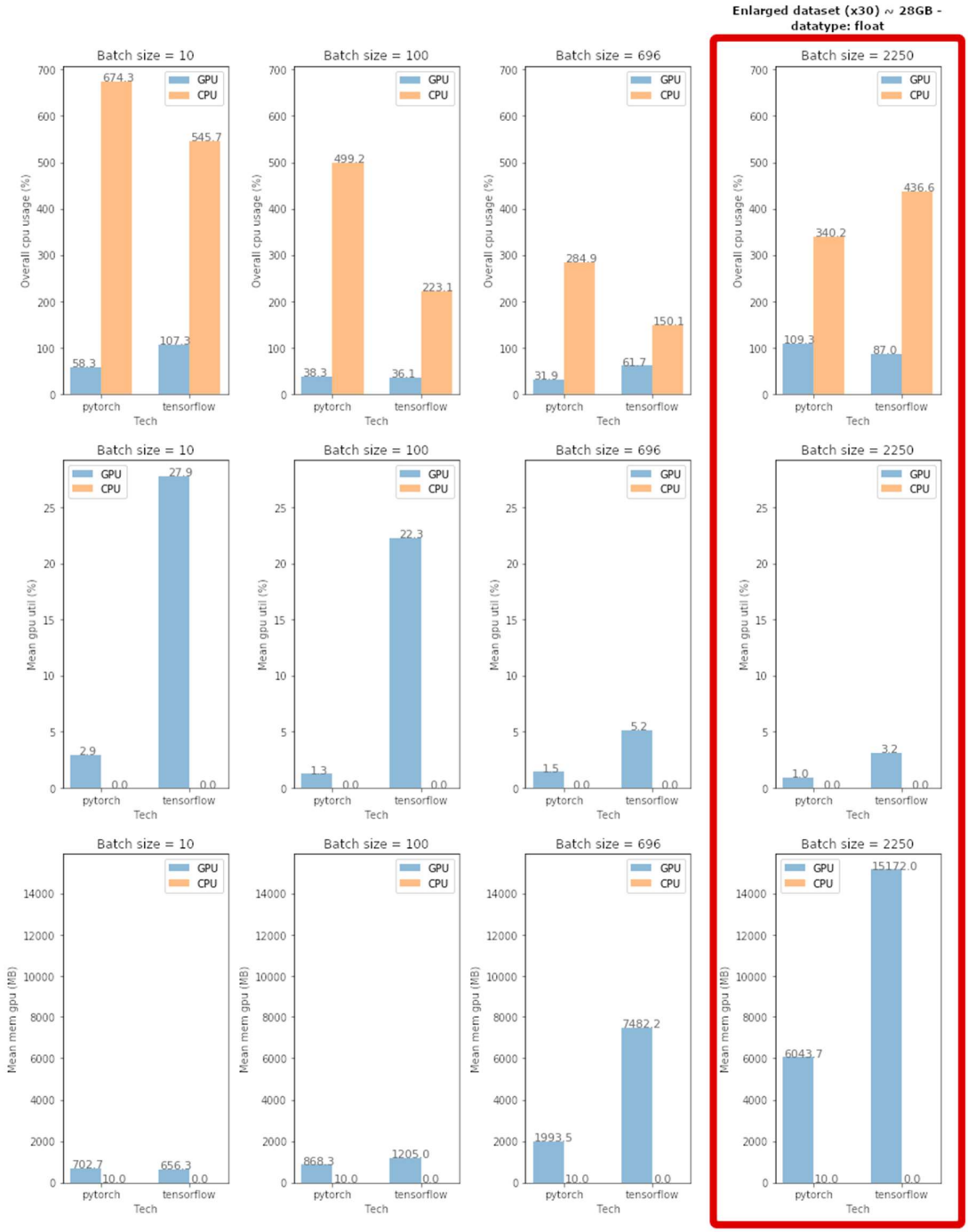

Figure 13 Mean CPU usage (top), mean GPU usage (middle) and mean GPU memory usage (bottom)

As shown in the CPU figures first, Tensorflow tends to use less CPU when training on CPU and more CPU when training on GPU, when compared to PyTorch performance. Note that this is only valid when using the normal dataset, since this trend inverts when using the enlarged dataset. The observed trend in itself does not allow for a sound conclusion, however, when paired with the previous charts, we can see that Tensorflow manages to perform the task faster while using less CPU. This particular behaviour only occurs on CPU usage, when we consider the second and third charts, we see that Tensorflow uses more GPU resources (core utilization and memory). This might be an indicator of two different situations: PyTorch does not fully leverage GPU resources, a point that can be made since training took

longer than Tensorflow runs; or, Tensorflow is not as GPU optimized, for the tested neural networks network, however it manages to use more GPU resources to complete the task faster.

One last behaviour that can be observed from these figures is that, for all technologies, when training on GPU, CPU is still used. Although, in a small percentage, the CPU is still used to perform data transfers to the GPU, alongside other tasks possibly related to either training process signalling/control.

3.2.2.1.4.4 Resource usage during training per session

Taking a deeper dive into what resources usage looks like for each training session performed on this benchmark, Figure 14 below shows the main measurements considered: CPU, CPU memory, GPU, GPU power draw and GPU memory utilization. All obtained values where normalized to fit a 0 to 100 scale, nevertheless be aware that CPU utilization varies from 0 to 800% (8 CPU cores where available). The chart layout is similar to previous ones, batch size varies through columns; however, row organization is a bit different. Each row can be grouped into two (i.e. first and second rows make a group, third and fourth make another, and so on), where each group refers to a single technology with one row containing train session on CPU and the other on GPU. This allows us to better visually compare CPU against GPU values. Notice that the last column, highlighted with a red box, uses the enlarged data set in order to explore a scenario where the whole dataset does not fit into GPU memory.

This chart group, while a bit complex, showcases how technologies make use of the GPU when training. It seems it is used in spikes and not constantly, which makes sense since the most time expensive operation is transferring data into the GPU. This is where it becomes clear how important is to be able to optimize the batch size, so that we can make better use of the available GPU memory, while minimizing transfer I/O. Another surprising scenario seen here is that, during training on CPU, less CPU is used when the batch size increases. This might be another indication that the most expensive operation is data transfer and not the actual calculations. One last conclusion that can be drawn from this data is that GPU definitively allows for faster training times, although only twice as faster, for this particular network.

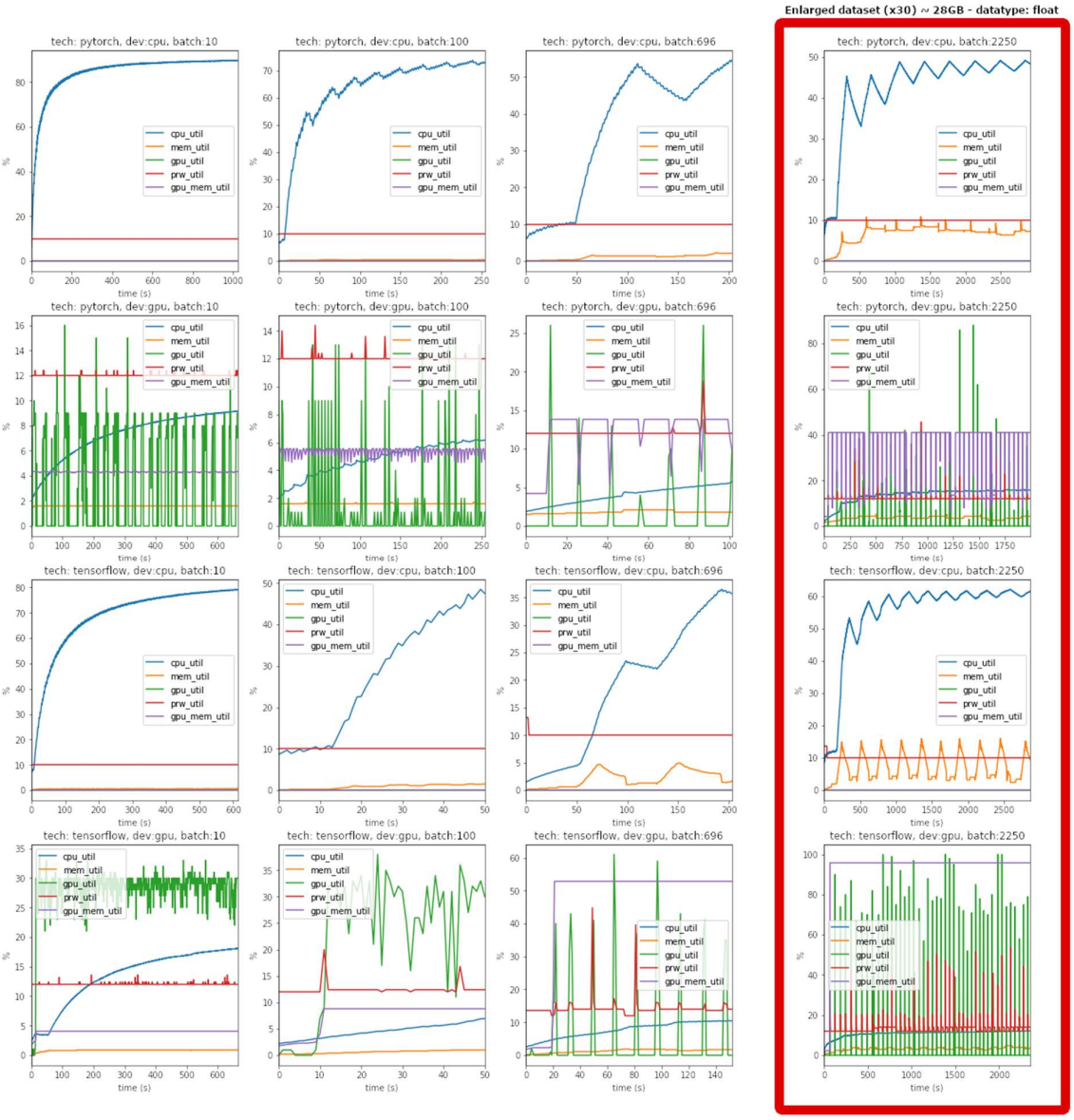

Figure 14 Mean CPU, CPU memory, GPU, GPU power draw and GPU memory utilization

3.2.2.2 ML data ingestion and modelling

As described in Section 3.2.1 for the design of the components, the data is processed through a pipeline where it undergoes through several transformations. To implement these transformations, we are using Apache Flink from the Apache Software Foundation [11], an open-source stream-processing framework, as a baseline to build the data pipeline.

Regarding the input, the data processed is expected to be a sequence of events for the equipment that makes the network, while the output are the same events structured in sliding windows. With Apache Flink the data is seamlessly integrated in the processes implemented by defining Kafka topics as the input, the streaming API provided allows for the real time processing of the data minimizing the response of the system. This particular data ingestion pipeline is divided several steps, where each one feeds the next ones:

1. **Clean-up, validation and enrichment**: Defines a stable output format that the following steps can follow. If a data source changes or is added, this step should be adapted to cope with those changes.
2. **Aggregation and segmentation**: Here the data is grouped according to a given configuration. For our predictive analysis, we grouped the data in sliding windows, segmenting them by the location of the equipment events.
3. **Classification**: Depending on the UC presented, this step allows to flag a given entry for pattern detection.

For the purpose of predicting issues in the network, there are three points that must be considered to understand how the data is processed:

- **Training/Mining**: Dependent on the model, this requires that a given historic of events is provided, usually grouped and classified according with the target of the prediction. Depending on the environment where the prediction is done this process might only be seldom needed, requiring the re-processing of stored historical data.
- **Online prediction**: This requires that the incoming data is streamed to a trained model. The result must be then be available to an actor that uses this information to make suitable decisions. The data should be processed as it arrives.
- **Monitoring**: A model trained/tuned for a historical data set risks being out of scope when the environment changes. To understand when this happens a monitoring process must be in place, otherwise the provided predictions might be misleading and cause more harm than good.

In order to test de data ingestion process, a dataset sample has been employed, following the same approach as described in previous sections regarding the dataset employed for the ML model training and testing. In this regard, Table 7 and Table 8 summarize the main characteristics of the employed dataset and techniques for its treatment as well as the main features present in the dataset samples, respectively. Note that in this case the employed data comes from the real datasets provided by MEO.

Table 7 Summary of the data size and the learning/training characteristics for data ingestion

Number of samples	Features per sample	Training /test data splitting	Feature selection/ extraction	Type of learning	Algorithm for training	Source of data	Annotation
1284400	511	25/75 %	Sequences of one-hot encoding; Sine/ cosine transforma -tion	Super-vised	Recurrent Neural Networks	Real external datasets provided by MEO	Censoring flag; Time To Event

Table 8 Summary of the main features for data ingestion

Label of the feature	Description	Data type	Values/range
problem_class	Specific problem of the alarm	One-hot vector	[0..510]
entity_technology	Technology of the alarm equipment	One-hot vector	[0..11]
dayhour	Day hour (sine, cosine)	float, float	[0..1], [0..1]
weekday	Week day (sine, cosine)	float, float	[0..1], [0..1]

3.2.3 Results

Having analysed the multiple ML technologies, this section is devoted to the evaluation of the ML model for the reliable RAN slicing, that is, the prediction of faults given the MEO dataset to infer the reliability (thus, the faults) of an NSP in terms of the RAN technology that it provides.

3.2.3.1 Model Evaluation

The defined ML model has been trained against a subset of the historical data of alarm instances provided by MEO operator. The train of the model is a supervised one, this means that the data samples must be classified with a target. With the purpose of working out the feasibility of predicting a single network issue for the next hours, we grouped the events in sliding windows of 6 hours, splitting those in half, thus having windows of observations of 3 hours, with the prediction for the next 3 hours as the target of the model. After being processed by the pipeline this equated to 12844 batches of 100 windows for a total of 1284549 sequences of 180 minutes sized windows representing 12277944 alarm instances.

The neural network output provides a censoring value for a given sequence of events provided as input. A censored observation means that the event was not found in the data observed, it might occur in the future, but we have no data regarding that. In the case of the data used, this value describes if the second half of our 6 hours window is censored or not.

This value is a ratio that, given a threshold, serves as a classifier. This threshold for censored or eventful window can be tuned to fit the UC, trading the model ability between correctly guessing event occurrences against correctly guessing the lack of the event. Canonically, a threshold of 0.5 (the middle value of the provided target labels) should provide an acceptable result when classifying the censoring of a window of events. After training for 3000 batches of the available 12844 (roughly 25% of the dataset) we proceeded to the evaluation of the predictions in the entire dataset. This evaluation was done for several threshold values.

From the results obtained, the following tables and figures summarize the ones that provide the most interesting insights.

Table 9 Obtained canonical score with a threshold offset of 0.0

Real data			
Censored	Eventful		
1272669	2000	Censored	Predicted Data
941	8790	Eventful	

Table 9 shows that from the existing 10790 windows with events, 2000 were missed by the predictive model and in the predictions of the event occurring in the next 3 hours, 941 of 9731 were false positives.

Table 10 Obtained Maximizing F1 score with a threshold offset of -0.2

Real data			
Censored	Eventful		
1272079	848	Censored	Predicted Data
1531	9942	Eventful	

Table 10 shows that as we reduce the offset, we also reduced the events missed by the predictive model, however that comes at the cost of increasing the ratio of false positives to 1531 of 10473.

Table 11 Reduction of false positives with a threshold offset of 0.3

Real data			
Censored	Eventful		
1273464	5258	Censored	Predicted Data
146	5532	Eventful	

By increasing the offset, we now apply the opposite effect (Table 11), this case would be suitable if dealing with false positives was costly. The value chosen allows for the prediction of roughly half of the existing events, but has a much lower false positive rate with 146 of 5678. With this, it becomes obvious that how the model is selected and tuned depends on the UC in question. If the process done by the Actuation Framework to deal with the events imposes high costs, we might want to avoid sending false positives, while if missing an event has a greater cost, we might want to tune our model to respect that. Plotting the distribution of the censoring ratio for when the target event occurs, we can perceive the ability of the model to distinguish those (Figure 15).

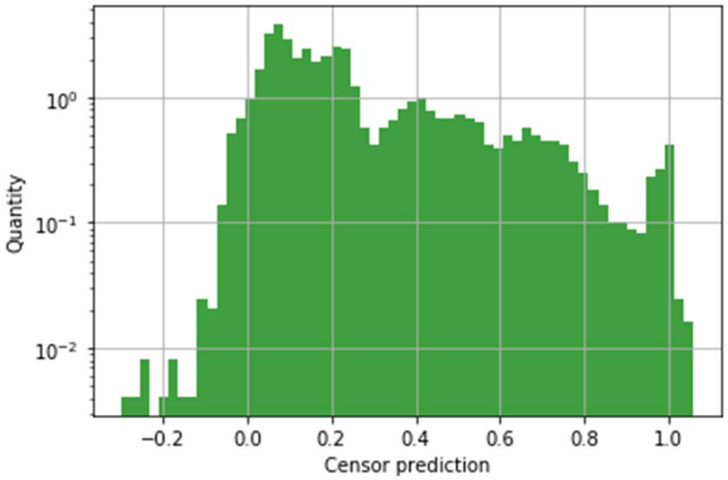

Figure 15 Histogram of predicted censoring of non-censored data

Ideally all the entries should have the prediction of zero (meaning it is a non-censored event that contains the target event) and while not perfect, most of the entries fall within 0 and 0.3, confirming that the results are coherent with the truth. A worry that remained was what time profile those events had, or in other words, if the events that were being predicted would be occurring right after the prediction instead of being more evenly distributed within the projected 3 hours. Looking at the Time-to-Event (TTE) of the entries, we can observe the following distribution (Figure 16). Note the logarithmic axis in the records axis and the normalized 3 hours in the real TTE axis. The shape of the distribution itself is expected as it is less likely to have events occurring further in time.

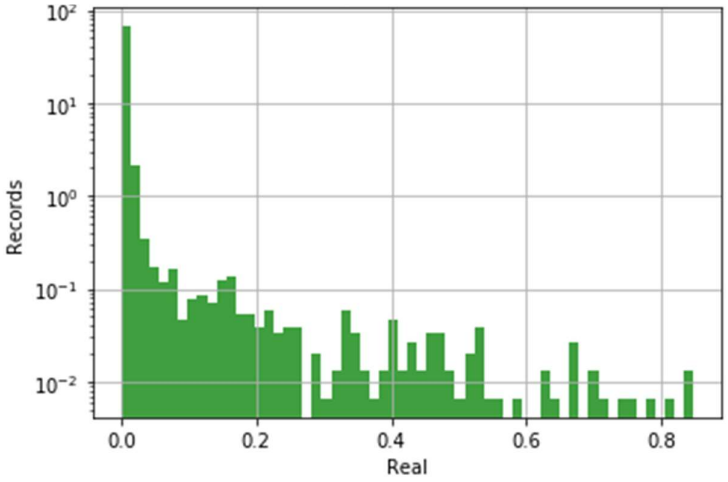

Figure 16 Histogram of real TTE values

Knowing when the events occur globally, we now look at how they compare with the predicted censoring ratio, if the prediction is not affected by the TTE, then the scatter should be fairly well distributed, conserving high TTE values for ratios that depict a strong certainty in the prediction (Figure 17).

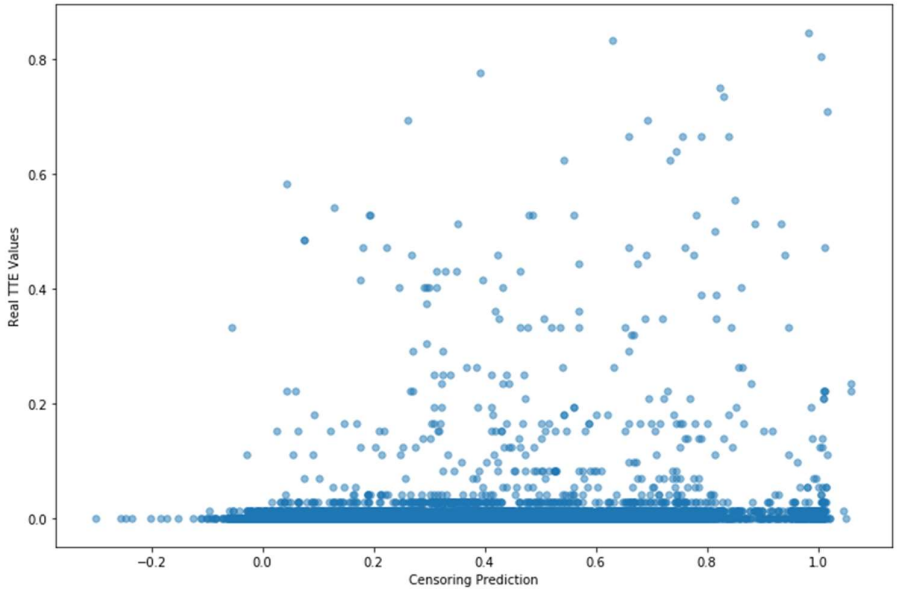

Figure 17 Comparison between real TTE values and censoring prediction

Within 0 and 0.3, where most of the predicted values fall, we see that we are not limited to predicting only events with small TTEs, even though most events have really small TTEs. To improve the obtained results several actions can be taken while keeping the same type of model:

- Increase the size of the train dataset.
- Increase the complexity of the neural network used.
- Increase the set of features used in the prediction.
- Tune the loss function to better deal with the imbalance of the data.
- Predict over multiple event problems.
- Predict over multiple time window sizes or aggregated time views.

3.2.4 Relation with SliceNet vertical use cases and 5G services

As said before, the Reliable RAN Slicing cognition UC is strongly related to the SmartGrid vertical UC. Indeed, the SmartGrid UC is enclosed within the Ultra-Reliable Low Latency Communications (URLLC) service type defined by 5G standards. Hence, it becomes imperative to guarantee means to keep the reliability of the provisioned slice in which the SmartGrid service is being supported. In this regard, the Reliable RAN Slicing cognition UC provides the means to achieve such goal. Thanks to the analysis of the RAN segment alarms, the analytical model can provide as outputs reliability scores which help on determining the most suitable NSP to construct the E2E NS during the provisioning phase of the slice. Moreover, the developed ML model acts as an enhanced sensor that allows the Actuation Framework to determine if the reliability of the slice is being compromised and act upon violations or degradations of the reliability to remedy the situation (e.g. by a modification of the NSP sequence, as explained in Section 4.3). All in all, this cognition UC allows for the delivery of a slicing service to meet the reliability of the URLLC service type, i.e. the SmartGrid vertical UC, as stated by 5G standards.

3.3 Noisy Neighbour detection

In summary, the NN analytical workflow is devoted in determining if a Virtual Network Function (VNF) instance belonging to a slice is being subject to CPU noise coming from another VNF instance collocated into the same physical server or if the VNF is being overloaded due to its internal processes (as described in D5.5 [5]). This section describes the different components designed for the training and runtime phase of the ML model for the NN detection enclosed within the framework of the Smart City vertical UC. The collected data and the performance result of the ML model are also described.

3.3.1 Design of components for the training phase

To realize the training phase of the model, a set of components were designed and implemented to materialize the different experimentations. To simulate the noise over the VNF instance of interest, a Fibonacci application was installed, and a script was developed in order to maintain the noise in the testbed. A KPI application is used in different experimentations to label the collected data. For the overload scenario, an Internet of Things (IoT) simulator was developed to stress the IoT application and to simulate traffic from a set of IoT devices.

3.3.1.1 IoT simulator

The IoT is an application developed to simulate a variable number of IoT devices connected to the network that sends telemetry data and attributes data through the whole ecosystem to the IoT control and processing platform. The device simulated by this software is a smart lighting controller used in the Smart City UC for lighting poles. All data is generated randomly within some range or selected from a list in order to be compliant with real messages sent by real devices. The simulator can be configured to control how often to send messages and how many messages to send to the platform.

To strictly observe how a device works and how the platform is able to collect all information, the application design is based on some modules (python classes) to do each one of the basic functions described below:

- **data_generator**: this module creates a Java Script Notation Object (JSON) object for telemetry data and another one for attribute data used by iot_mqtt module to send it over the network to the platform.
- **iot_devices**: this module offers the possibility to create a new device in the platform, to get the list of existing devices or to get the token id for every device in order to use it for sending Message Queuing Telemetry Transport (MQTT) messages.
- **platform_login:** this module is used to do authentication in the platform API.

- **iot_mqtt:** this is one of the main modules; it is used to send MQTT messages to every device in the platform. It allows for different message loads by controlling the number of worker threads.
- **core_app:** based on Flask Web framework this module exposes a web interface to the other modules which allows the user to control the process of generating messages, control the number of threads, create devices and list all available devices. This module makes use of every of the above modules.

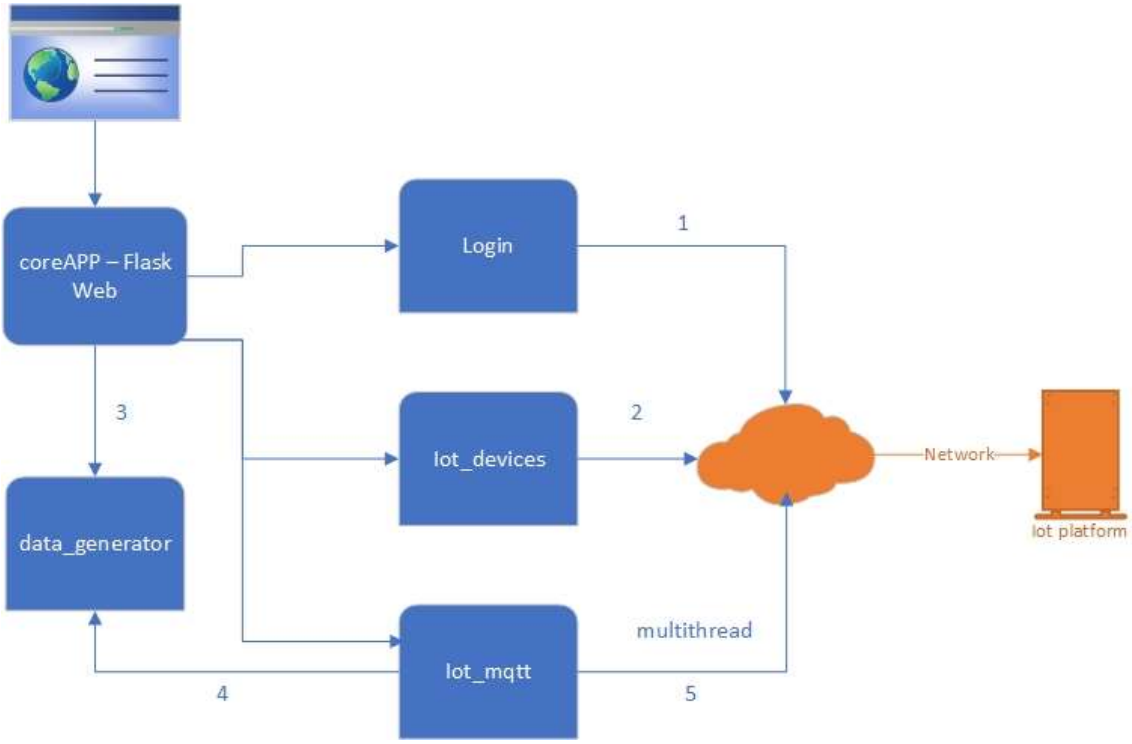

Figure 18 Software design of the IoT simulator application

Figure 18 above describes the entire design of the application with a simple flow of simulating messages from a number of IoT devices:

1. login in platform, get tokens for sessions.
2. get a list of available devices and their authentication tokens.
3. command generation of JSON messages.
4. generate message for every simulated device.
5. send to platform MQTT messages.

3.3.1.2 KPI application

The KPI application is a software tool developed in order to evaluate in real-time the E2E metrics status of the smart lighting Smart City service. It was designed to monitor the service metrics within the IoT Smart City deployed slice. The E2E NS is defined by the communication network that connects Lighting Poles, Radios, dedicated CN elements and the IoT platform. The testing packets travel though the specific VNFs within the slice up to the end-device lighting pole; in turn the end-device responds to the packets. Figure 19 depicts a schematic of the KPI application and its internal modules.

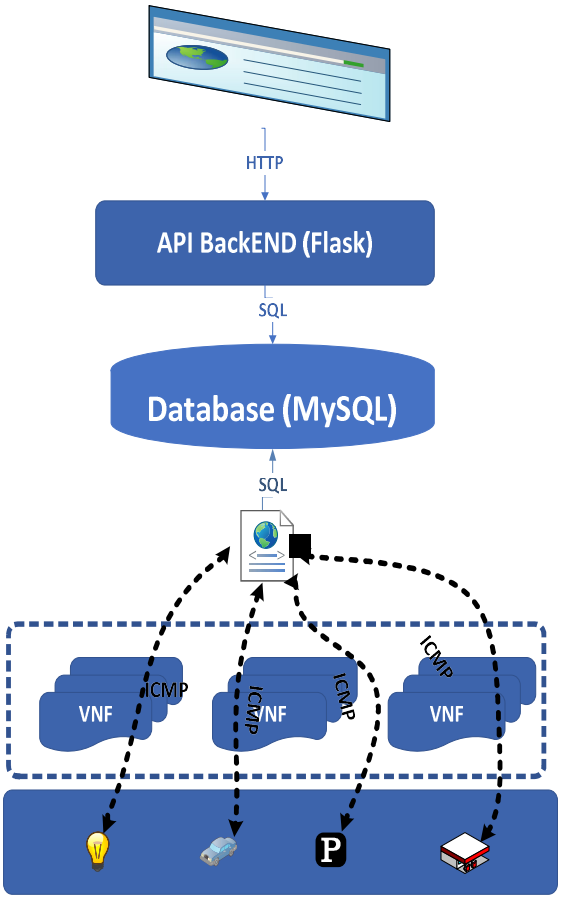

Figure 19 Smart lighting KPI software tool

In general, IoT devices are low-priced components and there are not many implemented testing features at their level. Thus, the KPI tool leverages scripts that analyse Internet Protocol (IP) Internet Control Message Protocol (ICMP) (ping) results. Based on the static IP allocation at the physical gateway level for each device lighting pole, the tool does the following actions, as described in Figure 20:

- Latency is measured using Inter-cloud Meta-scheduling (ICMS) packets:
  - The list of active devices is read from a file populated with per device IP.
  - 3 ICMP packets are sent to every device by a python script.
- The Average response time, packet loss and jitter for each device are recorded in a MySQL database.
- Using a Flask framework, information about metrics and active appliances is exposed via Representational State Transfer (REST) APIs.
- A web reporting tool exposes information provided by APIs in a user-friendly Graphical User Interface (GUI).

Figure 20 KPI software tool framework

3.3.2 Implementation details for the runtime phase

To achieve the integration of the NN detection model in the Smart City UC, a set of components were implemented and integrated together. The components are described in the following sections.

3.3.2.1 Monitoring system: Prometheus

To collect data from our testbed, we used Prometheus monitoring system. It is a monitoring solution that gathers time-series based numerical data. To monitor a specific server, Virtual Machine (VM) or VNF, we installed a Prometheus end-point (node-exporter) over it. This end-point is a REST-based interface that exposes a list of metrics and the current value of the metrics.

Prometheus server collects the metrics from server's agent over HyperText Transfer Protocol (HTTP), and then stores them locally or remotely and displays them back in the Prometheus server. For our UC we decided to install the Prometheus server on a bare metal server and supervise all hypervisors of the OpenStack infrastructure and VMs of the Smart City slice.

3.3.2.2 Noisy Neighbour ML model

The model retrieves regularly (every 15s) the metrics of VNFs to supervise and predict their status. Once we predict the noise or overload status, we expect 3 consecutive predictions of the same class before updating the state of the VNF in the Data Lake. This choice was made in order to avoid transient fluctuation happening inside the VNFs.

3.3.2.3 Integer Linear Programming model for resource migration

After applying the ML models, the corresponding action can be taken according to the VNF status; when there is an overload the VNF is scaled up, i.e. adding more CPUs or RAM, in this case we differentiate the overload from noise which is not always easy to identify. When there is noise, the identified VNF is migrated to another server that is not suffering from noise. In the case of noise, we run an Integer Linear Programming (ILP) model to migrate these VNFs to none noisy servers. Thus, we solve the NN problem in smart and optimal way. The optimization objective function of the ILP is to minimize the noise of all servers when migrating VMs to servers while satisfying the resource requirements. This ILP model will run on the orchestrator following the materialization of WP7.

3.3.2.4 Data Lake

The outputs generated by the NN ML model need to be stored at the DSP Data Lake for their consumption by the Actuation Framework (namely, the QoE Optimizer). Although its development is mainly being addressed within WP6, in order to exercise the NN UC, we developed a simplified Data Lake component. This component is based on the InfluxDB of the TICK stack, and has been installed inside a dedicated VM, which is then deployed within the NN experimental test-bed. The specific schemas in which the events are inserted and the interfaces employed to consume the data from it are reported in the context of the actuation framework and workflows (see Section 5.2).

3.3.2.5 QoE Optimizer

Lastly, the component enabling the actuations that result from the analysis from the ML model is the QoE Optimizer. In this regard, the QoE optimizer implements the actuations of VNF scaling and migration, which are the desired actions depending on the state of the VNF. The QoE Optimizer is being deployed as a container within a VM, which is then deployed along the rest of the components into the NN test-bed. The specific implementation and how actuations are achieved are discussed in detail during Section 4. Nevertheless, in the following we depict a simplified workflow, which states the main interactions across modules that allow for the realization of the NN UC.

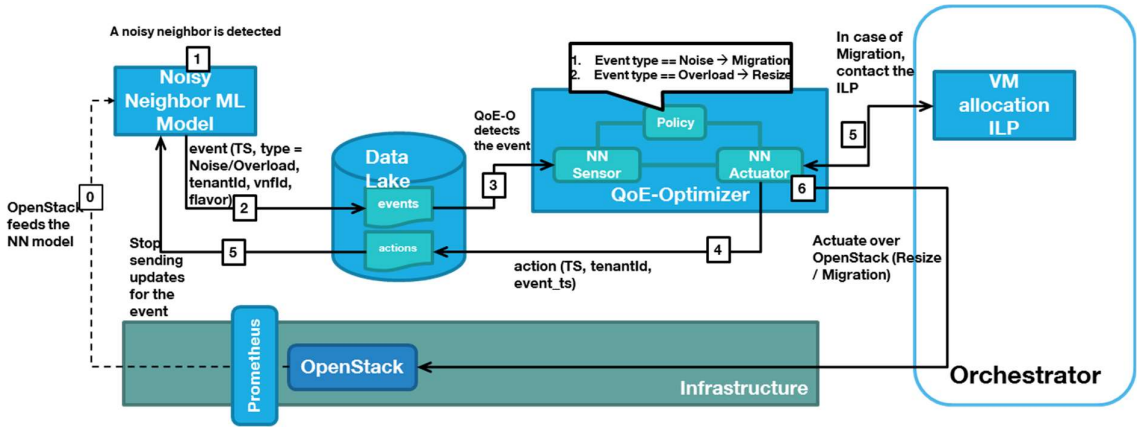

Figure 21 Noisy Neighbour UC workflow example

The following steps were implemented to realize the PoC, including all the developed and integrated software components:

0. Infrastructure utilization data is collected from OpenStack and fed to the ML model.
1. The ML derives the state (noisy, overloaded or normal) for the VNFs that are being supervised within the deployed NS.
2. The outputs of the model are inserted at the Data Lake.
3. Periodically, the QoE Optimizer polls the inserted models to check if an actuation is needed.
4. Once an actuation is under course, the QoE Optimizer inset a special event onto the Data Lake to alert the ML model. This is done in order to avoid overflowing the Data Lake with unnecessary "false predictions" due to the delay between prediction and the result from the actuation (e.g. the time required to scale or migrate a VNF).
5. The special alert events are consumed by the ML model, which stops the prediction from the specified VNF until further notice. At the same time, the QoE Optimizer contacts the ILP to gather the new host in which the VNF should be migrated (for the noisy state case).
6. The required actions are directly enforced to the OpenStack Virtual Infrastructure Manager (VIM) through the QoE Optimizer, closing the whole loop.

3.3.3 Description of data

The dataset used to create the NN detection model is obtained from monitoring the real testbed infrastructure, used for the use case implementation and demonstration; it contains 48112 records (entities) and 4 columns (features). All the records and outputs are generated using real UC data and traffic pattern for Smart City Smart Lighting UC. Table 12 provides a summary of the employed data, its usage for the learning and the techniques used for feature extraction and the learning phase. The entities are the experiments through the infrastructure for each timestamp, and the features are the Prometheus metrics as described in the SliceNet deliverable D5.2 [4], for which Table 13 provides the details for all the main features. The collected data for the different scenarios is shown in Table 14. Each status has its corresponding percentage. To label the data, we rely on the collected metric (latency, packet loss, jitter…) from the KPI application.

Table 12 Summary of employed data and learning for the Noisy Neighbour use case

Number of samples	Features per sample	Training/ test data splitting	Feature selection/ extraction	Type of learning	Algorithm for training	Source of data
48112	4	75/25 %	-Dealing with missing data -Standardization -Oversampling by using SMOTE	Supervised	Random forest	Real data extracted from VNFs in the SmartCity testbed

Table 13 Summary of the features for the Noisy Neighbour use case

Label of the feature	Description	Data type	Values/range
cpu_utilization	CPU utilization of the monitored VNF	Float	[0..100]
memory_utilization	Memory utilization of the monitored VNF	Float	[0..Max. memory assigned to the VNF]
Network inbound	Packets which originate elsewhere and arrive to the machine	Float	[265..202477] (depending on the machine)
Network outbound	Packets which originate at the machine and arrive elsewhere	Float	[1224..1 million] (depending on the machine)

Table 14 Percentage of data for each status

Status	Percentage
normal	30,7%
noise	47,7%
overload	21,6%

Figure 22 shows the dependency between the “cpu_usage” and the perceived QoS between the IoT application and the IoT simulator. In this figure, we draw the latency when we are in a noisy and a normal state. In the noisy state, the latency varies and can reach 15 ms, which mean that noise introduces a delay when delivering the packet from the IoT devices to the IoT application. In the normal state, the latency remains stable and it is almost zero.

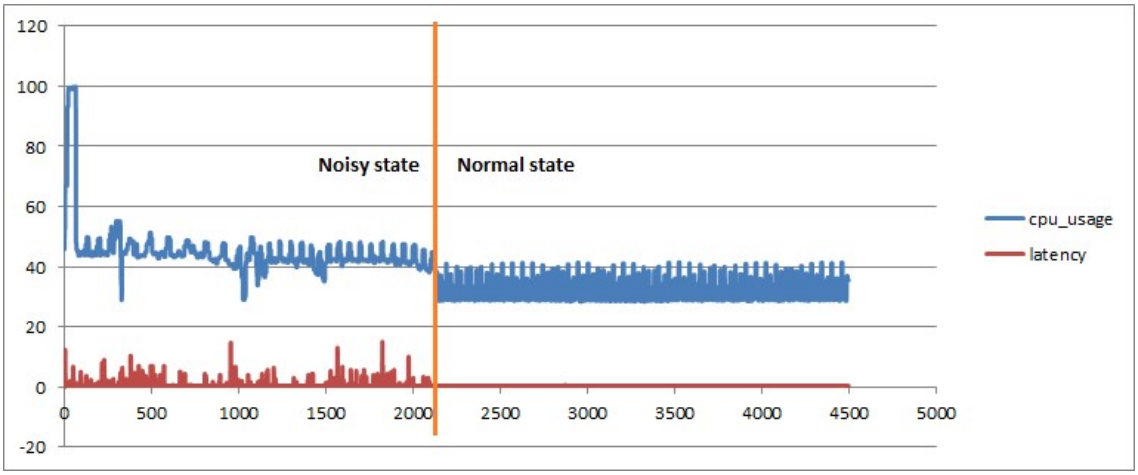

Figure 22 Correlation between VNF status and perceived latency

3.3.4 Results

After the pre-processing of data collected during the different identified scenarios (noisy, normal, overload), we are ready to use the data as input of our ML algorithm. In order to fairly compare different ML algorithms, to estimate their predictive accuracy and to choose the best one to build the NN detection model, we used 75% of the dataset for the training phase and 25% for the testing phase. To evaluate the predictive models' performance, we rely on the confusion matrix that summarizes the prediction results for the NN problem. It shows the ways in which our classification models are confused when they make predictions and gives us insight not only into the errors being made by the classifiers but more importantly the types of errors that are being made.

As shown in Table 15, Random Forest and K nearest neighbour algorithms predict well the dataset at around 99% as the mean. Meanwhile, we found that the decision tree algorithm has less accuracy, specificity and precision. We had approximately the same results when we used just CPU utilization and memory usage as features, instead of four features described earlier. We can understand that noise depends mainly on the comprehensive utilization of CPU and Memory. The random forest algorithm was selected to build the NN detection model.

Table 15 Evaluation metrics for each ML model

Algorithms / Evaluation metrics	Decision tree	Random forest	KNN
Accuracy	77,9	99,9	99,8
Classification error	22,1	0,02	0,15
Recall	77,9	99,9	99,8
Specificity	65,9	99,9	99,8
False positive rate	22,3	0,02	0,2
Precision	77,7	99,9	99,7
F_1 Score	77,79	99,9	99,74

3.3.5 Relation with SliceNet vertical use cases and 5G services

The NN cognition UC helps on articulating the QoE/QoS-aware management of the SmartCity vertical UC. The SmartCity UC is enclosed within the Massive Machine Type Communications (mMTC) service type of defined in 5G networks. The key aspect behind this type of services is the present of an enormous number of remote end-points, for which a communication service should be kept at optimal conditions when in need. Indeed, focusing on SliceNet's SmartCity UC, the smart lighting service relies

on a set of IoT sensors (i.e. the smart light poles) which exchange information with a central office at given periods of time. The IoT application is deployed as VNFs, whose performance affects the quality of the smart lighting service. As explained before, these VNFs are usually deployed onto servers in the different NFV PoPs (e.g. datacentres), normally being collocated with other VNFs belonging to the same service or with other services. Thus, the performance of the VNF may be affected given the load/noise of neighbouring VNFs as detailed along the whole section. As such, the NN cognition case offers the means to detect performance degradations onto the VNFs composing the E2E NS, which may compromise the integrity of the smart lighting service. If this is the case, the performance of the slice, hence, the service may be recovered through the interaction of the multiple WP5 components (mainly, the NN ML model and the QoE Optimizer) with the rest of the SliceNet architecture as explained previously (see Section 3.3.2.5).

3.4 QoE classification from QoS metrics

One of the main causes of unsatisfactory QoE levels relate to the underlying QoS metrics. In this regard, in order to estimate and classify correctly the perceived QoE, it is necessary to derive the relationship between underlying QoS metrics of the deployed NS and the QoE perceived by the vertical costumer. Following this reasoning, this analytical workflow is dedicated on this classification, enclosed in the scenario described in D5.5 [5]. For the current iteration, we report the implementation details of the modules that realize the workflow as well as the results obtaining during the validation of the workflow.

3.4.1 Implementation details

In order to simulate the end user's workload, an application to generate WordPress client traffic to a WordPress service has been implemented. A test driver was designed to coordinate the running of the WordPress client workloads concurrently with various levels of stress generated by the CogNETive network stresser. The test driver also maintained an index that recorded the start and end times of each experimental sample used to pair the data-bases maintained by Skydive and JMeter. The WordPress client workloads consisted of multiple concurrent downloads of the WordPress server's homepage for approximately 10 minutes. The downloads were done with HTTP Keep-Alive set to false. In the background, the network stresser ran various levels of stress including no stress.

Skydive collected network flow metrics (QoS) and stored it in an ElasticSearch database (DB). We directed Skydive to capture network flow metrics on the WordPress service interface, i.e. eth0. Likewise, JMeter collected performance date related to the client application and stored it in an influxDB. Skydive's ElasticSearch DB, Jmeter's influxDB, and the Test Driver's Index were copied to an IBM Cloud Object Store.

Analysis of the data was done in an IBM Watson Studio notebook with a Python 3.5 kernel. The Python Data Analysis Library (Pandas) was used to aggregate the QoS and QoE measures. Finally, the Python Scikit-learn library was used for creating the ML models. In our analysis, flow duration was derived from the difference between Skydive's Metric.Last and Metric.Start. Flow duration was used as the basis for its QoS measure. Likewise, the analysis used Jmeter's collection of average service completion time as the basis for its QoE measure, service time completion.

3.4.2 Description of data

Skydive is used to collect per flow network metrics from the WordPress server's eth0 interface. These metrics are then transformed and aggregated to supply the QoS features used to generate the ML model for predicting QoE classifications, namely binary and multiclass classifications of benchmark durations.

Specifically, the following raw Skydive TCP flow metrics are captured (where A is the source IP address and B is the destination source address of a flow):

- Metric.Start: flow start time
- Metric.Last: flow end time
- Metric.ABBytes: number of bytes sent from A to B
- Metric.BABytes: number of bytes sent from B to A
- Metric.ABPackets: number of data packets sent from A to B
- Metric.BAPackets: number of data packets sent from B to A
- Metric.RTT: round trip time

The raw flow metrics above are transformed into per flow metrics described below:

- flow_duration: Metric.Last - Metric.Start
- bytes_per_flow: Metric.ABBytes + Metric.BABytes) / flow_duration
- packets_per_flow: Metric.ABPackets + Metric.BAPackets / 'flow_duration
- AB_bytes_per_flow: Metric.ABBytes / flow_duration
- BA_bytes_per_flow: Metric.BABytes / flow_duration
- AB_packets_per_flow: Metric.ABPackets / flow_duration
- BA_packets_per_flow: Metric.BAPackets / flow_duration
- RTT: Metric.RTT

The above transformations are then aggregated into per benchmark instance mean values; these mean values are used as QoS features for the ML training and testing sets. The following QoS features are used in the evaluation:

- flow_duration_mean
- bytes_per_flow_mean
- packets_per_flow_mean
- AB_bytes_per_flow_mean
- BA_bytes_per_flow_mean
- AB_packets_per_flow_mean
- BA_packets_per_flow_mean
- RTT_mean

All combinations of the above features are exercised to determine which should be used by the classifiers described below to best predict the target QoE. Different scikit classifiers were used in the analysis. The classifiers are compared to determine which best predicts the target QoE using QoS features described above. The following classifiers are compared in the evaluation:

- LogisticRegression
- DecisionTreeClassifier
- KNeighborsClassifier
- LinearDiscriminantAnalysis
- RandomForestClassifier
- GaussianNB
- SVC
- MLPClassifier (removed because it was taking too long)
- GaussianProcessClassifier (removed because it was taking too long)
- AdaBoostClassifier
- QuadraticDiscriminantAnalysis

In order to determine which classifier and feature combinations perform best we rank their predictions by their F1, accuracy, and log loss scores. The results are described in the next section.

We collected QoE and QoS data for training and testing our ML model. The training set included over 1253 training samples and our testing set included 200 testing samples. Each sample consisted of the measured QoE and QoS data collected on a benchmark instance running concurrently with some stress (including no stress).

Classifications are based on benchmark durations with no background stress:

The Binary Classification decision boundaries are:

- good: range(0,quantile(0.99))
- bad: range(quantile(0.99),infinity)

The MultiClass Classification decision boundaries are:

- good: range(0,quantile(0.75))
- acceptable: range(quantile(0.75),quantile(0.99))
- bad: range(quantile(0.99),infinity)

The classifications distribution of the training set and testing set are described in Table 16.

Table 16 QoE from QoS classification distributions

Number of samples	Classification	QoE	Class	Range	Sample set	Source of data
706	Binary	Good	0	0-224983	Training	Real data extracted from WordPress server
547	Binary	Bad	1	224983-infinity	Training	Real data extracted from WordPress server
142	Binary	Good	0	0-224983	Testing	Real data extracted from WordPress server
58	Binary	Bad	1	224983-infinity	Testing	Real data extracted from WordPress server
231	Multiclass	Good	0	0-167708	Training	Real data extracted from WordPress server
475	Multiclass	Acceptable	1	167708-224983	Training	Real data extracted from WordPress server
547	Multiclass	Bad	2	224983-infinity	Training	Real data extracted from WordPress server
63	Multiclass	Good	0	0-167708	Testing	Real data extracted from WordPress server
79	Multiclass	Acceptable	1	167708-224983	Testing	Real data extracted from WordPress server
58	Multiclass	Bad	2	224983-infinity	Testing	Real data extracted from WordPress server

Figure 23 illustrates the binary classification distribution of the training set.

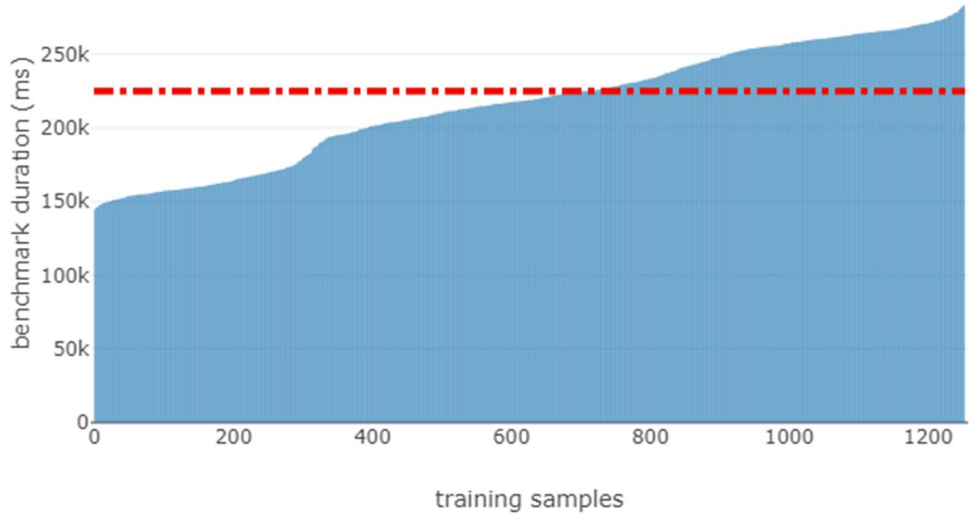

Figure 23 QoE from QoS binary classification training set distribution

Figure 24 illustrates the binary classification distribution of the testing set.

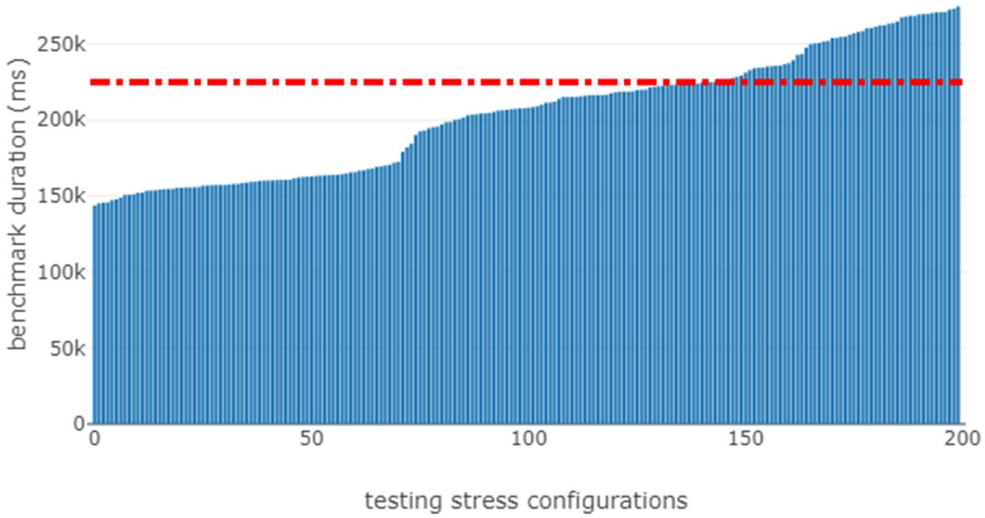

Figure 24 QoE from QoS binary classification testing set distribution

Figure 25 illustrates the multiclass classification distribution of the training set.

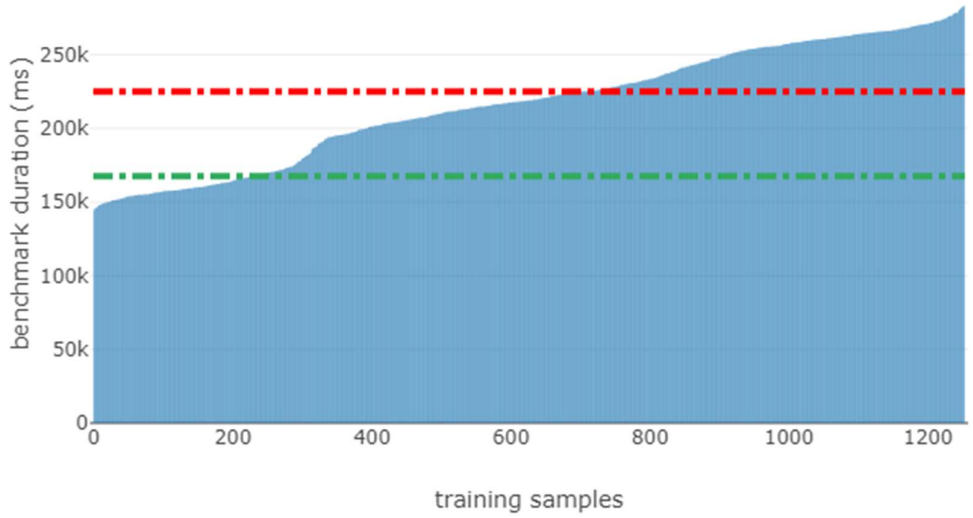

Figure 25 QoE from QoS multiclass classification training set distribution

Figure 26 illustrates the binary classification distribution of the testing set.

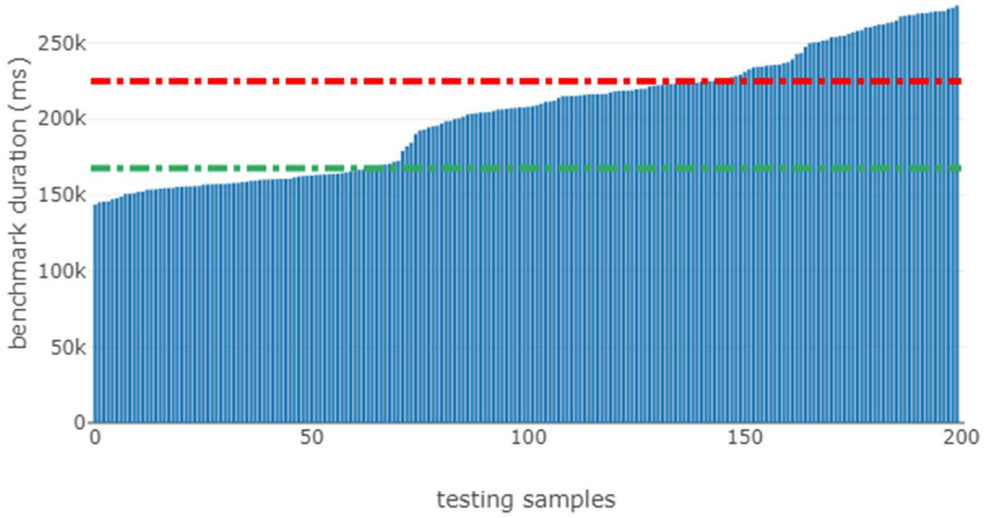

Figure 26 QoE from QoS binary classification testing set distribution

3.4.3 Results

The results of the PoC are summarized in Table 17. The Quadratic Discriminant Analysis classifier is the best classifier for both the binary and multiclass classifications. For binary classification, features *flow_duration_mean* and *BA_packets_per_flow_mean* resulted in the highest-ranking prediction, while for multiclass classification features *flow_duration_mean* and *packets_per_flow_mean* were the best ones.

Table 17 QoE from QoS results summary

Classification	Classifier	Features	F1 score	Accuracy score	Log loss score
Binary	Quadratic Discriminant Analysis	flow_duration_mean BA_packets_per_flow_mean	0.92	0.92	0.27

Multiclass	Quadratic Discriminant Analysis	flow_duration_mean packets_per_flow_mean	0.84	0.84	0.48

The results of the binary classification are described in the Figure 27 confusion matrix and Figure 28 Actual vs. Predicted distribution plot. The two figures show that there were 184 correct estimations, 5 under estimations and 11 over estimations.

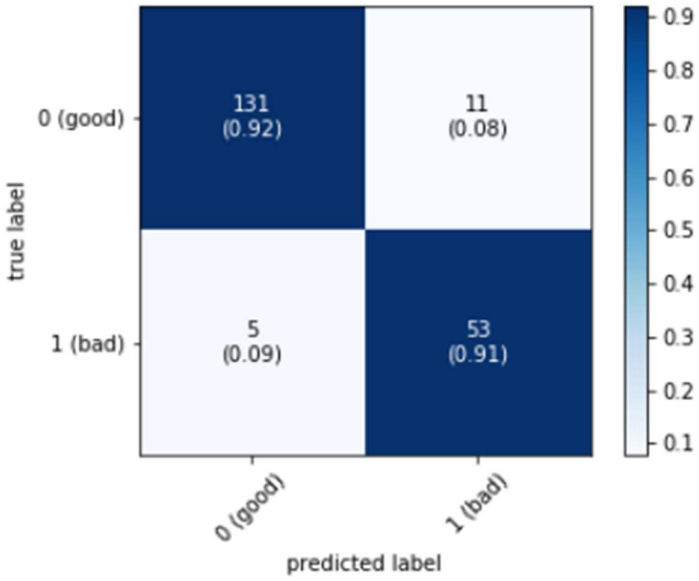

Figure 27 QoE from QoS binary classification confusion matrix

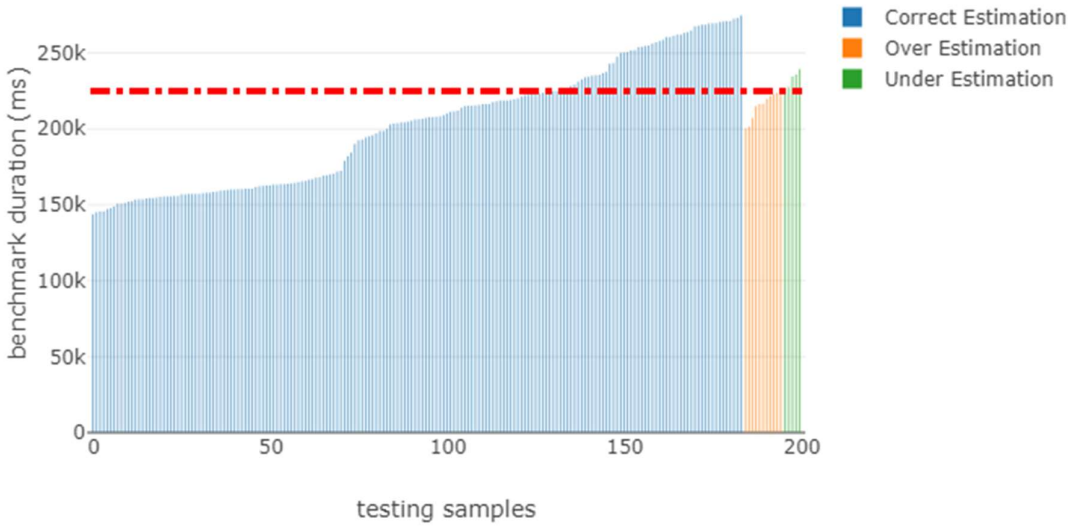

Figure 28 QoE from QoS binary classification actual vs prediction

The results of the multiclass classification are described in the Figure 29 confusion matrix and Figure 30 Actual vs. Predicted distribution plot. The two figures show that there were 167 correct estimations, 14 under estimations and 19 over estimations.

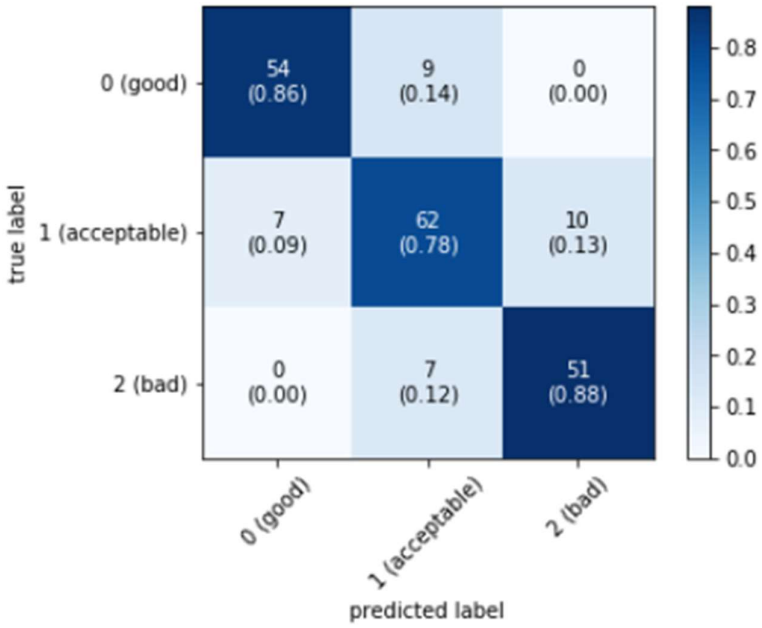

Figure 29 QoE from QoS multiclass classification confusion matrix

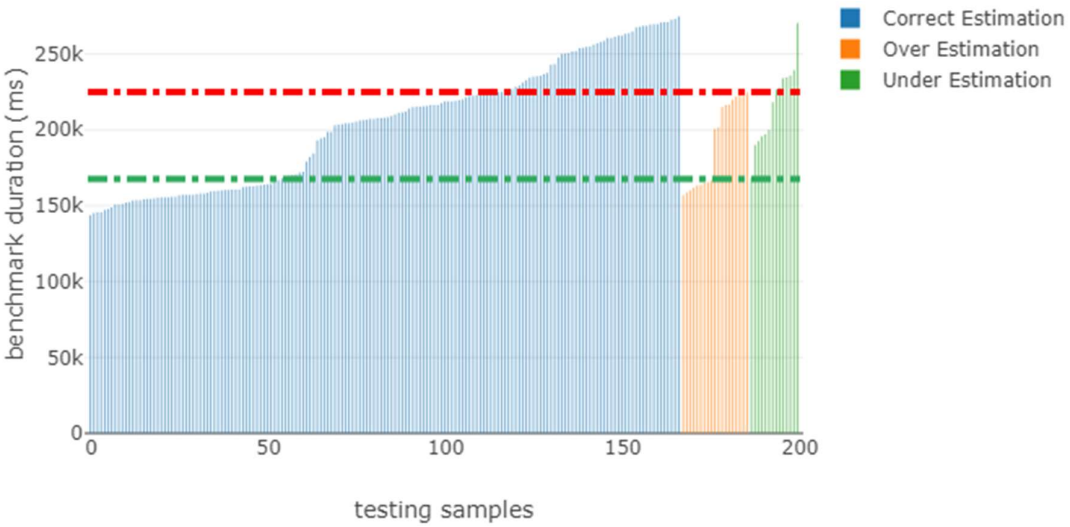

Figure 30 QoE from QoS multiclass classification actual vs prediction

In a production setting where the classifier predictions could be used to stimulate some remedial action, incorrect predictions could have negative consequences. Over estimation could result in allocation of more additional resources than required. Whereas, under estimation could result in inaction, whose consequent might be user dissatisfaction or a Service Level Agreement (SLA) violation. The models' accuracy is not good enough. In order to improve the predictions, we intend to try other classification methods and include additional features. We also want to see how this model performs on more complex workloads and for persistent connections, which will be explored and consolidated during the next iteration of WP5.

3.4.4 Relation with SliceNet vertical use cases and 5G services

The goal of the QoS to QoE classification cognition UC is to use ML models to predict QoE at run-time and to trigger corrective measures within the SliceNet framework. In particular, such QoE estimations

would serve as triggers for actuations by the QoE Optimizer. Given the monitored QoS parameters for the several deployed E2E NS, the corresponding QoE could be derived thanks to the proposed approach. Given such metric, a policy defined within the framework of the PF would define which actions (actuations) need to be triggered when facing bad QoE situations, for which the measurements would come as outputs from the proposed ML model. Then, these outputs would feed an instance of the QoE Optimizer, which will check the conditions stated by the slice policies in place and trigger the necessary (re-)configurations in case the conditions are met (e.g. the estimated QoE is below a certain threshold). Such approach can be exploited by the multiple SliceNet vertical UC, specially the eHealth one, since it correlates the several network-level metrics (QoS) with the quality being perceived by the customers of the slice. Note that, while the developed ML may not match perfectly with the different vertical UCs expectations, mainly due to the different data employed in them, which differs between them and the PoC and also between the multiple vertical UCs, the methodology employed sets a solid framework to allow for QoE classification-based management of NS within the context of the SliceNet architecture.

3.5 RAN optimization

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 NSes 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, as described in D5.5.

3.5.1 Design of components

We analyse the cognitive RAN described in D5.5 by considering a video streaming UC on the top of the OpenAirInterface (OAI) [12] and Mosaic5G [13] platforms (Figure 31 (b)) on a technology development level for lab validation. We will show how the combination of the radio resource management information with spectrum management information can be used to optimize resource utilization. We demonstrate how this resource utilization can help a video optimizer to fulfil its performance objective to maximize a user's video quality while running in an evolved node B (eNB) in low-power mode and where a handover is not a viable option. 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) [14] (namely transmission power, operating frequency, and bandwidth) and Radio Resource Management (RRM) Control Applications (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 Deliverable 5.5, Section 6.1)). SMA parameters can easily conflict with other domains; for instance, the operating frequency of cells of another operator or the regulatory maximum allowed transmission power. These conflicts and others are addressed by the proposed RRM-SMA resource utilization approach. 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 monitored and fed into the knowledge base (KB). Through it, the video optimizer is able to track the network state and control the RAN through appropriate actions in order to satisfy the active users.

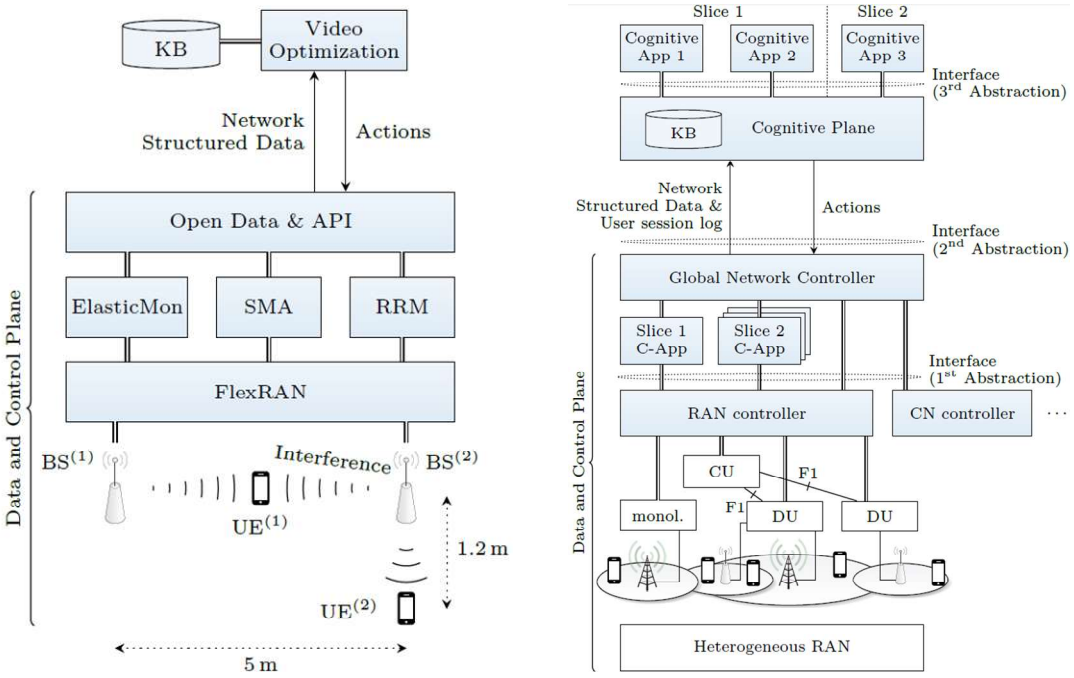

Figure 31 Integration of the Data-Driven Control and Management (DDCM) with a RAN. (a) Architecture of cognitive RAN control and (b) Example of DDCM integration in SliceNet testbed [D5.5]

3.5.2 Implementation details

We report relevant experimental results that address the optimization of resource utilization in a small-cell setting, tailored to the case of 4G. Due to the increased cell densification and lighter network planning, situations of dynamically varying interference are deemed to be more and more common in the case of 5G. Two eNBs ($BS^{(1)}$, $BS^{(2)}$) are located at 5 m of each other. Initially, only $BS^{(1)}$ streams video to its user equipment ($UE^{(1)}$) located exactly in between both eNBs. The eNBs operate in the same band 7 on close to equal frequencies and therefore create interference when both are connected to users. Both eNBs operate around 2.6 GHz and with a bandwidth of 5 MHz.

We measured the average throughput $Thr^{(1)}$ between $BS^{(1)}$ and $UE^{(1)}$ and $Thr^{(2)}$ between $BS^{(2)}$ and $UE^{(2)}$ in Mb/s based on six independent measurements of User Datagram Protocol (UDP) traffic of 20 s length, representing average video performance. The control of the network is exerted through the following parameters:

- The transmission power (P_{tx}) can be lowered in order to reduce interference, save energy (green networking) and reduce electromagnetic fields exposure as long as the network's objectives are met. We classify the transmission power into two levels, 'L' and 'M'.
- In a similar vein, an overlap in the operating frequency (BW_o) might be desirable as long as the network's performance does not suffer, because in this case the frequency reuse in a given area increases the spectral efficiency.

The system throughput saturates at 16.8 Mb/s. We normalize the throughput to obtain a score per base station.

3.5.3 Results

Consider a single $BS^{(1)}$ sending traffic to $UE^{(1)}$ (see Figure 31 (b)), using a low transmission power in order to save energy. The video optimizer is fed the information through the C-Apps, builds the model

and saves the state as state *1* in (Table 18, where No. column indicates the number of state). In particular, a relationship between any two rows indicates a relationship between the low transmission power, interference-free transmission and the resulting throughput for the single user in different states.

Table 18 Simplified KB for Band 7, 25 RBs, and $PTX^{(1)}$ = M.

State No.	$P_{TX}^{(1)}$	BW_o (%)	$Thr^{(1)}$ (Mbps)	$Thr\ Score^{(1)}$	$Thr^{(2)}$ (Mbps)	$Thr\ Score^{(2)}$
1	L	0	12.1	0.72	16.8	1.00
2	L	50	5.5	0.33	16.8	1.00
3	L	100	5.5	0.33	16.7	0.99
4	M	0	16.8	1.00	16.8	1.00
5	M	50	16.6	0.99	16.8	1.00
6	M	100	14.2	0.85	15.8	0.94

Consider $BS^{(2)}$ is in a passive-ready state and senses the environment for UE activity. For instance, users moving towards this base station would be recognized through an increasing signal energy. Feeding this information to the KB, increasing activity recognized. Since $BS^{(1)}$ serves its current user and is not able to serve another, $BS^{(2)}$ is switched on.

Now, a new user $UE^{(2),}$ associating to $BS^{(2)}$ enters the system. Since $BS^{(2)}$ sends on the same frequency and due to the close proximity of the two base stations and increasing activity of $BS^{(2)}$ interference for user $UE^{(1)}$ increases. This gradually lowers the score of $UE^{(1)}$. The reasoner searches through the KB (Table 18) in order to identify a new configuration, which eases the imperfect state *3*, to maintain the SLA. Furthermore, $BS^{(2)}$ is labelled as a "bad" base station, introducing a new concept in the KB ($score^{(2)}$).

The reasoner identifies as possible future states (a) state *1* with no bandwidth overlap by changing the frequency or (b) a new state (later becoming state *6*) using an increased transmission power. This new state is to be understood as a forecast future state, based on data collected in the past or by observing the other base stations with higher transmission power.

Based on the inferred future states, the prediction evaluates these states in terms of the future impact, leaving the decision maker with the following choices:

- For an interference-free transmission, a shift in frequency necessitates a restart of the base station with a reconnect of $UE^{(1)}$. In the long term, this re-establishes the previous status but in the short term degrades the user experience because connectivity is interrupted.
- Changing the transmission power. This has the advantage of maintaining service continuity, but the exact score needs to be predicted as close as possible. Given that $UE^{(2)}$ has a high throughput, it is expected that even under increased interference the throughput will be acceptable between the current throughput (5.5 Mb/s) and the other base station's throughput (16.7 Mb/s).

The control decision made by the video optimizer opts for the increase of transmission power in an attempt to maintain service continuity, sending the corresponding action to the SMA app. Once this is established, the system switches to the state *6* and experiences an even slightly higher throughput than what has been observed before. As the video streaming is downlink-dominant and though not explicitly considered in our KB, RRM is triggered to isolate each user in uplink by creating two slices such that the uplink interference is minimized.

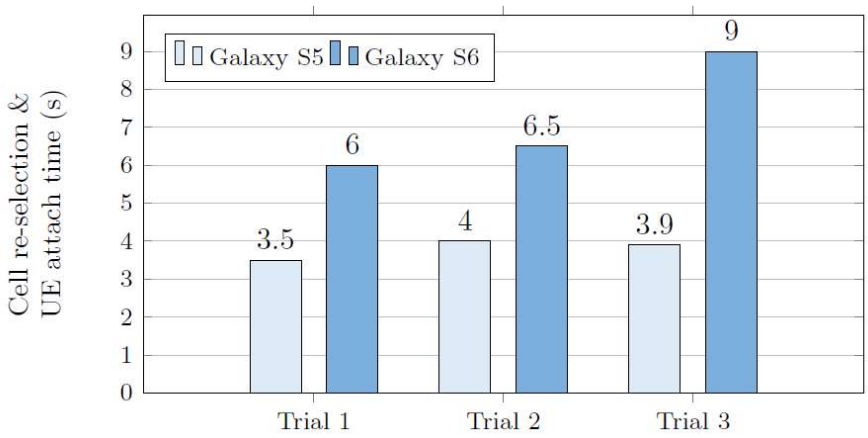

Figure 32 Indicative UE-related reattach times for different UEs

The system now deviates from part of the original objective to save energy. Furthermore, it is in a state of overprovisioning, since the 12 Mb/s of state *1* were deemed enough. Therefore, when users are inactive, video optimizer requests SMA to send a new reconfiguration command to another frequency and lower the transmission power, reaching again state *1*. This is shown in Figure 32, as phones typically find their old carrier rather fast.

Other objectives could be fed to the video optimizer to influence the outcome:

1. Avoid interference to high priority users and/or their slices,
2. Small base station energy consumption,
3. Maximize joint system throughput,
4. High frequency reuse.

Objectives (1) and (2) would likely result in choosing state *1* whereas it is likely that the system would always change to state *3* for objective (3). Other possibilities could include instantiation of NSes in the downlink to separate users in the RAN domain. In addition, objective (4) could lead to a state *5*.

3.5.4 Relation with SliceNet vertical use cases and 5G services

The RAN segment is the key enabler of 5G and vertical UCs supported across the E2E 5G infrastructure. It allows for the flexible and dynamic communication of the multiple UE that exchange information across the multi-segment and multi-layered network infrastructure. As such, it becomes essential that the performance of the RAN segment is maintained at its optimum in order to guarantee the quality of all the communications and, as a consequence, all the services/applications supported by it. Indeed, all the considered SliceNet vertical UCs (i.e. SmartGrid, SmartCity and eHealth) rely heavily on the RAN segment and its slicing capabilities in order to be supported. The SliceNet architecture should not only provide the means to efficiently slice the RAN in order to deploy E2E NS in support of vertical but also provide the means to optimize its performance at the different NSPs engaged in the slice run time. In this regard, the RAN Optimization cognition UC provides the necessary intelligence to achieve this goal. The reasoner functions of the RAN DDCM framework can be assumed by the Analyzer component of the Cognition Plane, which, as explained during Section 2.2, may be instantiated either at NSP or DSP levels. In such a case, the analysis functions and the intelligence is exercised at the NSP level, which is the entity responsible to collect the data from the RAN and store it in its own KB (i.e. NSP Data Lake) for its later consumption by analytical functions, such as the ones presented in this cognition UC. Thanks to the collected data at the RAN segment and the intelligent mechanisms described across the section, the performance of the RAN available at the several NSPs can keep its performance at peak levels, helping on maintaining satisfactory QoE levels independently of the considered SliceNet vertical UC.

3.6 Anomaly detection

This workflow is focused on the prediction of future QoE using QoS metrics, more specifically, in the prediction of anomalies in the LTE RAN segment, which could affect the QoE of deployed slices supporting vertical UC. In this regard, the focus of the developed ML model is to serve as an intelligent QoE sensor in the context of the SliceNet eHealth UC, as described in D5.5. For the current iteration, we describe the experimental validations performed for the analytical workflow, starting from the used data, the architecture and components of the analytical workflows and the concrete results obtained during the ML model evaluation within the umbrella of the eHealth UC.

3.6.1 Description of data

In order to collect real data, a mobile phone (UE) has been put into an ambulance. A mobile application is installed in the phone, which allows computing the network QoS metrics perceived by the UE. Eight real traces are captured having between 2 to 4 hours of length. A measurement is computed every 1 second. For each trace, seven QoS metrics (KPI) are considered for determining the quality of the network signal. The data set is described in the following Table 19 whereas the used features are described in Table 20.

Table 19 Summary of employed data and learning for the Anomaly Detection use case

Number of samples	Features per sample	Training /test data splitting	Feature selection/ extraction	Type of learning	Algorithm for training	Source of data	Annotation
11820	7	7 traces for training; 1 trace for testing	-Smoothing of the curves of KPIs -Application of a dimension reduction by using a principal component analysis for functional data -Oversampling by using SMOTE technique	Supervised	Random forest	Real data captured from a UE	The training set is a 3D matrix composed of 11820 instances. Each instance is defined by 7 KPIs and each KPI is defined by 300 values that correspond to 5 minutes of observation.

Table 20 Summary of the features for the Anomaly Detection use case

Label of the feature	Description	Data type	Values/range
Reference Signal Receive Power (RSSP)	The average power received from a single reference signal	Float	[-141 dBm ..-44 dBm]

Reference Signal Receive Quality (RSSQ)	It indicates quality of the received signal	Float	[-19.5dB ..15dB]
Signal to Noise Ratio (SNR)	It measures the signal strength relative to background noise.	Float	[0..41]
Channel Quality Indicator (CQI)	It is an indicator carrying the information on how good/bad the communication channel quality is	Float	[0..17]
Received Signal Strength Indication (RSSI)	Signal strength measured from all base stations	Float	[0..49]
Downlink bitrate (DL-bitrate)	The bit rate in down link	Float	[0..112649]
Uplink bitrate (UL-bitrate)	The bit rate in up link	Float	[0..9540]

3.6.2 Architecture and description of components

Figure 33 describes the architecture of the signal strength prediction model for the eHealth UC. Note that this figure depicts the Anomaly Detection cognition UC as exercised from the DSP perspective, in which the QoE Optimizer is the actuation endpoint. The UE collects measurements about the network for a certain period. These measurements serve as training data and are injected as external data into the Data Lake. The aggregator will pre-process the training data by dealing with missing data, smoothing and putting the data into the right format for training. The pre-processed data is now ready to be trained by the analyser. The resulted anomaly prediction model will learn the patterns of normal and of suspicious behaviours of the QoS metrics regarding the signal strength. A trained model is produced and saved within the analyser. This model is usable for the exploitation phase in order to forecast the signal strength quality for new measurements collected by the UE. In case of a prediction of bad signal strength in the future 5 minutes, the actuation endpoint (QoE Optimizer for the DSP case and TAL engine for the NSP case). Depending on the prediction, the actuation endpoint, as a client of the prediction data, may solicit the orchestrator to apply corresponding re-configurations in order to correct the quality of the signal issue in advance, which allows avoiding service degradation and a bad QoE perceived by the involved medical multi-disciplinary team.

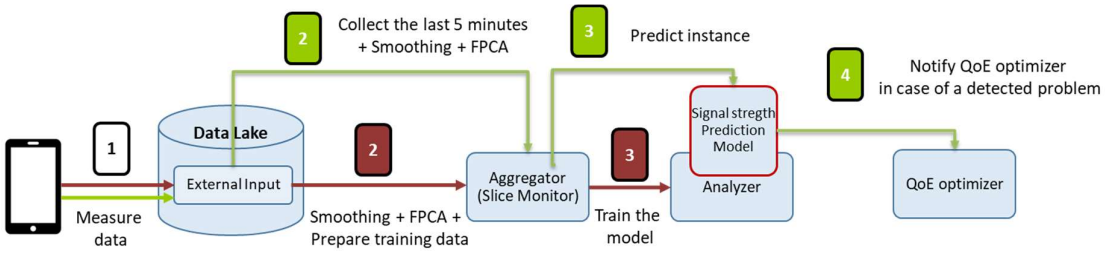

Figure 33 Architecture of signal strength prediction model for eHealth UC (red for training phase and green for run-time phase)

3.6.3 Actuation

As commented in the previous sub-section, the outputs of the Anomaly Detection serve as input for an actuation endpoint. In this regard, two levels may be considered: 1) NSP-level actuations, in which (re-)configurations are enforced within the NSP by the TAL engine without the DSP being involved; and 2) DSP-level actuations, in which (re-)configurations take place at the DSP, involving the full E2E NS perspective. With such a premise, the actuation will take place only when the QoE Optimizer (DSP) or the TAL engine is notified that the model has predicted a bad signal quality in the prediction horizon

(i.e. the future 5 minutes) and when 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 signal strength degradation. Note that in the case of DSP-level actuations, a cross-role/layer communication for the actuation may be involved, following the approach described in sub-section 2.1.1.

Once the ML model predicts low signal strength for an urgent video stream, several scenarios for actuation may possible, exercised at different roles of the SliceNet architecture. 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. Other potential actuations may involve the migration of VNFs from one NFV PoP to another, or the association of the UE to healthier RAN slices, wither within the same NSP or from other available NSPs. These could be potential actuations that may be exercised thanks to the capabilities exposed by the TAL Engine (NSP-level) and the QoE Optimizer module of the Actuation Framework (DSP-level). Section 4.4 for provides concrete examples of potential actuations, their workflows and scenarios.

3.6.4 Implementation details

The model has been implemented using R programming language [15]. Among the eight real traces, seven were used for the training phase and one is kept for the run-time phase. The first step aims to apply a sliding window that will transform the data into a 3D matrix. Each window will observe the KPIs for the last five minute and it will label the data by observing the future five minutes. The window will advance each 5 seconds. The labelling aims to define the signal strength label for each instance by according thresholds to RSRP, RSRQ and SNR metrics with the help of a telecom expert as depicted in Table 21.

Table 21 Labelling example for the Anomaly Detection UC

Class	RSRP (dBm)	RSRQ (dB)	SNR (dB)
Good condition of signal strength	>-90	>-15	>13
Bad condition of signal strength	>=-90	>=-15	<=13

The result of the previous step is a 3D matrix composed of 11820 instances. Each instance is defined by 7 KPIs and each KPI is defined by 300 values that correspond to 5 minutes of observation. The resulted training set is imbalanced. Only 25% of all instances correspond to problems in signal strength.

A GUI has been developed using R shiny library as illustrated in Figure 34 and Figure 35. In the GUI, the context of the demo is explained. In background, the anomaly prediction ML model is running over the test set. Every 5 seconds, a session in the GUI is created. Within this session, the model reads an instance in the test set, and it plots all the KPIs curves observed for the last 5 minutes. The trained model predicts the signal strength for the future 5 minutes and it shows the prediction result. In case of an anomaly, a notification appears.

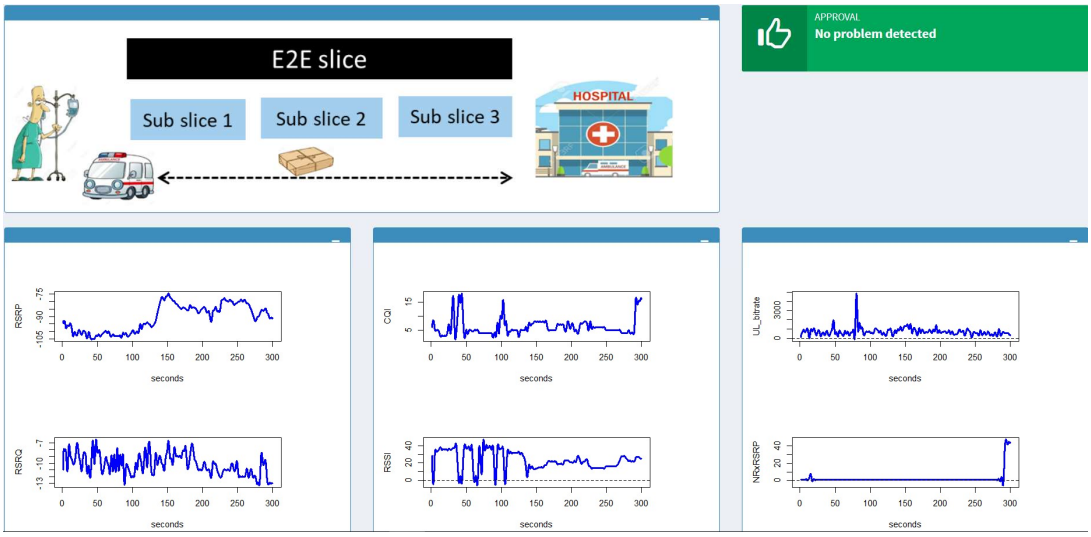

Figure 34 Snapshot of the demo in case of a detected good signal quality for the future 5 minutes

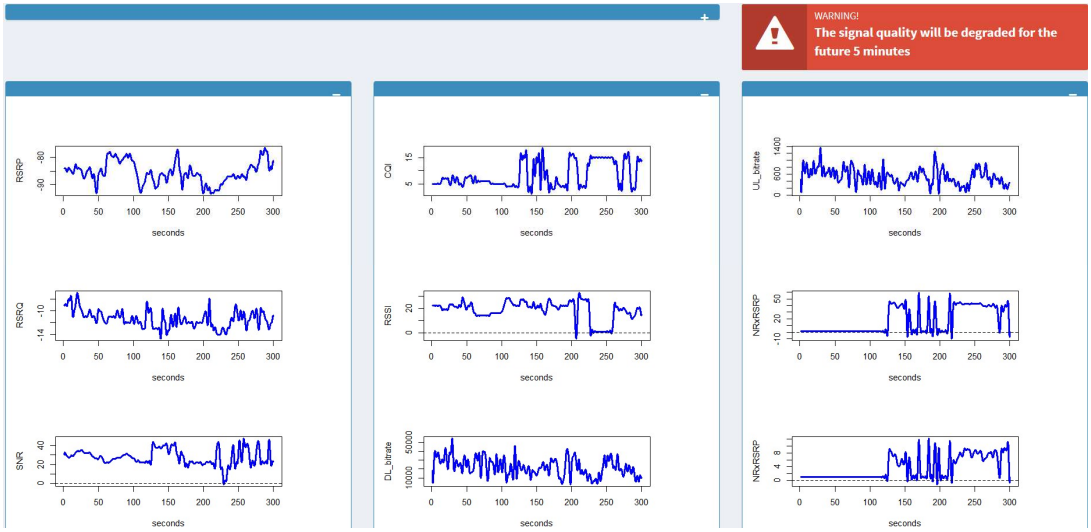

Figure 35 Snapshot of the demo in case of a detected bad signal quality for the future 5 minutes

3.6.5 Results

In order to evaluate the model, two techniques are used: (1) Evaluation using cross-validation technique over the training set; and (2) evaluation over a test set. The evaluation metrics are the following:

- **Correct classification rate**: also called **accuracy**.

$$CCR = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}}$$

- **Recall**: it is a measure that indicates to which point the degradation in signal strength is detected among all the real problems in the data.

$$Recall = \frac{\text{Number of correctly detected problems}}{\text{Total number of real problems in the data}} = \frac{\text{True positive}}{\text{True positive+Fa negative}}$$

- **Precision**: It is a measure that indicates to which point the detected degradation in signal strength are pertinent.

$$Precision = \frac{\text{Number of correctly detected problems}}{\text{Sum of the total number of real problems in the data and of false alarms}} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

- **F_1-score**: It is a weighted average of the precision and recall, (harmonic mean) where an F_1 score reaches its best value at 100% and worst at 0%.

$$F_1\text{-}score = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Table 22 presents the results over the training data using a cross validation with 5 folds where the classification is achieved using a random forest since it is well-known in case of imbalanced data. It also illustrates the results over the test data that corresponds to a real trace covering 2 hours and 21 minutes with a total of 960 instances. As shown in Table 22, the results are promising since it indicates that 93% of the signal strength degradation in the data is detected and 91% of the detected problems are not false alarms.

Table 22 Performance of signal strength degradation prediction model

Evaluation metric	Results of the cross-validation	Results over the test set
Precision	91%	99%
Recall	93%	99%
F_1-score	92%	99%
CCR	96%	99%

3.6.6 Relation with SliceNet vertical use cases and 5G services

The Anomaly Detection cognition UC is an integral part for the maintenance of the QoS and, as a consequence, the QoE of services deployed within the context of the SliceNet's eHealth vertical UC. The eHealth vertical UC is enclosed within the Enhanced Mobile Broadband (eMBB) type of service defined by 5G standards. The highlight of such type of services is the provisioning of high E2E bit-rates to high mobility endpoints (i.e. UEs). Such characteristic matches perfectly with the eHealth services defined within SliceNet, for example, tele-stroke assessment, which require the sending of audio and video data at high bit-rates from a mobile endpoint (the ambulance) to a central remote endpoint (the hospital). In such situation, it is essential to maintain both the quality of the data stream transmission as well as allowing for the mobility of the endpoints within the provisioned E2E NS. The Anomaly Detection ML model serves as an advanced QoS/QoE sensor which allows, given data monitored from the RAN segment and the UEs, to determine if an anomalous situation would happen to the data streams (i.e. the communication), thus affecting to the eHealth service. Thanks to the prediction outputs from the ML model, the Actuation Framework enclosed within the WP5 architecture (mainly, the QoE Optimizer) can enforce the necessary action previously decided (e.g. by means of a policy) that are needed to correct, or even avoid, the anomalous detection that is being predicted through the model.

4 Vertical-informed actuators workflows

4.1 Scope and preliminary implementation

The scope of the implemented actuation workflows, their details and purpose as well as their scenarios have been already described in the previous iteration I of WP5 (see D5.5 [5]). Figure 36 summarizes the overall actuation process, stressing the presence of the different sources of stimulus, which are then inserted onto the Data Lake. Then, the QoE optimizer, together with the disseminated policies, triggers the necessary actuations and enforces them towards SliceNet's Orchestration Plane.

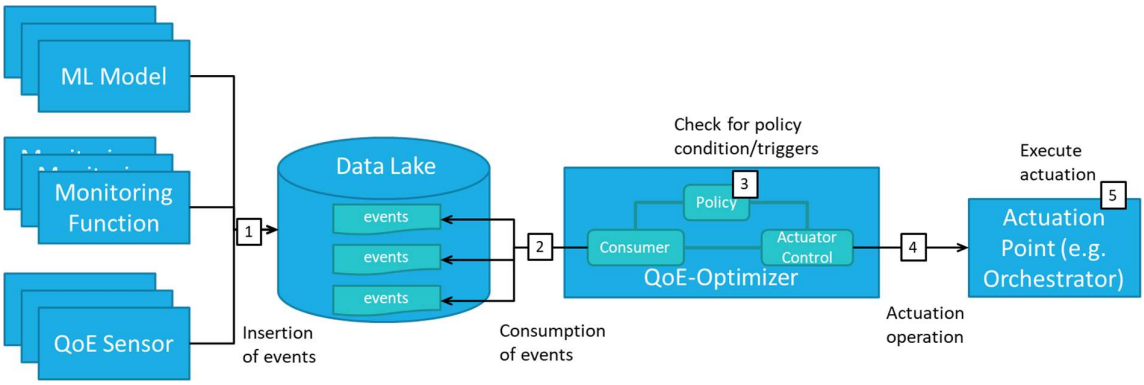

Figure 36 Schematic of generic actuation workflow

In this regard, for the second iteration of the actuation framework, we update the actuators presented during the first iteration given the refinements of the overall SliceNet architecture as well as a better understanding of the Cognition Plane architecture/components and role. Nevertheless, we still keep the two families of actuations: workflow-based and function-based. For the workflow-based actuators, both the PF and the QoE Optimizer are the main components at WP5 level that govern the actuation. On the other hand, for the function-based actuation, we consider the specific case of Open Virtual Switch (OVS)-based traffic classification, enforcing specific traffic control rules to the slices at flow level thanks to the extended functions ingrained directly onto the OVS instances along the provisioned data-paths of the concerned slices. Table 23 provides a summary of the considered actuators, their scope and targeted scenario, the main involved components across the SliceNet logical architecture as well as marking if the presented actuator has been updated relative to the previous iteration or if it consists on a new actuation introduced in the current iteration

Table 23 Summary of actuators

Name	Short description	Approach	Scope	Main involved components	Update status/New
QoS modification	Modify QoS parameters associated to the E2E NS or specific NSSes	Workflow -based	Slice; sub-slice	QoE Optimizer; Policy Framework; QoE plugin; Service/slice Orchestrator	Updated; modified main point of enforcement; introduced policy definition
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	Updated; introduced policy definition; refined algorithm for reliable NSP selection

VNF scaling	Scales the resources associated to a VNF instance inside the service-chain of the E2E NS	Workflow-based	Slice	QoE optimizer; Policy Framework; QoE plugin; Service/Slice Orchestrator	New; introduced in iteration 2
VNF migration	Migrates from one VNF instance from one NFV PoP/server to another NFV PoP/server	Workflow-based	Slice	QoE optimizer; Policy Framework; QoE plugin; Service/Slice Orchestrator	New; introduced in iteration 2
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	No updates for the current iteration

4.2 QoS modification

4.2.1 Architecture and description of components

Figure 37 depicts the components involved in the QoS modification actuator.

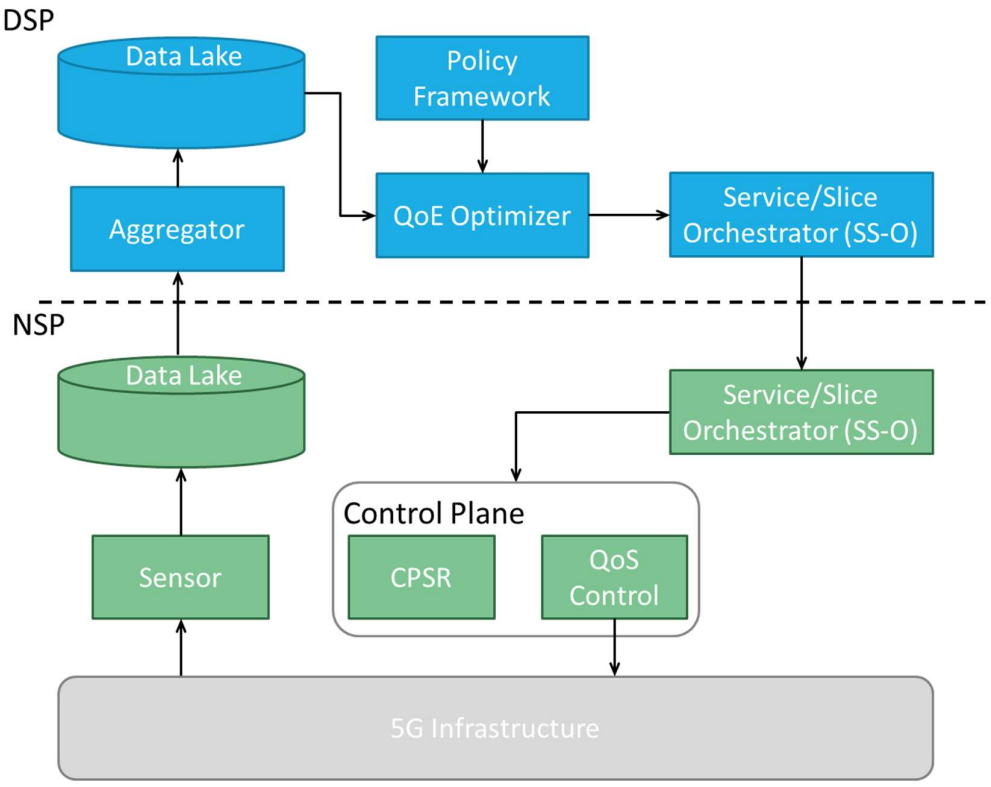

Figure 37 Logical architecture of the QoS modification actuator

The QoS Modification involves both levels, NSP and DSP, with monitoring information flowing from the **5G Infrastructure** to the NSP **Data Lake** thanks to specific **Monitoring Functions** gathering monitoring data (metrics, counters and alarms) of the NSSes deployed onto the NSP infrastructure. The **Aggregator** function at the DSP collects the information coming from the NSP, processes it to derive E2E NS data and stores it at the DSP **Data Lake**. This information is consumed by the **QoE Optimizer**, which triggers the actuation based on the policies disseminated from the **PF**. To achieve the desired actuation, the Service Slice Orchestrator (**SS-O)** at the DSP is contacted, which then will contact the corresponding **SS-O** at the NSP level, enforcing the desired (re-)configuration onto the **5G infrastructure** through the **QoS Control** function of the corresponding NSSes.

4.2.2 Design of components

The current iteration design is based on the design provided in the previous iteration, with some slight modifications to reflect the updates in the way in which information is consumed from the Data Lake, the format of the policies and the enforcement point for actuations, which for the second iteration is only the DSP orchestration system (i.e. the SS-O). Meanwhile, Figure 38 depicts a diagram of the software architecture and design followed in the updated implementation of the QoE Optimizer module.

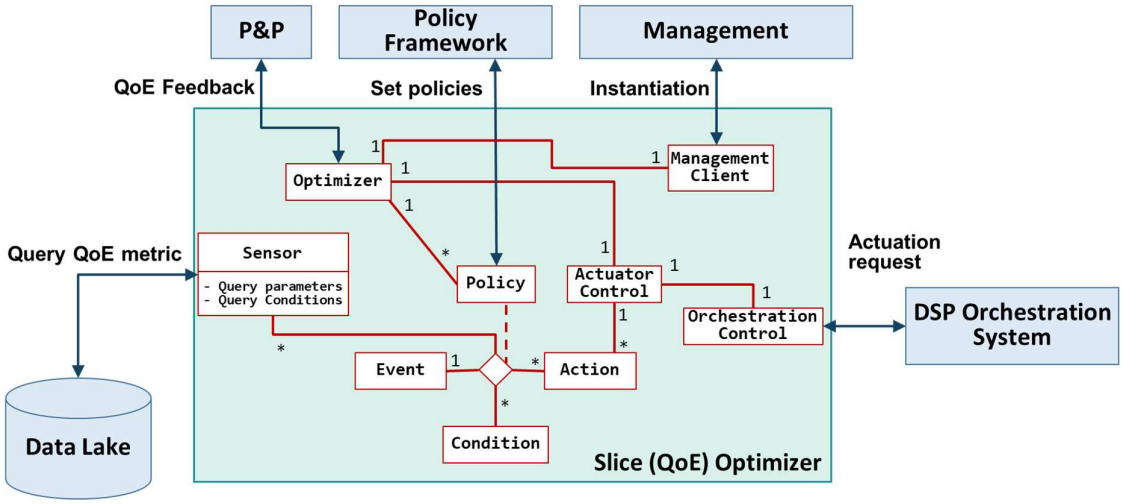

Figure 38 Software design and class diagram of the QoE Optimizer

New to the current iteration is the definition of a **Sensor** class tied to a specific **Policy** object instance. In the previous iteration, sensors responsible to collect information from the Data Lake where defined independently from the concrete **Policy** object instance, although the definition of sensors was tied to the event that was defined for the polices. For the current iteration, sensors are directly tied to a policy, querying, aggregating and generating event data customized for that specific policy. Thus, a **Sensor** class is also defined and tied to the representation of a policy instance (i.e. the **Policy** class). In this regard, the **Sensor** class is the one that models the external stimulus (e.g. senor/ML model outputs, vertical QoE feedback) that trigger actuations given current instanced policies. To achieve such task, a list of query parameters and query conditions are defined inside the class, defining what information should be gathered from the Data Lake and how plus if some aggregation is required to produce the event represented by a policy.

In regards of the actuation part, the current design and implementation are focused on actuations enabled thanks to the orchestration system at the DSP level, which then coordinates all the necessary (re-)configurations across the multiple NSSes and NSPs to achieve the desired actuation. For this, the **Actuator Control** contacts the **Orchestration Control**, responsible for all actuations that require the intervention of SliceNet Orchestration plane (e.g. modify the sequence of NSPs, migrate VNFs, etc.).

The presented design is flexible and modular enough to be enhanced by actuations carried over other enforcement points. In such a case, a specific class would be defined to model the actuations carried out through the specific enforcement point. Then, the **Actuator Control** would contact the corresponding control class depending on the type of action needed, such as was the case in the previous iteration for CP-enabled actuations.

In terms of interfaces design, aside from the interface defined between the QoE plugin at the P&P controller and the QoE Optimizer instance specified in the previous iteration, in the current iteration we introduce the definition of the interface between the Data Lake and the QoE Optimizer (specifically, the **Sensor** class, which acts as the consumer of the Data Lake), the interface between the QoE Optimizer and the PF for gathering slice policies as well as several interfaces between the QoE Optimizer and the DSP orchestration system designed in the context of several actuators exercised for the current iteration. The definition and details of all these interfaces will be presented during Section 5.

Aside from the QoE Optimizer design, which is common for all workflow-based actuators, another key element which has been designed in iteration 2 is the policy related to the actuator itself, tying the information gathered from the Data Lake to the specific action that need to be carried out for events related to that data. The definition of the policies follows the format defined for the TAL Engine at the NSP level (see D6.6 [6]) While the overall structure of the policies is common for all actuators, the specific details, parameters and elements are particular for each one of them. As such, for all reported actuators, we present an example of the specific policy instance employed for the actuator. Meanwhile, Figure 39 reports the policy details for the QoS Modification actuator, more specifically, for the case related to the modification of the bandwidth assigned to the E2E NS.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns0:tal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="tal.cse.com" xsi:schemaLocation="tal.cse.com tal.xsd" OID="">
  <ns0:reaction>
    <ns0:diagnosis>
      <ns0:symptom OID="symptom.e2eBW">
        <ns0:analysis>
          <ns0:aggregation>
            <ns0:aggregationItem>
              <ns0:aggregationData>
                <ns0:sensor OID="sensor.e2eBW">
                  <ns0:output>
                    <ns0:parameter>
                      <ns0:name>nssId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>nssBw</ns0:name>
                    </ns0:parameter>
                  </ns0:output>
                </ns0:sensor>
              </ns0:aggregationData>
              <ns0:threshold comparison="equal">
                <ns0:level>
                  <ns0:parameter>
                    <ns0:name>nssId</ns0:name>
                    <ns0:value>{nssId1}</ns0:value>
                  </ns0:parameter>
                </ns0:level>
              </ns0:threshold>
              <ns0:threshold logical="or" comparison="equal">
                <ns0:level>
                  <ns0:parameter>
                    <ns0:name>nssId</ns0:name>
```

```
                    <ns0:value>{nssId2}</ns0:value>
                  </ns0:parameter>
                </ns0:level>
              </ns0:threshold>
              <ns0:threshold logical="or" comparison="equal">
                <ns0:level>
                  <ns0:parameter>
                    <ns0:name>nssId</ns0:name>
                    <ns0:value>{nssId3}</ns0:value>
                  </ns0:parameter>
                </ns0:level>
              </ns0:threshold>
              <ns0:aggregationRule>
                <ns0:ruleItem>
                  <ns0:min>
                    <ns0:parameter>
                      <ns0:name>nssBw</ns0:name>
                    </ns0:parameter>
                    <ns0:period duration="00:01:00Z" />
                  </ns0:min>
                </ns0:ruleItem>
                <ns0:metric>
                  <ns0:name>
                    <ns0:parameter>
                      <ns0:name>minE2EBW</ns0:name>
                    </ns0:parameter>
                  </ns0:name>
                  <ns0:dimensions>
                    <ns0:parameter>
                      <ns0:name>{nssId}</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>{nssBw}</ns0:name>
                    </ns0:parameter>
                  </ns0:dimensions>
                </ns0:metric>
              </ns0:aggregationRule>
            </ns0:aggregationItem>
          </ns0:aggregation>
          <ns0:threshold comparison="lessOrEqual">
            <ns0:level>
              <ns0:parameter>
                <ns0:name>minE2EBW</ns0:name>
                <ns0:value>10</ns0:value>
              </ns0:parameter>
            </ns0:level>
          </ns0:threshold>
        </ns0:analysis>
      </ns0:symptom>
      <ns0:causes>
        <ns0:cause likelihood="0.99" causeOID="cause.e2eBW" />
      </ns0:causes>
    </ns0:diagnosis>
    <ns0:tactic>
      <ns0:cause causeOID="cause.e2eBW" />
      <ns0:action>
        <ns0:actionOption operation="increaseE2EBW">
          <ns0:actuator OID="actuator.dsp.sso">
            <ns0:configuration>
              <ns0:parameter>
                <ns0:name>nssId</ns0:name>
```



```
                    <ns0:value>{nssId}</ns0:value>
                  </ns0:parameter>
                </ns0:configuration>
              </ns0:actuator>
            </ns0:actionOption>
          </ns0:action>
        </ns0:tactic>
      </ns0:reaction>
</ns0:tal>
```

Figure 39 Example of a policy instance for the QoS modification actuation

The exemplified policy follows the structure defined previously, which can be modelled into several macro-structures, namely Sensor, Event, Condition and Action. For the Sensor part, an aggregation function is defined at the DSP level, which queries information from the Data Lake in regards of bandwidth utilization for the E2E slice. More specifically, several other aggregation functions are defined per deployed NSS, each one of them querying the current bandwidth of the NSS, which has been exposed by the multiple NSPs and persisted onto the DSP Data Lake. Given such information, the aggregation function at the DSP level determines the E2E bandwidth of the NS as the minimum among all bandwidth values extracted for the multiple NSP, since the NSS that has the least bandwidth determines the bottleneck for the E2E NS. For the Event part, the event that triggers the policy is precisely the aggregated bandwidth information processed by the aggregation functions (i.e. the Sensor). Given this event, the Condition part is represented by the comparison of the information carried by the event to a certain threshold (for the exemplified policy, the condition is less or equal to a bandwidth threshold). Lastly, the Action part carries both the information about the operation that should be carried once the actuation is triggered (increaseE2EBW in the example) as well as the end-point inside the SliceNet ecosystem that need to be contacted to enforce the operations associated with the Action, in this example, the DSP orchestrator (actuator.dsp.sso).

4.2.3 Implementation details

Given the previously described design, we experimentally evaluated the overall workflow of the QoS Modification. To this end, a small experimental set-up has been implemented to exercise the main components involved. The experimental set-up is based on iteration I, updating the interactions between modules given the refinements of the SliceNet overall architecture and the Cognition Plane. It focuses on exercising a QoS Modification operation due to a poor QoE feedback received from the vertical customer of the deployed slice. Thus, as a reaction to the received feedback, an increase of the bandwidth of the deployed slice will be triggered. This increase on the bandwidth will be enforced onto the RAN segment of the slice, which is the focus of the PoC reported. Given that, Figure 40 depicts a schematic of the employed experimental set-up.

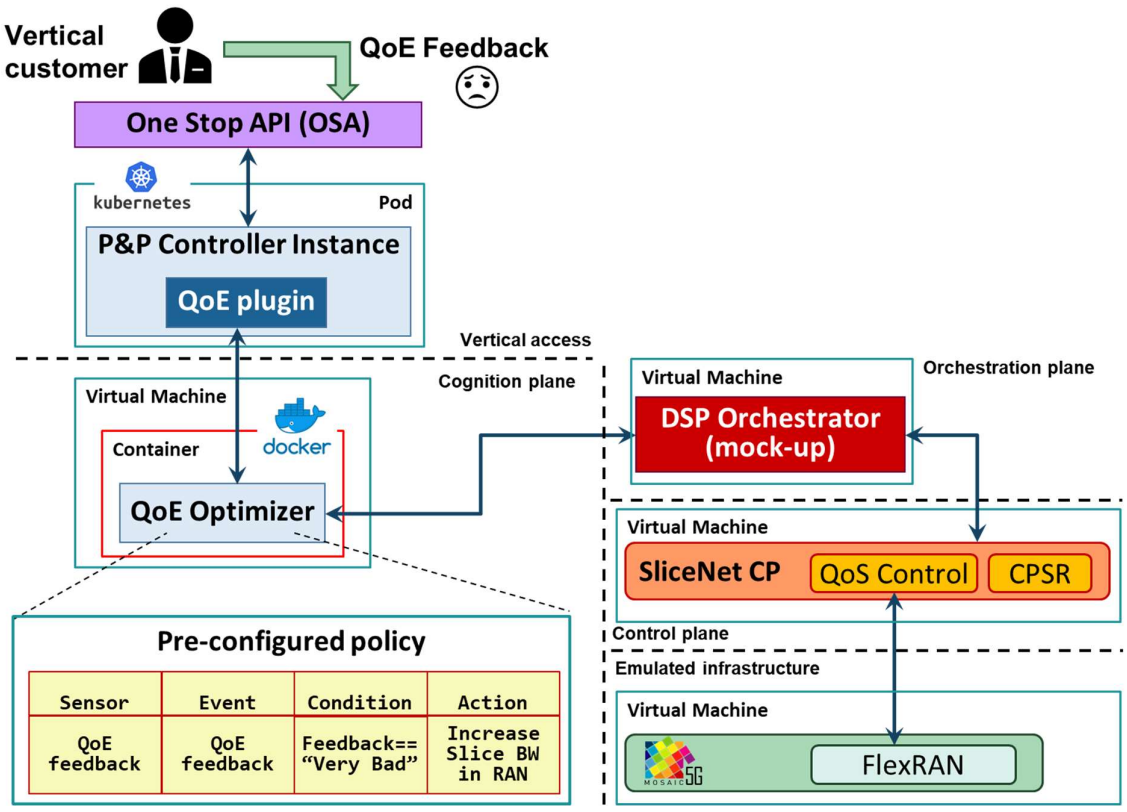

Figure 40 Experimental set-up for the demonstration of the QoS modification actuator

In essence, the experimental set-up consists in an instance of the OSA, which enables the access of the vertical customer to the SliceNet ecosystem. Then, a Kubernetes pod in which an instance of the SliceNet P&P controller has been deployed, together with an instance of the QoE plugin is also considered. 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. For the experimental evaluation of the actuation workflow, a pre-configured policy is present in this instance of the optimizer, with the concrete details of the policy being depicted in the figure, highlighting its macro-components (ECA and Sensor) plus a high-level description of its parameters. The policy follows the aforementioned policy structure for the QoS Modification actuator, with the particularity that the sensor is tied to the direct QoE feedback information received from the vertical, 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, a mock-up version of the DSP orchestrator is employed, which is contacted once the actuation needs to be triggered. Once contacted, it forwards the petition of an increase of bandwidth in the slice towards an instance of the SliceNet CP, consisting both in a Control Plane Service Registry (CPSR) instance plus a QoS Control service instance, responsible for QoS modification for the deployed RAN slice. Lastly, an emulated infrastructure consisting in a RAN network segment based on Mosaic 5G FlexRAN [17], 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 DSP orchestrator, which, in turn, contacts the QoS Control function at the CP, enforcing the increase of bandwidth onto the FlexRAN-based RAN segment.

4.3 NSP sequence modification

4.3.1 Design of components

Besides the design of the QoE Optimizer module, for the NSP sequence modification actuator, an algorithm to select the new sequence of NSPs in which the E2E NS should be supported has been designed. The context of the algorithm is the same as defined in iteration I, that is, the provisioning of an E2E NS with reliability guarantees. To this end, a reliability score **r** and a cost related to its selection are associated to each one of the NSPs. Given these parameters, the current iteration further refines the proposed algorithm for the selection of NSPs to achieve the desired reliability once a modification of the NSP sequence is required. Figure 41 depicts a pseudo-code for the designed algorithm.

Pseudo-code 1: Reliable NSP sequence selection

Input: Original NSP sequence, faulty NSPs, targeted reliability r, neighbouring and technology information
Output: Sol *//Solution (New NSP sequence)*

1: $G_n \leftarrow$ Reachability graph representing the multiple NSPs and their neighbouring relationships
2: Delete nodes representing faulty NSPs from G_n
3: Determine reliable NSP spans S from original NSP sequence
4: $N \leftarrow$ Set of NSPs present in S
5: **For** i=1 **to** |S|-1 **do**
6: $h_i \leftarrow$ number of NSPs between span i and i+1 in the original sequence
7: $R_i \leftarrow$ Calculate all routes from span i to span i+1, avoiding NSPs in N and with a maximum path length of h_i
8: $P \leftarrow$ Set of combined paths from previous routing operations, avoiding node repetition in the path
9: **For** $p \in P$ **do**
10: suitable$\leftarrow$ Check if nth NSP in p is suitable for nth NSS
11: **If** not suitable **then**
12: discard p
13: **If** suitable **then**
14: $r_p \leftarrow$ product of reliability scores of all NSPs in p
15: $cost_p \leftarrow$ summation of costs of all NSPs in p
16: **If** $r_p < r$ **then**
17: discard p
18: Sort paths in P in descending order according to $r_p/cost_p$. In case of equal value, higher reliability $r_p$takes precedence. In case of equal reliability, lower cost $cost_p$ takes precedence.
19: Select first path of the set as the solution (i.e. the new NSP sequence), subject to resource availability

Return Sol

Figure 41 Pseudo-code for reliable NSP selection algorithm in NSP sequence modification actuator

In more details, the steps of the algorithm are summarized as:

1. First of all, the algorithm receives as input the original sequence of NSPs currently employed to support the E2E slice, the NSPs that has been deemed as faulty or unreliable (according to sensor information extracted from the DSP Data Lake), the targeted reliability for the E2E NS and the neighbouring information as well as available technologies (the network segments supported) for all NSPs. Such information is extracted from the different inventories present at the DSP level, which agglutinate the characteristics of the NSPs under its supervision. Then, as an output of the algorithm, the new NSP sequence will be produced.

2. As a first step, an auxiliary graph consisting on the NSPs as nodes in the graph and links between nodes stating that the two NSP have reachability between them (i.e. they are neighbours) is constructed. Following the construction of the graph, nodes representing faulty/unreliable NSPs are removed from the graph as well as the links associated to them. In this way, the graph only represents NSPs that are still suitable to support a reliable E2E NS.
3. One of the goals of the algorithm is to minimize the needs of re-orchestrating NSSes deployed at the multiple NSPs. For this, it is important to maintain in the sequence NSPs that are still reliable from the original NSP sequence, also keeping their position in the sequence. In this way, the number of NSSes that will need to be re-instantiated/orchestrated will be minimized. To this goal, the algorithm determines the still reliable NSP spans from the original NSP sequence, that is, the sub-sequences of NSPs that contain NSPs suitable for the provisioning of a reliable E2E NS. To put an example, let us imagine that the original NSP sequence is $NSP_1 \rightarrow NSP_2 \rightarrow NSP_3 \rightarrow NSP_4 \rightarrow NSP_5$ and that NSP_3 is deemed as faulty. Given such scenario, the reliable NSP spans would be $NSP_1 \rightarrow NSP_2$ and $NSP_4 \rightarrow NSP_5$.
4. Once the spans are determined, the algorithm searches for the required NSPs to complete the full sequence, maintaining the previously determined spans. To do so, the algorithm treats the selection of NSPs as a routing problem over the constructed graph containing the still reliable NSPs. Indeed, finding a route over the graph translates to finding a sequence of NSPs that can potentially satisfy the reliability requirements of the E2E NS. To find said paths, the algorithm first searches for all available paths between end-points of the NSP spans previously determined, limiting the paths to avoid nodes present on other spans (to avoid loops) as well as to a hop count that will result in the same number of NSPs globally once the full sequence is determined.
5. When the routes between spans are determined, the E2E paths across the graph are determined, combining the spans and the routes between them. In this process, E2E combinations that would result on paths with node repetitions (i.e. NSP) are not considered. Once this process is done, the result is the set of all paths (i.e. NSP sequences) that satisfy the reachability requirements among NSPs as well as the number of required NSSes for the E2E NS.
6. For all the determined paths, the algorithm checks if the resulting NSP sequence would satisfy the technological requirements of the NS, that is, if NSP n is suitable for the deployment of NSS n. This being the case, the algorithm computes the reliability of the resulting sequence as the product of reliabilities for all involved NSPs and the cost of the sequence as the summation of the cost of each NSP. Otherwise, the path (hence the sequence) is discarded. Moreover, during the reliability calculation, if the determined reliability is lower than the target reliability r, the path is discarded.
7. After having curated the candidate paths, the algorithm sorts the set in a descending order according to the ratio of their reliability scores and their cost. In this way, paths with a good trade-off between reliability and associated cost are prioritized. In the case that multiple paths have the same value for the calculated ratio, the path with the highest reliability score is prioritized. Similarly, in the case that multiple paths have the same reliability score, the path with the lowest associated cost is prioritized.
8. Once the paths are sorted, the first path in the set is selected as the solution, that is, the sequence of NSPs to be employed for the re-insanitation/re-orchestration of the E2E NS, subject to the availability of resources for the selected NSPs.

The output of the algorithm will be used at the DSP orchestration system as the input to start the actions required to enforce the NSP sequence modification actuation.

Aside from the presented algorithm, and continuing with the definition of the policies employed for each one of the implemented actuators, in the following we present an example of a policy exercising the NSP Sequence Modification actuator, contextualized for the case of reliable NS provisioning. Figure 42 depicts the schematic of the policy.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns0:tal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="tal.cse.com" xsi:schemaLocation="tal.cse.com tal.xsd" OID="">
```



```
  <ns0:reaction>
    <ns0:diagnosis>
      <ns0:symptom OID="symptom.NSPreliability">
        <ns0:analysis>
          <ns0:aggregation>
            <ns0:aggregationItem>
              <ns0:aggregationData>
                <ns0:sensor OID="sensor.NSPreliability">
                  <ns0:output>
                    <ns0:parameter>
                      <ns0:name>nspId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>reliabilityScore</ns0:name>
                    </ns0:parameter>
                  </ns0:output>
                </ns0:sensor>
              </ns0:aggregationData>
              <ns0:threshold comparison="lessOrEqual">
                <ns0:level>
                  <ns0:parameter>
                    <ns0:name>reliabilityScore</ns0:name>
                    <ns0:value>7</ns0:value>
                  </ns0:parameter>
                </ns0:level>
              </ns0:threshold>
            </ns0:aggregationItem>
          </ns0:aggregation>
        </ns0:analysis>
      </ns0:symptom>
      <ns0:causes>
        <ns0:cause likelihood="0.99" causeOID="cause.NSPreliability" />
      </ns0:causes>
    </ns0:diagnosis>
    <ns0:tactic>
      <ns0:cause causeOID="cause.NSPreliability" />
      <ns0:action>
        <ns0:actionOption operation="changeNSP">
          <ns0:actuator OID="actuator.dsp.sso">
            <ns0:configuration>
              <ns0:parameter>
                <ns0:name>nspId</ns0:name>
                <ns0:value>{nspId}</ns0:value>
              </ns0:parameter>
            </ns0:configuration>
          </ns0:actuator>
        </ns0:actionOption>
      </ns0:action>
    </ns0:tactic>
  </ns0:reaction>
</ns0:tal>
```

Figure 42 Example of a policy instance for the NSP sequence modification actuation

For the presented example, an aggregation function (i.e. a sensor) is deployed, which is in charge of retrieving from the DSP Data Lake the reliability information exposed by the corresponding NSP. Given such information, a condition is defined in which if the reliability of the NSP is lower or equal than a certain threshold, then an actuation is required, defined by the field operation (changeNSP). As explained, and following the overall design for all workflow-based actuators at DSP level, once the actuation has been triggered, the DSP orchestrator end-point is contacted (actuator.dsp.sso).

Following the example, once the orchestrator is contacted, it will employ the defined algorithm to determine the new sequence of NSPs that satisfies the reliability requirements of the E2E NS.

4.4 VNF scaling

4.4.1 Demonstrated functionality

Aside from network data path resources (both physical and virtual), the other major components when provisioning an E2E NS are the VNF instances associated to it. Such VNFs may encapsulate specialized vertical functions that fulfil multiple roles within the vertical's service/application space or may implement network functions that enrich the capabilities of the NS (e.g. traffic shaping, firewalling, encryption), enhancing its performance. As such, it is important to always maintain the performance of deployed VNFs at their peak level to assure an optimal quality for the overall NS.

One of the KPIs for VNFs relates to the load of the virtual resources supporting them (i.e. the VMs). In order to maintain good performance levels, enough computational resources should be assigned to VNFs so they can perform their tasks optimally. Given this premise, NFV PoPs usually implement scaling functionalities on their VIM sub-systems to enable the provisioning of extra resources to VNFs in order to maintain their performance. For example, given the CPU usage of a VNF instance, once the utilization reaches a certain value, it may be necessary to provision more CPU resources so as to avoid potential bottlenecks if the CPU load increases further in the future. Such fluctuations on the utilization of the computational resources may be intrinsic to the tasks that the VNF needs to perform or a consequence derived from an increase of the traffic flowing through the service chain. As such, solutions that proactively scale the resources of VNF instances to keep up with their required utilization are a must.

While reactive or preventive scaling of a VNF instance may be relatively easy when only inspecting the resources consumed by the instance, it becomes a more complex task to achieve once the totality of the service chain (thus, the NS) is considered. For instance, an increase of traffic or users on a far apart network segment (e.g. a RAN segment) may have as a consequence an increased load at several VNF instances processing the data associated to the traffic/users. Such global events are hard to predict if load inspection is only performed locally, thus, incorrect or defective measures may be carried out to balance the increase of the load, affecting the QoE of the NS. As such, an analysis considering the whole NS perspective to detect if a VNF requires a scaling operation is essential to keep the desired quality levels.

Given this goal in mind, the VNF scaling actuator is designed to enable the scaling of VNF instances as a result of collected data at the DSP level. Such data may result of information directly exposed by the NSPs supporting the VNFs or as elaborated data inserted at the DSP Data Lake coming from the insights performed by multiple analytic functions. Utilizing this data as input, and with the proper policies in place, the QoE Optimizer will trigger the scaling of VNF instances whose performance is being affected by current load conditions. To enforce the scaling of VNFs, the QoE Optimizer contacts the DSP orchestrator (more in particular, the SS-O), which then will coordinate the scaling of VNF instances with the corresponding NSPs responsible for their management. At the NSP level, their orchestration system then contacts the VIM of the NFV PoP to trigger the scaling of VNFs.

4.4.2 Scenario

The general scenario that exercises the workflow of the VNF scaling actuator includes the steps depicted in Figure 43. Like previously described actuators in iteration I, the workflow is also divided in three macro-steps, for which the first two are in essence the monitoring from the NSP 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 VNF (or multiple ones) is suffering due to being overloaded, thus affecting the overall NS QoE,

the actuation is to modify the associated resources to the affected VNFs, that is, to scale them. To enforce such action, the QoE Optimizer contacts the Orchestrator at the DSP level, which then contacts the orchestration system of the affected NSP. The orchestrator system at the NSP level will then contact the VIMs of NFV PoPs under its responsibility, particularly the ones that are managing the VNFs that need to be scaled (exemplified as VIM1 and VIM2 in the figure). The NSP orchestrator will indicate the new resources that need to be assigned to specific VNFs. With such information, the multiple VIMs will finally perform the scaling of the desired VNFs, restoring appropriate QoE levels at the E2E NS.

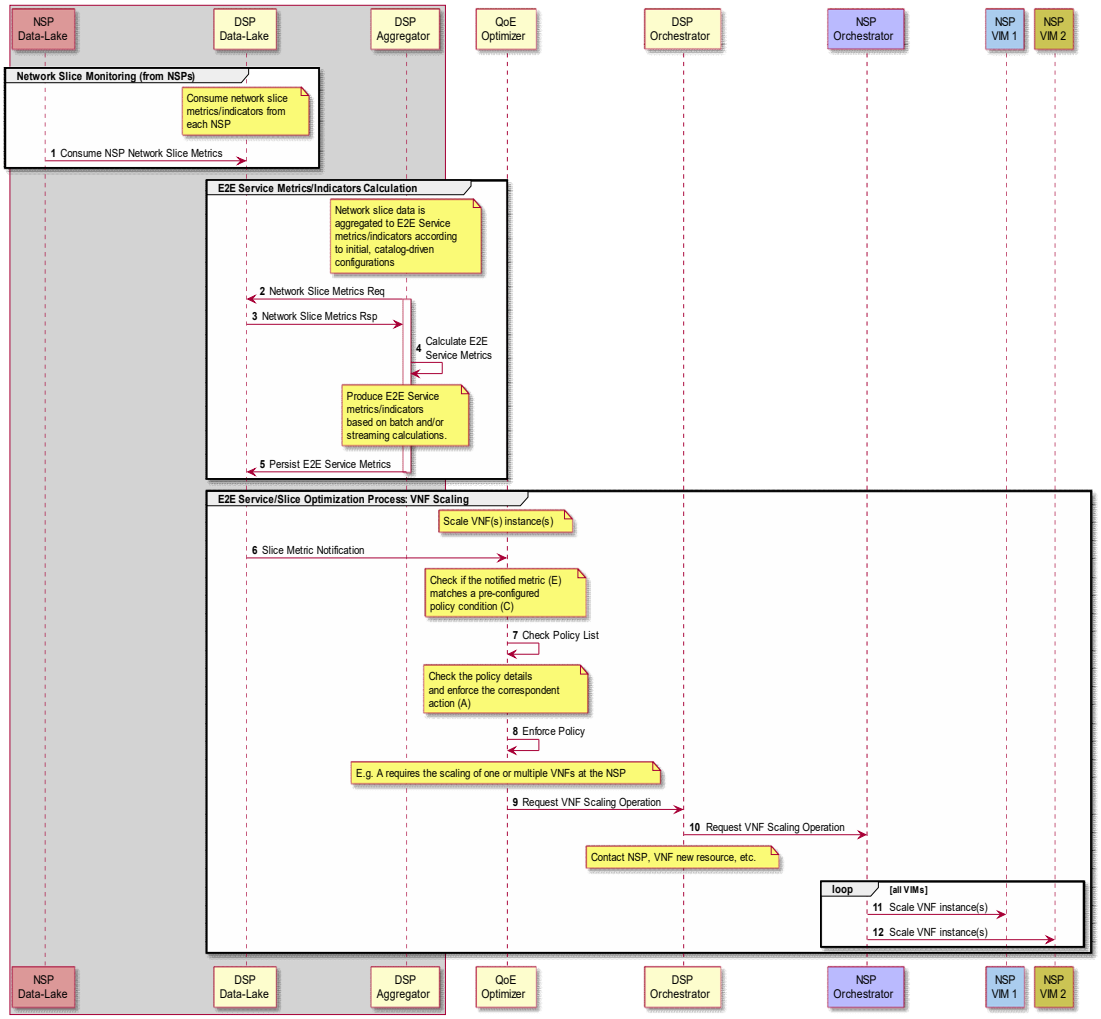

Figure 43 VNF scaling actuator general workflow

4.4.3 Architecture and description of components

With the presented scenario and scope in mind, Figure 44 depicts a schematic of the main components involved on the VNF Scaling actuator. The main difference with respect of the architecture presented in Section 4.2.1 is that, at the NSP level, the final enforcement point for the VNF Scaling actuator is not a CP service but instead the VIM responsible for the management of resources assigned to VNFs (e.g. VMs deployed in servers). As such, the NSP orchestration system will contact the VIMs of the affected NFV PoPs. Aside from this difference, the rest of the components is the same as described in the other actuators, with sensors extracting monitoring information from the physical/virtual infrastructure (VNFs instances in this case) and aggregation function processing and elaborating the data which then is stored at the Data Lakes of the multiple roles (DSP/NSP) to be utilized as inputs for the actuation system (QoE Optimizer and PF).

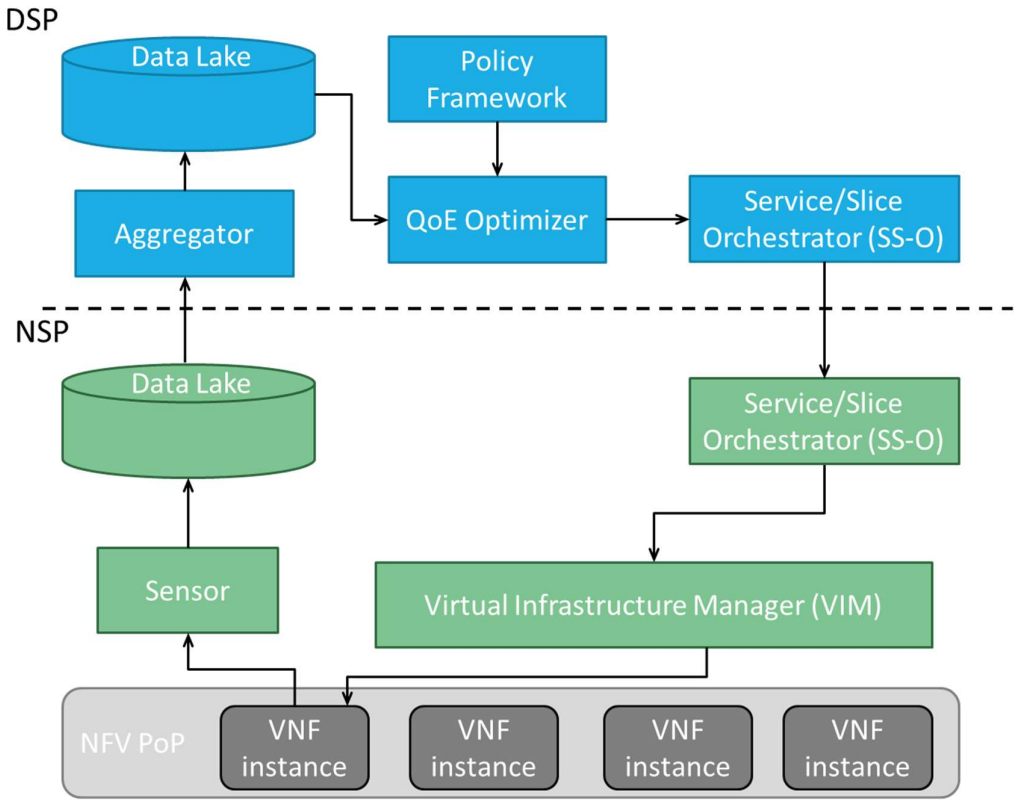

Figure 44 Logical architecture of the VNF scaling actuator

4.4.4 Design of components

In this sub-section we detail the design of the policy format employed for the VNF scaling actuator. Figure 45 depicts an example of the policy instance, particularized for the scenario of the NN analytical UC, as one of the potential actuations to be carried out due to the outputs produced by the ML model is the scaling of VNF instances due to determining that a particular VNF is in an overloaded state.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns0:tal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="tal.cse.com" xsi:schemaLocation="tal.cse.com tal.xsd" OID="">
  <ns0:reaction>
    <ns0:diagnosis>
      <ns0:symptom OID="symptom.VNFOverload">
        <ns0:analysis>
          <ns0:aggregation>
            <ns0:aggregationItem>
              <ns0:aggregationData>
                <ns0:sensor OID="sensor.VNFOverload">
                  <ns0:output>
                    <ns0:parameter>
                      <ns0:name>Time</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>SliceId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>VnfId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>Flavor</ns0:name>
```



```
                        </ns0:parameter>
                        <ns0:parameter>
                          <ns0:name>Type</ns0:name>
                        </ns0:parameter>
                      </ns0:output>
                    </ns0:sensor>
                  </ns0:aggregationData>
                  <ns0:threshold comparison="equal">
                    <ns0:level>
                      <ns0:parameter>
                        <ns0:name>Type</ns0:name>
                        <ns0:value>OVERLOAD</ns0:value>
                      </ns0:parameter>
                    </ns0:level>
                  </ns0:threshold>
                </ns0:aggregationItem>
              </ns0:aggregation>
            </ns0:analysis>
          </ns0:symptom>
          <ns0:causes>
            <ns0:cause likelihood="0.99" causeOID="cause.VNFOverload" />
          </ns0:causes>
        </ns0:diagnosis>
        <ns0:tactic>
          <ns0:cause causeOID="cause.VNFOverload" />
          <ns0:action>
            <ns0:actionOption operation="Scale">
              <ns0:actuator OID="actuator.dsp.sso">
                <ns0:configuration>
                  <ns0:parameter>
                    <ns0:name>SliceId</ns0:name>
                    <ns0:value>{SliceId}</ns0:value>
                  </ns0:parameter>
                  <ns0:parameter>
                    <ns0:name>VnfId</ns0:name>
                    <ns0:value>{VnfId}</ns0:value>
                  </ns0:parameter>
                  <ns0:parameter>
                    <ns0:name>Flavor</ns0:name>
                    <ns0:value>{Flavor}</ns0:value>
                  </ns0:parameter>
                </ns0:configuration>
              </ns0:actuator>
            </ns0:actionOption>
          </ns0:action>
        </ns0:tactic>
      </ns0:reaction>
    </ns0:tal>
```

Figure 45 Example of a policy instance for the VNF scaling actuation

For the presented example, the sensor part is related to a sensing function that determines if a VNF is in an overloaded state, thus requiring a scaling operation to maintain optimal performance. Aside from the state of the VNF (overloaded), the sensor defined at the aggregation function also carries information about the NS and VNF identifiers as well as the current flavour of the VNF instance (the assigned resources), which are required information in order to properly carry out the desired actuation: on one side, locate the VNF that needs to be scaled and in the other side, understand the current resources employed by the VNF so a proper scaling of them can be decided. Once the policy is triggered, as defined by the policy condition (the VNF is in an overloaded state), the QoE optimizer contacts the actuator end-point (actuator.dsp.sso), passing both the identifier of the VNF that needs

to be scaled as well as the name of the new flavour for the VNF. This information will be employed for the multiple orchestration systems (DSP and NSP) to instruct to the VIM responsible for the VNF instance to enforce the scaling of its resources according to the new resource profile (i.e. the flavour).

4.4.5 Results

Given the presented scenario and design, and following the experimental set-up explained during Section 3.3 for the NN cognition UC, in this section we present the results obtained in regards of VNF scaling. The experiments considered the presence of two VNF instances distributed in two different servers, which are then stressed following the approach previously presented in Section 3.3. As such, the load of the VNFs increases. This load, represented by the CPU consumption of the VNFs is collected by the developed ML model, which then may label the VNFs as overloaded. In such case, the event is stored at the Data Lake and consumed by the QoE Optimizer, which triggers the scaling of the VNF identified as overloaded. This scaling is performed by directly contacting the VIM that is managing the virtual infrastructure (i.e. the VMs) that is supporting the VNF instances, following the interfaces that are defined in Section 5.4. Figure 46 depicts the obtained results.

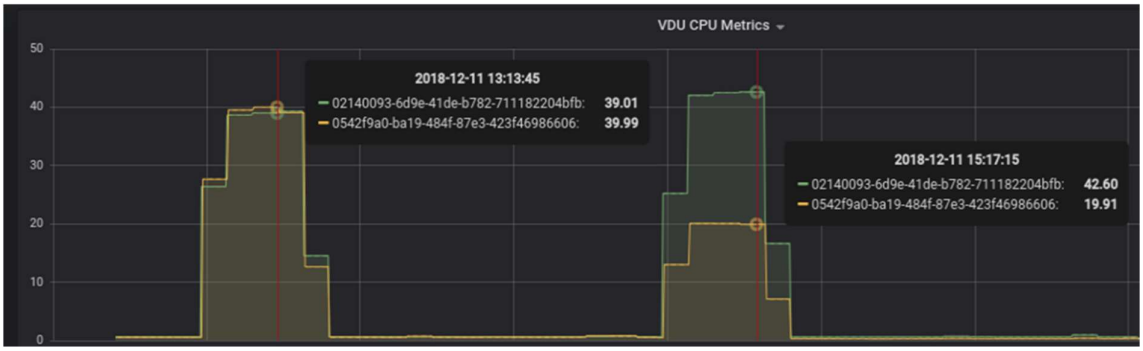

Figure 46 Monitored VNF CPU consumption and VNF scaling actuation

The depicted plots are the CPU consumption of the two VNF instances, which is collected thanks to the presence of a Kafka bus. Then, the obtained events are plotted to an instance of Grafana monitoring dashboard. It can be appreciated how the initial CPU consumption of the two VNFs is around 40%. For the reported experiments, VNF_1 (the orange plot) is labelled as overloaded. As a consequence, the QoE Optimizer decides to scale the VNF instance. In particular, the number of CPU cores assigned to the VNF is doubled, in order to ease the CPU burden. This can be appreciated in the plots, for which the CPU load of VNF_1 is reduced to the half (20%) thanks to the increased number of provisioned cores. Note that during the scaling period, the session between the VNF_1 instance and the Kafka bus is lost, hence the reason of zero values on the depicted plot. This is because for scaling the VM associated to the VNF, the VIM temporarily freezes the VM, creating a snapshot of it, which is then assigned with the new resources and re-activated again as the scaled (resized) VM.

4.5 VNF migration

4.5.1 Demonstrated functionality

As explained on the previous section, VNF instances are a central resource when deploying a NS that needs to support verticals' services and applications. Hence, keeping their performance at optimal levels is essential to guarantee the perceived QoE at the E2E NS. Performance degradations may be due to several factors, for instance, an increase of the traffic that needs to be processed by the VNF or changes on the resource consumption of other VNF instances collocated onto the same server, among other. As such, depending on the cause, it may be necessary to migrate the VNF instance from one server to another, or even between NFV PoP, to maintain optimal working conditions (more available resources or less noise from neighbouring VNFs). Other scenarios that may require the migration of

VNFs are scenarios in which end points (i.e. UCs) require some mobility across the provisioned E2E slice. In these cases, in order to keep acceptable latency figures, VNF instance may need to follow the end-points along the network segments in which they are traveling, thus requiring the migration of VNF instances from a remote NFV PoP to a closer NFV PoP to keep latency requirements.

The VNF migration actuator exercises precisely these situations. Given monitoring data extracted from the DSP Data Lake, it may be detected that the performance of a VNF instance is being affected, having a negative impact on the QoE, and that the remedy to overcome such situation (as stipulated on the policies in place) is to migrate the VNF instance from its current location to another. Once this is detected, the QoE Optimizer will contact the DSP orchestration system to instruct which VNF needs to be migrated. Then the Orchestration system will determine the new placement of the VNF, like in the PoC presented for the NN analytical workflow.

4.5.2 Scenario

The general scenario that exercises the workflow of the VNF Migration actuator includes the steps depicted in Figure 47. The workflow is very similar to the workflow of the VNF Scaling actuator, with the final result of enforcing the desired operation (in this case, the migration of a VNF) to the VIM of the NFV PoP in which the VNF is placed. In the case that migration is done within the same PoP, the VIM takes charge of the whole migration process, with the NSP orchestration system re-arranging (if needed) the associated data-path (both physical and virtual) to maintain the consistency of the service chain (i.e. the inter-VNF connectivity). As an extension of the single PoP scenario, the workflow also exercises the scenario in which the VNF needs to be migrated across different PoPs, being the most generic case the migration between NSPs. In this case, the DSP orchestration system needs to contact the orchestration system of all involved NSPs, taking charge of the coordination of the VIM system to achieve the migration of the VNF instance from one PoP to another. Moreover, such operation also entails the reconfiguration of the inter-PoP connectivity graph, that is, the connectivity between PoP involved on the provisioning of the E2E NS. As such, it is necessary to (re-)configure the path interconnecting the PoPs in the new disposition of the service chain. The function responsible for that configuration is the Inter-PoP Communication (IPC) service of the SliceNet CP. As such, once contacted and when the migration process between PoPs has been successful, the NSP orchestration system needs to contact the corresponding IPC service of the slice to enforce the new inter-PoP path configuration. To do so, first it needs to contact the CPSR to retrieve the instance of the service responsible for the NS/NSS at hand. Then, the (re-)configuration operation is requested to the retrieved service instance. This process needs to be repeated for all NSPs that have been involved in the migration to keep the consistency of the inter-PoP connectivity graph at all ends (i.e. NSPs).

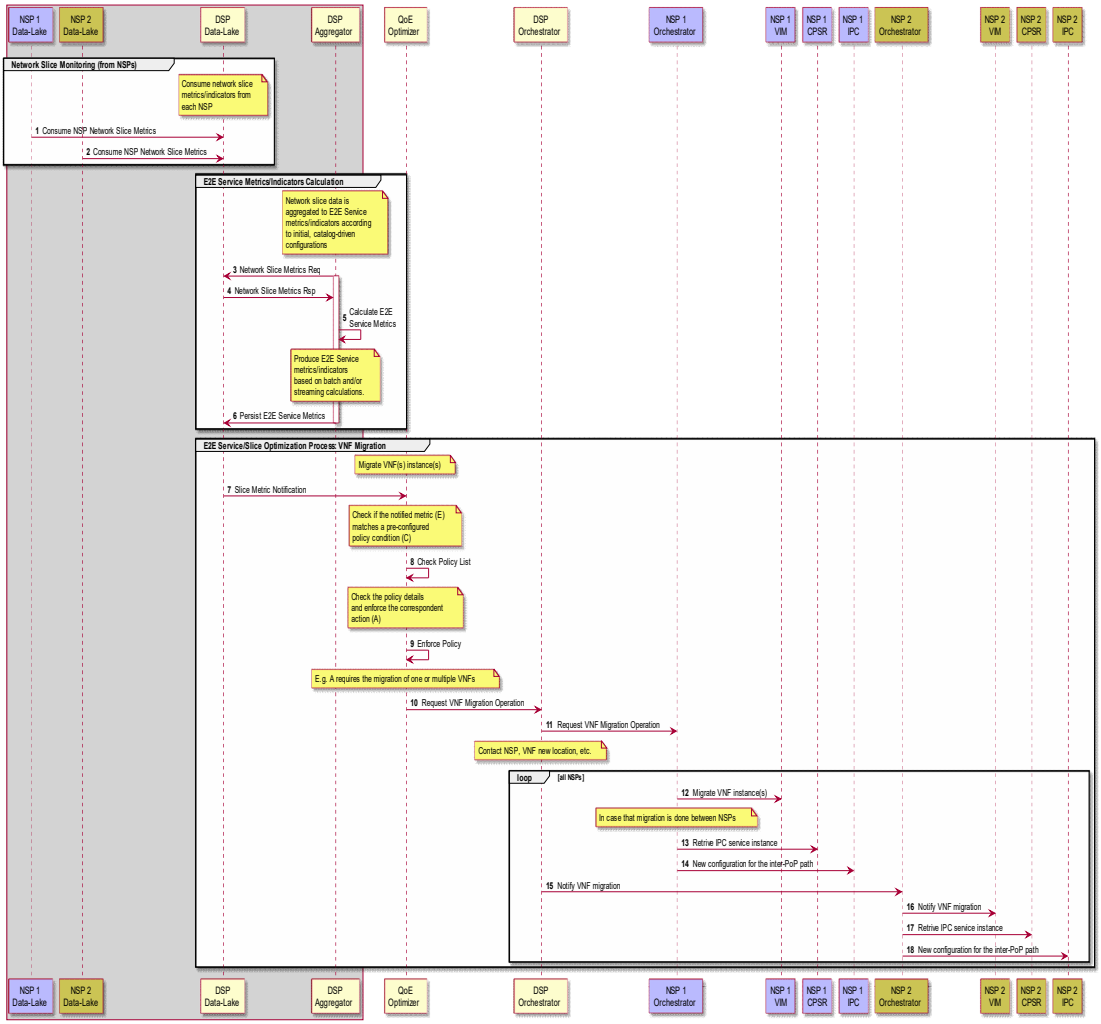

Figure 47 VNF migration actuator general workflow

4.5.3 Design of components

In this section, we present the design of the policy format for the VNF Migration actuator. As in the case of the VNF scaling actuator, the context of the depicted example is within the NN analytical UC. Thus, some of the fields of the policy are particularized for this specific technical UC. Nevertheless, the generic structure of the policy would be the same for other situations that require the migration of a VNF instance, with the main difference being the associated sensor (aggregation) function. Figure 48 depicts an example of the policy. As before, the policy defines a sensor in order to extract information from the DSP Data Lake (in this case, obtained through the NN ML model). For each data sample of the sensor, an event is defined, which then is contrasted with a condition, encoded in the threshold field of the policy. If the condition matches the defined cause of the policy (i.e. the defined trigger for actuation), then the QoE Optimizer will contact the enforcement end point defined inside the action block, more specifically, the actuator field (actuator.dsp.sso in the example). Several parameters, like the VNF identifier and the NS identifier, are encoded onto the parameters block of the policy, which will be passed done to the actuation end-point in order to execute the desired action (e.g. VNF migration).


```
<?xml version="1.0" encoding="UTF-8" ?>
<ns0:tal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="tal.cse.com" xsi:schemaLocation="tal.cse.com tal.xsd" OID="">
  <ns0:reaction>
    <ns0:diagnosis>
      <ns0:symptom OID="symptom.NoisyNeighbor">
        <ns0:analysis>
          <ns0:aggregation>
            <ns0:aggregationItem>
              <ns0:aggregationData>
                <ns0:sensor OID="sensor.NoisyNeighbor">
                  <ns0:output>
                    <ns0:parameter>
                      <ns0:name>Time</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>SliceId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>VnfId</ns0:name>
                    </ns0:parameter>
                    <ns0:parameter>
                      <ns0:name>Type</ns0:name>
                    </ns0:parameter>
                  </ns0:output>
                </ns0:sensor>
              </ns0:aggregationData>
              <ns0:threshold comparison="equal">
                <ns0:level>
                  <ns0:parameter>
                    <ns0:name>Type</ns0:name>
                    <ns0:value>NOISE</ns0:value>
                  </ns0:parameter>
                </ns0:level>
              </ns0:threshold>
            </ns0:aggregationItem>
          </ns0:aggregation>
        </ns0:analysis>
      </ns0:symptom>
      <ns0:causes>
        <ns0:cause likelihood="0.99" causeOID="cause.NoisyNeighbor" />
      </ns0:causes>
    </ns0:diagnosis>
    <ns0:tactic>
      <ns0:cause causeOID="cause.NoisyNeighbor" />
      <ns0:action>
        <ns0:actionOption operation="Migrate">
          <ns0:actuator OID="actuator.dsp.sso">
            <ns0:configuration>
              <ns0:parameter>
                <ns0:name>SliceId</ns0:name>
                <ns0:value>{SliceId}</ns0:value>
              </ns0:parameter>
              <ns0:parameter>
                <ns0:name>VnfId</ns0:name>
                <ns0:value>{VnfId}</ns0:value>
              </ns0:parameter>
            </ns0:configuration>
          </ns0:actuator>
        </ns0:actionOption>
      </ns0:action>
```

```
      </ns0:tactic>
   </ns0:reaction>
</ns0:tal>
```

Figure 48 Example of a policy instance for the VNF migration actuation

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 in the operations described across the deliverable. The focus here is on reporting the interfaces that have been designed or implemented during iteration 2, leaving the documentation of other interfaces and APIs for future developments on the overall Cognition Plane framework.

5.1 QoE Optimizer CP interface

In the previous iteration, SliceNet CP was considered as one of the enforcement points for actions related to actuations triggered from the QoE Optimizer. As such, an interface was designed to allow the QoE Optimizer to retrieve CP services from the CPSR as well as contacting the service instances to execute the desired operations. For the current iteration, SliceNet CP is no longer an enforcement point for actuators triggered from the QoE Optimizer, thus, the interfaces reported during iteration I are not exercised in the current implementation of the module. Nevertheless, the QoE Optimizer still needs to implement an interface with SliceNet CPSR. The reason behind this interface is that, aside from serving as an inventory for CP services, the CPSR also serves as an inventory for SliceNet modules/functions that are deployed per NS/NSS. In this way, the rest of modules within the SliceNet framework can locate easily the modules instances associated to a certain slice identifier whenever they need to contact them. For this reason, every deployed QoE Optimizer instance has to implement an interface with the CPSR that allows for registering the instance to the common registry. Having this in mind, in the following we detail the implementation of said interface. The interface is based on REST, with Table 24 reporting the specific details

Table 24 QoE Optimizer to CPSR interface

Endpoint	http://<IP>:<PORT>/slicenet/ctrlplane/cpsr_cps/v1/cps-instances/<serviceUUID> IP: address of the CPSR instance PORT: port of the CPSR instance serviceUUID: identifier of the service instance that is going to be registered onto the CPSR (e.g. the QoE Optimizer instance)
HTTP operation	PUT
Description	It enables to register a QoE Optimizer instance onto the CPSR specified in the endpoint
Caller	QoE Optimizer
Request body	JSON
Response body	None
Response code	200 OK – Service registered 400 Bad Request - A generic problem happened with the operation

The request body contains a **CPSProfile** object, which is a representation of the service instance to be registered to the CPSR (i.e. the QoE Optimizer, in this case) plus its capacities and properties. The specific format of the object has been documented in deliverable D4.3 [16], to which the interested reader can refer.

5.2 QoE Optimizer – Data Lake interface

Following the architecture design described along this deliverable, all the data that triggers actuations from the QoE Optimizer is being stored at the DSP Data Lake. The data is then associated to the sensors (aggregation) functions described at the instantiated policies. To gather the information elaborated by this sensor functions, the QoE Optimizer has to contact the entries from the Data Lake. As such, an interface has to be developed for this purpose. For the current iteration of the Cognition Plane, a first

version of the DSP Data Lake has been implemented, based on the InfluxDB technology. This implementation has been exercised in the context of the NN analytical workflow, in which events resulting from the output of the ML model are inserted into the DB. Then, the QoE Optimizer needs to retire the events from the DB. Following these principles, in the following we specify the design of the interface between the QoE Optimizer and the Data Lake (influxDB) along with the supported operations. First, Table 25 depicts the model of the events that are inserted into the influxDB, putting as an example the concrete model employed for the NN case. In general, events are inserted as standard entries in relational databases, with the addition of having an extra field, named **Instant**, which represents the point in time in which the event was inserted at the DB.

Table 25 Schematic of event entries at the InfluxDB-based Data Lake

Instant: an instantaneous point on the time-line. Represents the timestamp for the event once it is inserted in the DB	**Field1:** generic entry on the DB schema	**Field2**: generic entry on the DB schema	**…**	**FieldX**: generic entry on the DB schema
…	…	…		…
29/05/2019 14:37	tenantID	vnfID		flavour

As for the operations supported by the interface, they are summarized in Table 26. In essence, two types of operations are implemented. On one hand, the **openDataLake** method opens up a connection with the Data Lake end-point. This serves as enabler to perform queries in the schemas stored inside the InfluxDB. On the other hand, both methods **getEvents** and **getLastEvent** allow extracting event information from the desired schema, with the addition that the last method only extract the information of the event with the most recent timestamp. The type of information that is extracted depends on the concrete SQL query that is passed down as parameter when calling these methods. Thanks to the combination of both, it is possible to consult event information of collections of events or specific ones from any remote end-point, in this case, the QoE Optimizer module.

Table 26 QoE Optimizer to InfluxDB interface

Operation name	openDataLake
Description	opens a connection to the InfluxDB-based Data Lake, which will be used by other operations to perform their actions
Caller	QoE Optimizer
Parameters	dbURL: URL direction of the influxDB end-point
Operation name	getEvents
Description	extracts the events information stored in a specific schema inside the InfluxDB
Caller	QoE Optimizer
Parameters	dbName: name of the schema queryStr: string containing the SQL operation employed to select the desired information
Operation name	getLastEvent
Description	extracts the last inserted event inside a specific schema
Caller	QoE Optimizer
Parameters	dbNmae: name of the schema queryStr: string containing the SQL operation employed to select the desired information of the last inserted event

5.3 QoE Optimizer – Policy Framework interface

Following the web socket-based interface defined for the PAP at the PF in the previous deliverable D5.5, Section 5.1, in this section we describe the client side implemented by the QoE Optimizer, which enables the consumption of policies disseminated from the PF. Table 27 summarized the operations supported by the interface and their parameters. The interface implements the methods to enable the connection and disconnection from the PAP as well as the operation to collect whatever policy is notified from the PAP.

Table 27 QoE Optimizer to PF interface

Operation name	connect
Description	opens a connection to the PAP at the PF
Caller	QoE Optimizer
Parameters	URI: direction of the PAP in the format of wss://<IP>:<port>/pdp/notifications, in which IP is the IP address of the PAP and port is the port in which the PAP is listening for subscriptions from the multiple PDPs
Operation name	disconnect
Description	closes the connection to the PAP at the PF
Caller	QoE Optimizer
Parameters	None
Operation name	onMessage
Description	extracts the message notified from the PAP, i.e. the policies notifications, including updated policies, new policies and removed policies.
Caller	QoE Optimizer
Parameters	None

5.4 QoE Optimizer – OpenStack interface

In order to enable the execution of the two newly introduced workflow-based actuators, namely, VNF Scaling and Migration, it is necessary to introduce also the related interfaces that allow enforcing such actions towards the orchestration system. Because the SliceNet orchestration system is currently being developed, no mature end-point is available which could be contacted to enforce the actions associated with the actuators. As such, for the current iteration, specific interfaces towards an OpenStack-based VIM have been developed, which enable the scaling of a VM supporting a VNF as well as the migration of a VM. The interfaces are based on the API provided by the Nova service of OpenStack, for which the QoE Optimizer implements the client side. Although a client for the Keystone service has been also implemented in order to gain the credentials to interact with core services of OpenStack, like Nova, we only report the interfaces that enable the actions associated to the developed actuations. Such interfaces will be evolved during the development of SliceNet orchestration system (particularly, the SS-O at the DSP level), to not directly contact the VIM but to have the orchestration system as the unique end-point that needs to be contacted from the QoE Optimizer. The interface is based on REST, for which the details are provide in the following table.

Table 28 QoE Optimizer to OpenStack Nova interface

Endpoint	http://<IP>:<PORT>/v2/<project_ID>/servers/<vm_ID>/action IP: address of the QoE Optimizer instance PORT: port in which the QoE Optimizer is listening for QoE feedbacks project_ID: identifier of the OpenStack project hosting the VMs for which the operation is requested vm_ID: identifier of the VM into which the action is performed

HTTP operation	POST
Description	It enables the execution of an action specified by an **Action** object, whose parameters and structure depend on the type of action called
Caller	QoE Optimizer
Request body	JSON, see example in Figure 49
Response body	None
Response code	200 OK - Operation performed correctly 400 Bad Request - A generic problem happened with the operation

The **Action** object, for which Figure 49 details its structure and main fields for both VM resizing and VM migration actions, specifies both the desired action as well as the parameters. For the VM resizing action, the flavorRef parameter specifies the identifier of the new flavour (i.e. the resources) that has to be assigned to the VM. For the VM migration, the host field specifies the identifier of the host into which the VM needs to be migrated.

```
resize:
type: object
  properties:
    flavorRef: string
    example: “3b3888dc-3502-11e9-b210-d663bd873d93”


os-migrateLive:
type: object
  properties:
    host: string
    example: “3b3888dc-3502-11e9-b210-d663bd873d93”
```

Figure 49 Schematic of employed Action objects (resize and os-migrateLive)

6 Summary of new software components

In this section, we provide a summary of the main new software components presented along the deliverable through the multiple exercised analytical and actuation workflows. Links to their codebase can be found for each one of them.

Other WP5 software components, which were described in D5.5, have also been updated.

6.1 Anomaly detection experiment

Name	**Anomaly detection model**
Description	The model predicts the signal strength quality in the future 5 minutes by observing a set of QoS metrics for the last 5 minutes.
License	NA
Version	NA
Design	Section 3.6
Codebase	https://gitlab.com/slicenet/anomaly_detection

6.2 QoE Plugin

Name	**QoE Plugin**
Description	The QoE Plugin is the responsible to collect feedback data from the vertical user of the NS. Such data may be then directly processed by the QoE Optimizer to trigger the corresponding actuation or stored at the DSP Data Lake for its later processing by analysis functions to derive elaborated data, which, at its turn may serve as triggers for future actuations.
License	NA
Version	1.0
Design	NA
Codebase	https://gitlab.com/slicenet/qoe-plugin

6.3 QoE Library

Name	**QoE REST Client Library**
Description	The QoE REST Client Library provides a client API to contact the different SliceNet components, such as the PF, the orchestration system and the QoE plugin. Having the multiple interfaces as an external library facilitates the separation of core QoE Optimizer capabilities from the external modules that it needs to contact, allowing for an enhanced inclusion of new functionalities on both the core aspects as well as the client aspects of the QoE Optimizer module.
License	NA
Version	1.0
Design	Section 5
Codebase	https://gitlab.com/slicenet/qoe-rest-client

7 Conclusions

This deliverable has presented the design of the overall Cognition Plane within SliceNet's management layer and provided the second iteration for the design and implementation of the several elements that 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 NSes as offered by DSPs entities towards vertical customers. To this end, it is essential to provide means for analysing 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.

Following an agile approach, the current deliverable has updated the contents of the previous deliverable D5.5, expanding its contributions. As such, the efforts of the current iteration have been focused on the implementation, validation and integration of the several Cognition Plane components and interfaces/APIs. This has been demonstrated across the document through several PoCs and experimental set-ups, highlighting the obtained results and the integration efforts. In particular, this document has demonstrated the development and implementation of ML models that aid in determining the QoE of deployed NSes, as well as more specific cognition UCs, such as reliable RAN slicing, RAN optimization, anomaly detection in NSes with mobility (as it is the case of the eHealth vertical UC) and VNF noise detection (Noisy Neighbour). All these developed ML models and cognition UCs have demonstrated the capacities of the SliceNet Cognition Plane to aid in the management of QoE-aware 5G NSes, serving as stimulus for corrective measures. In this regard, the developed actuation framework (PF and QoE Optimizer) has shown the integration of several actuation workflows, aided by policies, that react to the inputs provided by QoE monitoring/ML models, executing the desired actions to achieve the necessary corrective measures, all governed through the developed Data Lake, which has been preliminary tested and integrated in the context of some of the cognitive UCs (e.g. the NN UC). As a result of all these efforts, MS4 is validated, which constitutes an important milestone towards the integration of the developed Cognition Plane towards the final SliceNet system prototype (to be performed in WP8).

The several components will be further developed and enhanced to be exploited by both WP6 and WP7. As such, the next deliverable of WP5 will extract the final conclusions and results that have come from the integration of the Cognition Plane and how these have helped evolve the cognitive management of 5G NSes.

References

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

[2] SliceNet Deliverable D8.4, “SliceNet System Integration and Testing (Iteration II)”, SliceNet Consortium, June 2019.

[3] “Introduction to InfluxData’s InfluxDB and TICK Stack”, September 2017. [online]: https://www.influxdata.com/blog/introduction-to-influxdatas-influxdb-and-tick-stack/

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

[5] SliceNet Deliverable D5.5, “Modelling, Design and Implementation of QoE Monitoring, Analytics and Vertical-Informed QoE Actuators - Iteration I”, SliceNet Consortium; March 2019.

[6] SliceNet Deliverable D6.6, “Single-Domain, Multi-Tenant Network Slice Management (Fault, Configuration, Accounting, Performance / SLA and Security)”, SliceNet Consortium; June 2019.

[7] Altice Labs Tensorflow ML algorithm [online]: https://wiki.ptin.corppt.com/display/AI/TensorFlow

[8] Altice Labs PyTorch ML algorithm [online]: https://wiki.ptin.corppt.com/display/AI/PyTorch

[9] Altice Labs Machine Learning Technologies [online]: https://wiki.ptin.corppt.com/display/AI/Machine+Learning+Technologies

[10] Altice Labs BenchmarkUtils application [online]: http://svn.ptin.corppt.com/repo/slicenet/trunk/notebooks/rfp/BenchmarkUtils.ipynb

[11] Apache Flink - Stateful Computations over Data Streams [online]: https://flink.apache.org/

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

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

[14] 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.

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

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

[17] 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.