Deliverable D4.3

Single-Domain, Multi-Tenant Network Slicing Control

Editor:	Ciriaco Angelo, Ericsson Telecomunicazioni SpA (TEI)
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date v1.0:	30/11/2018
Actual delivery date v1.0:	14/12/18
Required re-submission delivery date v1.1:	05/10/2019
Actual re-submission delivery date v1.1:	05/10/19
Suggested readers:	Network Administrators, Vertical Industries, Telecommunication Vendors, Telecommunication Operators, Service Providers
Version:	1.1
Total number of pages:	96
Keywords:	Network Slicing, Control Plane, Network Slice, 5G, SDN

Version information

This document is the new revised version 1.1 of the previous version 1.0.
Reason for revision is to include recommendations from SliceNet 2nd year review as described in the Review Report [44] .
Main changes are:

- New Sections **4.1.1.4** and **6.1.2** added to describe the CPSR High Availability Architecture and Prototype.
- Updated Section **5** to clarify IPv6 capabilities
- Updated Section **4.1.2** to clarify why the Inter-PoP Path descriptor is based on IP
- Updated Section **1** to align with current SliceNet evolved architecture.
- Minor updates in Section **2**, **3**, and **7** to align with current main technical evolutions.

Abstract

This document reports the design and prototype implementation of the SliceNet Control Plane Services and Backhaul Adapters for single administrative domain according to the SliceNet Control Plane architecture as defined in Deliverable D2.3. The Control Plane Services handle tasks for the enforcement of network functions, configuration rules and policies governing the run time operations of Radio Access Network (RAN), Core Network (CN), Mobile Edge Computing (MEC) and Backhaul network segments within the same administrative domain. The Service Based Architecture (SBA) principles applied to the SliceNet Control Plane are as well addressed describing how the Control Plane components offer or consume services communicating by an agreed Service Based Interface (SBI) over a common bus.

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] WP4- 5G Multi-Domain Slice Control Plane

[Number and title of task] T4.3 – Single-Domain Multi-Tenant Network Slicing Control

[Document title] Single-Domain Multi-Tenant Network Slicing Control

[Editor: Name, company] Ciriaco Angelo, Ericsson Telecomunicazioni SpA (TEI)

[Work-package leader:] Ciriaco Angelo, Ericsson Telecomunicazioni SpA (TEI)

Copyright notice

Executive summary

This document deals with the “Single-Domain Multi-Tenant Slicing Control” functionality according to the SliceNet Control Plane architecture as defined in Deliverable D2.3 [2] .

The functionality is thought to handle tasks for the enforcement of network functions, configuration rules and policies governing the run time operations of Radio Access Network (RAN) RAN, Core Network (CN) Core, Mobile Edge Computing (MEC) MEC and Backhaul network segments within the same administrative domain.

Main achievements with this deliverable are:

- Realization of a set of SW components called Control Plane Services (CPS) each offering specific configuration and control capabilities.
- Realization of a Service Based Architecture framework to accommodate the required SliceNet Control Plane components and describe how the SW components offer or consume services communicating by an agreed Service Based Interface (SBI) over a common bus.
- Prototyping of SW components and related documentation in terms of reference open source frameworks used and implementation choices. Most of the delivered components have been implemented from scratch as containerized application.

List of authors

Company	Author
TEI	Ciriaco Angelo, Carmine Galotto, Biagio Maione, Giacomo Borlizzi, Giuseppina Carpentieri, Anna Maria Santonicola
CSE	Konstantinos Koutsopoulos, Athanasios Kokkinis
NXW	Giacomo Bernini, Pietro G. Giardina
UPC	Fernando Agraz, Albert Pagès, Salvatore Spadaro
UWS	Pablo Salva Garcia, Antonio Matencio Escolar, Ruben Ricart Sanchez, Enrique Chirivella Perez, Ricardo Marco Alaez, Qi Wang, Jose M. Alcaraz Calero

Deliverable Reviewers

Company	Author
ECOM	Nasim Ferdosian
ORO	Marius Iordache
RZ	Ricardo Figueiredo

Table of Contents

Executive summary 4
List of authors 4
Deliverable Reviewers 4
Table of Contents 5
List of figures 8
List of tables 10
Abbreviations 12
1 Introduction 14
1.1 Scope 14
1.2 Document structure 14
2 State of the art 16
2.1 Service-based approach 16
2.2 Multi-segment technology-agnostic control 17
2.2.1 Backhaul and WAN 17
2.2.2 RAN 17
2.2.3 Core Network 18
3 Principles and architecture 19
3.1 Control Plane Services Principles and Architecture 19
3.2 Service Based Architecture 19
3.3 Technology agnostic abstraction 20
3.4 Control Plane Services Life Cycle Management 20
3.4.1 CPS Creation 22
3.4.2 CPS Configuration 23
3.4.3 CPS Termination 23
3.5 Registration and retention flow 23
4 Description of Single-Domain CP components 25
4.1 Control Plane Services 25
4.1.1 Control Plane Service Register 25
4.1.2 InterPop Connection Control Service 34
4.1.3 QoS Control Service 43
4.1.4 NF Config Control Service 51
4.2 Control Plane Adapters 53
4.2.1 Backhaul DPP Adapter 53
4.2.2 Backhaul Adapter 54

5 Interfaces 61
5.1 CP-CPSR-S 61
5.1.1 CPSR main data structure 61
5.2 CP-CONF-S 65
5.3 CP-QOS-S 66
5.3.1 Set QoS constraints 66
5.3.2 Set Priority 67
5.4 CP-IPC-S 68
5.4.1 Provision InterPoP Connections 68
5.4.2 Update InterPoP Connections 70
5.4.3 Remove InterPoP Connections 71
5.5 CP-RAN-A 73
5.6 CP-MEC-A 73
5.7 CP-CORE-A 73
5.8 CP-WAN-A 73
5.9 CP-BKHL-A 73
5.9.1 Provision SDN Intent 73
5.9.2 Update SDN Intent 75
5.9.3 Remove SDN Intent 76
5.9.4 Get SDN Topology 77
5.10 CP-BKHL-DPP-A 79
5.10.1 Parameters 79
5.10.2 Backhaul DPP Adapter Set/Remove Priority 81
5.10.3 Backhaul DPP Adapter Set/Remove Min BW 81
5.10.4 Backhaul DPP Adapter Set/Remove Max BW 81
6 Prototype 83
6.1 CPSR 83
6.1.1 CPSR, no High Availability Architecture 84
6.1.2 CPSR, High Availability Architecture 85
6.2 QoS Control Service 86
6.2.1 QOS prototype and test environment description 87
6.3 IPC 88
6.3.1 IPC prototype and test environment description 89
6.4 NF Config Control Service 89
6.5 Adapters 90
6.5.1 Backhaul Adapter 90

6.5.2 Backhaul DPP Adapter 91
6.6 Network Controllers 92
6.6.1 ONOS 92
6.6.2 FCA Controller 92
7 Conclusions 94
8 References 95

List of figures

Figure 1.1 Intra-Domain Slicing within the SliceNet logical architecture 14
Figure 3.1 SliceNet Control Plane Services architecture 19
Figure 3.2 CPS Life-cycle management steps 22
Figure 3.3 Registration and Retention flow 23
Figure 4.1 request/response and subscribe/notify 25
Figure 4.2 registry and discovery 26
Figure 4.3 registration operation 27
Figure 4.4 update operation 28
Figure 4.5 partial update operation 28
Figure 4.6 Heart-Beat 29
Figure 4.7 Deregistration operation 29
Figure 4.8 Subscription to new registered CPS instances 30
Figure 4.9 Subscription to change of CPS profile 30
Figure 4.10 Status Notify 31
Figure 4.11 Status UnSubscribe (subscriptionID) 31
Figure 4.12 Status UnSubscribe (cpsSubscriptionID) 31
Figure 4.13 CPS Discovery 32
Figure 4.14 Authorization Flow 33
Figure 4.15 CPSR High Availability Architecture 34
Figure 4.16 Backhaul Network 35
Figure 4.18 Provision InterPoP Connections 36
Figure 4.19 Update InterPoP Connections 37
Figure 4.20 Remove InterPoP Connections 38
Figure 4.21 InterPoP Connections internal structure 38
Figure 4.22 IIPC use cases workflows 39
Figure 4.23 Provision InterPoP Connections flow 40
Figure 4.24 Update InterPoP Connection 42
Figure 4.25 Remove InterPoP Connection Flow 43
Figure 4.26 Control Plane components involved by QoS 43
Figure 4.27 Set QoS Constraints on RAN per Slice 46
Figure 4.28 Set QoS Constraints on Core per IMSI 47
Figure 4.29 Set QoS constraints on CORE per Bearer ID 48
Figure 4.30 Set QoS constraints all segments 49
Figure 4.31 Set Priority per Slice/Flow 50

Figure 4.32 NF Config Workflow 52
Figure 4.33 Backhaul DPP Adapter 54
Figure 4.34: Provision SDN Intent 55
Figure 4.35 Update SDN Intent 55
Figure 4.36 Remove SDN Intent 56
Figure 4.37: Get SDN Topology 56
Figure 4.38 Backhaul Adapter interworking 57
Figure 4.39 Backhaul Adapter internal structure 57
Figure 4.40 Provision SDN Intent flow 58
Figure 4.41 Update SDN Intent flow 59
Figure 4.42 Remove SDN Intent flow 60
Figure 4.43 Get SDN Topology flow 60
Figure 5.1 Registration model A: Many services instances over one single server 64
Figure 5.2 Registration model B: Many service instances over one single server 65
Figure 5.3 Registration model C: One service instance over a single server 65
Figure 5.4 set/remove priority 81
Figure 5.5 set/remove minimum bandwidth guaranteed 81
Figure 5.6 set/remove maximum bandwidth allowed 81
Figure 6.1 Prototype reference architecture 83
Figure 6.2 CPSR Prototype reference architecture with no High Availability 84
Figure 6.3 CPSR Prototype reference architecture with High Availability 86
Figure 6.3 QoS prototype architecture 87
Figure 6.4 QoS prototype docker image 87
Figure 6.5 IPC prototype architecture 88
Figure 6.6 IPC prototype docker image 88
Figure 6.7 BKH ADTAPTER prototype architecture 90
Figure 6.8 BKH ADAPTER prototype docker image 90
Figure 6.9 Mininet and Backhaul Adapter prototype 91
Figure 6.10 FCA Controller Architecture 93

List of tables

Table 4.1 CPS type 26
Table 4.2 Mapping between IPC CPS operations and D2.3 FGE operations 36
Table 4.3 Set QoS Constraints 44
Table 5.1 CPSR Resources and methods overview 61
Table 5.2 CPS Discovery 62
Table 5.3 CPS profile 62
Table 5.4 CPS Types 63
Table 5.5 CPS Service 63
Table 5.6 CPS Service Ip End Point 64
Table 5.7 CP CONF-S supported operations 65
Table 5.8 Body request structure 66
Table 5.9 Set QoS constraints, parameters 66
Table 5.10 Set QoS constraints, JSON parameters 67
Table 5.11 Set Priority, parameters 67
Table 5.12 Provision interPoP Connections parameters 68
Table 5.13 InterPoP Path parameters 69
Table 5.14 endPoint Parameters 69
Table 5.15 constraints Parameter 69
Table 5.16 Update interPoP Connections parameters 70
Table 5.17 InterPoP Path parameters 70
Table 5.18 endPoint Parameters 70
Table 5.19 constraints Parameter 71
Table 5.20 remove InterPoP Connections Parameters 72
Table 5.21 InterPoP Path parameters 72
Table 5.22 endPoint Parameters 72
Table 5.23 intent object parameters 73
Table 5.24 IntentObj structure parameters 73
Table 5.25 endPoints structure parameters 74
Table 5.26 constraints Parameter 74
Table 5.27 intent object parameters 75
Table 5.28 IntentObj structure parameters 75
Table 5.29 endPoints structure parameters 75
Table 5.30 constraints Parameter 76
Table 5.31 remove SDN intent parameters 76

Table 5.32 IntentObj structure 76
Table 5.33 endPoint structure parameters 77
Table 5.34 get SDN topology parameters 77
Table 5.35 Intent parameters 78
Table 5.36 endPoint Parameters 78
Table 5.37 constraints Parameter 78
Table 5.38 parameters 79
Table 5.39 traffic definition parameters 80

Abbreviations

API	Application Programming Interface
BBU	Base Band Unit
CN	Core Network
CP	Control Plane
CPS	Control Plane Service
CQI	Channel Quality Indicator
CQI	Channel Quality Indicator
CSP	Communication Service Providers
DSC	Digital Service Customer
DSP	Digital Service Provider
E2E	End-to-End
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
ID	Identifier
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
KPI	Key Performance Indicator
LCM	Lifecycle Management
MANO	Management and Orchestration
ME	Mobile Edge
MEC	Mobile/Multi-access Edge Computing
MEO	Mobile Edge Orchestrator
NE	Network Element
NEF	Network Exposure Function
NF	Network Function
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NS	Network Service; Network Slice
NSaaS	Network Slice as a Service
NSD	Network Service Descriptor
NSEP	Network Slice Provider
NSI	Network Slice Instance
NSP	Network Service Provider

NST	Network Slice Template
OAM	Operations, administration and maintenance
OSS	Operations Support System
P&P	Plug & Play
PNF	Physical network Function
PoP	Point-of-Presence
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RO	Resource Orchestrator
RRH	Remote Radio Head
SBA	Service Based Architecture
SBI	Service Based Interface
SDN	Software Defined Networks
SDO	Standards Developing Organization
SLA	Service Level Agreement
SliceNet	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualized Multi-Domain, Multi-Tenant 5G Networks
SW	Software
SS-O	Slice Service Orchestrator
UC	Use Case
UE	User Equipment
URI	Uniform Resource Identifiers
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	VNF Descriptors
VNFFG	VNF Forwarding Graphs
WG	Working Group

1 Introduction

1.1 Scope

This deliverable aims to design and prototype the "Intra Domain Slicing" functionality according to the SliceNet Control Plane architecture as defined in Deliverable D2.3 [2]

With respect to the SliceNet logical architecture [1] and current evolution [45] as depicted in Figure 1.1 the Intra Domain Slicing framework is positioned within the Control Plane layer [2] as responsible for enforcement (per-slice runtime) of network functions configuration rules and policies governing the run time operations of RAN, Core, MEC and Backhaul network segments within the same administrative domain.

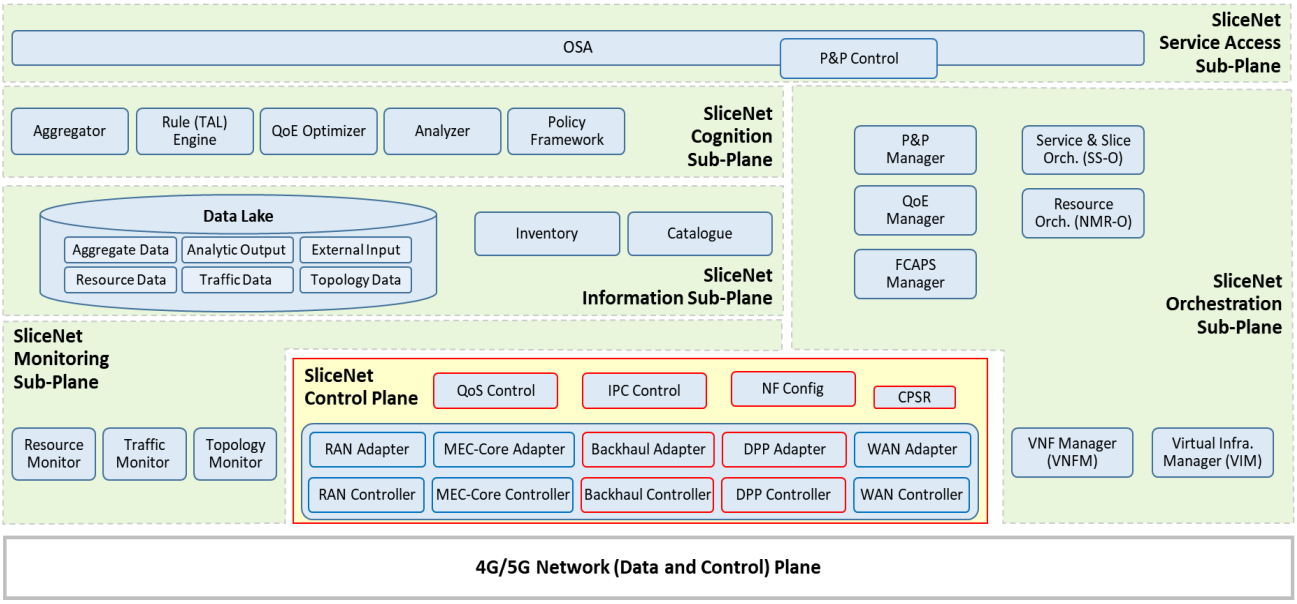

Figure 1.1 Intra-Domain Slicing within the SliceNet logical architecture

In details, the document deals with the implementation of Slice control functionalities by a set of SW components called Control Plane Services (CPS) each offering specific configuration and control capabilities as detailed within next chapters. The internal architecture following the Service Based Architecture (SBA) principles is as well addressed describing how the SW components offer or consume services communicating by an agreed Service Based Interface (SBI) over a common bus.

The following Control Plane Services and Adapters are covered by this deliverable:

- Control Plane Service Register (CPSR);
- Inter-PoP Connections Control (IPC);
- Quality of Service (QoS) Control;
- Network Functions (NF) Configuration Control
- Backhaul Adapter
- Data Plane Programmability (DPP) Backhaul Adapter

1.2 Document structure

This document is structured as follows:

- Section 2 provides an overview on current architectures and technologies for controlling the different segments of 5G networks (e.g. RAN, MEC, Core, Backhaul) and highlights how current state of the art is advanced by this delivery.
- Section 3 defines the main principles that have driven the design and implementation of the SliceNet Control Plane Architecture and of the related components in the scope of this deliverable. Focus is on Service Based Architecture and technology agnostic API abstraction.
- Section 4 is the core part of the document where all the delivered components are described in terms of functionalities and offered operations through their exposed Service Based Interface; example of possible workflows involving the concerned components are also reported.
- Section 5 provides a complete overview of all the Interfaces with reference to the SliceNet Control Plane Architecture. The interfaces directly related to the components in the scope of this deliverable are fully described in terms of possible operations and parameters while the others are reported for completeness including the reference where the detailed description is.
- Section 6 provides a description of the components software prototype that has been implemented following the architecture, work flows and APIs definitions of Section 4.
- Section 7 provides some concluding remarks and highlights the future work.

2 State of the art

This section provides an overview of the technologies, paradigms and approaches that are being currently used to implement the control of communication networks, mainly focusing in 5G/4G and their composing network segments. Since the main goal of the SliceNet Control Plane (CP) is to seamlessly interconnect different network segments to provide end-to-end connectivity through the creation of slices, the CP has to be able to deal with the technologies associated to the control of such network segments. In addition, the advances on the control of end-to-end 5G/4G networks offered by the SliceNet CP are highlighted.

2.1 Service-based approach

Looking at the literature, several approaches can be found to implement efficient 5G/4G control planes. Such approaches span from the clean-slate proposals like [8] to more standard-aligned initiatives that define modifications to the architecture proposed by standardization bodies such as the 3rd Generation Partnership Project (3GPP). For example, the authors of [9] propose a centralized control by moving the control functionality of the 5G RAN to the Core in order to reduce the signalling between these two segments. Taking a different approach, the authors of [10] propose a parallelization of several procedures in the LTE networks control aiming to reduce the latency in different situations: latency in establishing a new data service, latency in retaining data service in a handover and latency in wide-area roaming.

In contrast, the SliceNet control framework aims to extend the functionalities of the 5G/4G CP without compromising the modularity and standards compliance of the existing control plane. To do this, the SliceNet Control Plane [2] defines a set of functional modules that interact with the heterogeneous 5G/4G infrastructure to provide end-to-end slices in support of vertical-oriented services. Such modules are deployed in three layers in the SliceNet CP. The first layer contains one of the most innovative components proposed by SliceNet, namely, the Plug & Play (P&P) [4] , which is aimed to provide per slice runtime customization. The second layer is composed of the control plane services (e.g., CPSR, QoS Control, IPC and NF Configuration Control), which are responsible for implementing the configuration and control functions requested from the other planes of the SliceNet system (i.e., the Orchestration, the Management and the Cognition sub-planes). Finally, the lower layer of the SliceNet CP contains the Adapters, which are a set of modules that implement the functions to be invoked by the control plane services to configure each segment of the network in order to provide the end-to-end slices. These Adapters have been designed to be technology-agnostic in the northbound and technology-specific in the southbound.

The enabler of the communication between the different layers of the proposed architecture is the service bus. During operation, the modules of the SliceNet architecture (i.e., the ones at the Management, the Orchestration and the Cognition sub-planes) that need to contact the control plane to carry out their responsibilities connect to the SliceNet CP CPSR, which contains a reference to the control plane functions available, to obtain the pointers to the needed control planes functions. These, in turn, contact the CPSR to obtain access to the proper adapters, which contacts the appropriated segment and technology control at the infrastructure level.

The rationale behind the service-based approach is to provide a technology agnostic control plane able to configure any kind of underlying technology, as well as achieving a highly flexible architecture where services can be dynamically added at any layer of the control plane. The service-based architecture designed for the SliceNet CP is illustrated in Figure 1.1 and further described in section 3.2.

2.2 Multi-segment technology-agnostic control

As previously stated, the goal of the SliceNet CP is to obtain an intra-domain multi-tenant slicing functionality, offering a set of overlay services on top of 5G/4G standard control and data planes, which advances the state of the art in enabling an adaptive, flexible, standard-compliant and interoperable network slicing architecture. To achieve this, the SliceNet CP takes advantage of the existing control technologies that can be found for the different segments of the 5G/4G control and data plane infrastructure: Radio Access Network (RAN), Mobile Edge Computing (MEC), Backhaul, Core and Wide Area Network (WAN).

2.2.1 Backhaul and WAN

Different state-of-the art paradigms and technologies are assumed to be controlled from the SliceNet CP. Currently, one of the most accepted models for network control is the Software Defined Networks (SDN) [11] . By definition, SDN is aimed to dynamically configure the connectivity requests coming from the application layer. Hence, the SDN controller, which has a global vision of the network, centralizes the logic to program the data plane forwarding behaviour according to the application needs. To do this, the controller manages policies and rules to configure data flows over the network infrastructure, which can be either physical or virtual. In this regard, along with the network programmability, virtualization is one of the added values that the SDN paradigm offered to the traditional network control, thus paving the way to the current network slicing.

There is a number of well-known open source SDN controllers (e.g. Floodlight [12] , OpenDaylight [13] , ONOS [14] and Ryu [15]). Most of them support OpenFlow [16] as the protocol to configure the data plane. Nonetheless, other protocols are extensively used to implement the southbound interface. For example, the IETF defined the NetConf [17] protocol to install, manipulate and delete the configuration of network devices. On the contrary, there is no standard defined to implement the northbound of the SDN controller, which is typically based on proprietary REST-based interfaces. SDN is assumed to be one of the technologies used in the Backhaul and in the WAN segments.

Also in the Backhaul segment, the SliceNet CP introduces the Backhaul DPP Adapter (described in section 4.2.1) that is able to provide highly flexible control and configuration of the wired network data flows, thus enabling enhanced slicing capabilities to this segment of the network. More specifically, the Backhaul DPP Adapter relies on the Flow Control Agent (FCA) Controller. The FCA Controller offers fully-configurable fine-grained data flow grouping in support of nested encapsulation, which enables mobility support and multi-tenancy, thus making the 5G technology compliant. In light of this, the Backhaul DPP Adapter and its underlying technology overcome the slicing limitations of the current standard SDN controllers.

2.2.2 RAN

Several 5G RAN design requirements and paradigms to enable RAN slicing are elaborated in [18] . 3GPP mentions the RAN slicing realization principles in [19] and [20] including RAN awareness slicing, QoS support, resource isolation, SLA enforcement among others. A fully centralized architecture of CP functionalities is proposed such as OpenRAN in [21] and as SoftAir in [22] that may face the challenge of real-time control given the inherent delay between the controller and underlying RAN. The SoftRAN [23] architecture statically refactors the control functions into the centralized and distributed ones based on the time criticality and the central view requirement. The SoftMobile approach [24] further abstracts the CP processing in several layers based on the functionalities in order to perform the control functionalities through the application programming interfaces (APIs). As for the UP programmability and modularity, the OpenRadio [25] and PRAN [26] are pioneered to decompose the overall processing into several functionalities that can be chained. FlexRAN [27] realizes a SD-RAN platform and implements a custom RAN south-bound API through which programmable Control Logic can be enforced with different levels of centralization, either by

the controller or the local RAN agents. To make a unified and flexible execution environment to run multiple virtualized RAN instances with the required levels of customization over the monolithic or disaggregated RAN, the RAN runtime slicing system [28] is proposed in Deliverable D4.2 [5] .

2.2.3 Core Network

In order to create an end-to-end Network Slice, the RAN sharing should be combined with Core Network (CN) slicing approaches. Many architectures and prototypes have been proposed for CN slicing [29] [30] , [31] . The challenge of CN slicing has been also addressed by 3GPP, realized through a dedicated Core network (DECOR) [32] and evolved DECOR (eDECOR) [33] . In this regard, different approaches for supporting network slicing in 5G system and a prototype implementation of network slicing in 4G LTE CN are presented in Deliverable D4.2 [5] , which provide the Adapter to be used by the SliceNet CP.

3 Principles and architecture

This section provides relevant information related to the architecture and principles forming the foundations for the introductions of the Control Plane Services.

3.1 Control Plane Services Principles and Architecture

The SliceNet Control Plane is designed around two main principles:

- a Service Based Architecture (SBA) approach;
- a technology agnostic APIs abstraction.

Figure 3.1 shows how the functionality is exploited by the introduction of a number of CP Services (blue boxes) evolving the general Control Plane architecture [2] in terms of both CP Services and Adapters; namely the Data Plane Programmability (DPP) is removed and the new Backhaul DPP Adapter is introduced in order to optimize the related functionality as further described in Section 4.

In addition, the CP Services are instantiated per slice (described in section 3.2) and require a Life Cycle Manager for handling instances, as described in section 3.4.

The interfaces exposed by the CP components are named as shown in Figure 3.1, and described in detail in the current deliverable document under Section 5.

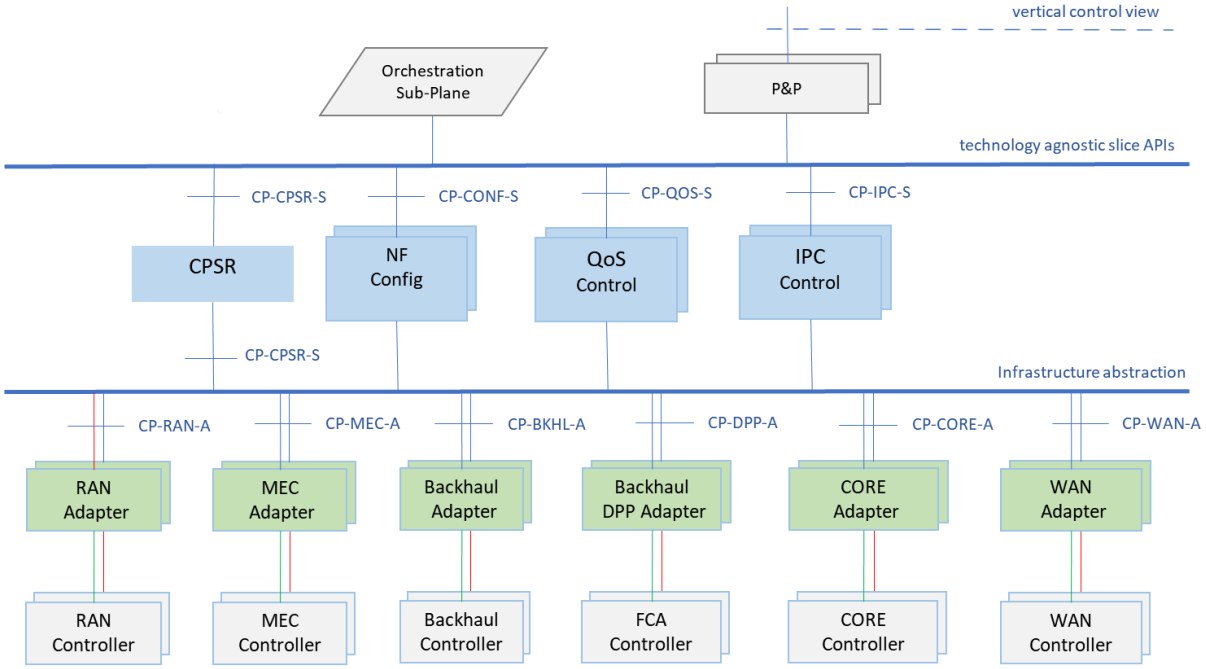

Figure 3.1 SliceNet Control Plane Services architecture

3.2 Service Based Architecture

The SliceNet CP is based on the realisation of a Service Based Architecture (SBA) that is quite aligned with the concepts being exercised by Next Generation Mobile Networks (NGMN) and 3rd Generation Partnership Project (3GPP).

According to the SBA principles, the system functionality in object is achieved by a set of services, here identified as CPSs, loosely-coupled with each other allowing individual services to be developed, deployed and upgraded with minimal impact to other services.

Each Service can interact directly with other services with light-weighted Service Based Interface (SBI) and can be reused by other services.

A SBI represents how the set of services is provided or exposed by a given Control Plane Service. This is the interface where the control plane service operations are invoked.

Protocols used for the SBI are: HTTP/2 for application layer, TCP for transport, JSON for serialization, RESTful framework for the API style and OpenAPI for Interface Description Language (IDL).

Each service instance is deployed in a SBA framework allowing service registration, authorization and discovery; the SBA framework is implemented by the Control Plane Service Register (CPSR) which is mainly a database of available services instances and their reachability. A Service registers itself to the CPSR when its instance is created; Service consumers (SliceNet components) can query the CPSR to find available services and their addresses.

SBA is considered an effective enabler for the system scalability as new instances of the same CP Service or even of new CP Services can easily be added.

There is one instance of the CPSR while in general all the other CP Services have multiple instances.

One instance of each CP Service is created to be serving one Slice instance; this one-to-one relationship is regarded as a key enabler for Slice performance and security isolation as well as for the overall system scalability when it comes to the number of Slices instances.

The security access authorization to servers are based on OAuth2.0 procedures.

3.3 Technology agnostic abstraction

The main purpose of the CP Services is to provide the Slice control context by a set of SliceNet configuration endpoints exposing technology and implementation-agnostic control APIs towards slice management and orchestration components following the SBA approach.

The CP Services lay on top of the intra-domain SliceNet physical and virtualised infrastructure, which spans across multiple network technology domains covering the RAN, MEC, Backhaul and Core segments of 4G/5G networks belonging to the same administrative entity.

As shown in Figure 3.1 the set of Adapters provide a first level of abstraction over the network pillar functionalities which is further exploited and abstracted by CP Services which expose specific services by their own Service Based Interface (SBI).

The Controllers support the interaction with specific control technologies of the SliceNet infrastructure segments possibly provided by different vendors.

The Adapters translate the Controllers Northbound interface into a technology agnostic Interface, thus enabling a common SliceNet CP information model and control logics.

In summary, with reference to Figure 3.1, the Controllers layer expose a technology dependant Northbound Interface, the Adapters layer expose a technology agnostic Northbound Interface and the CP Services expose a further abstracted technology agnostic interface which offer a Slice control context hiding the Slice detailed composition in terms of network segments and vendors technology.

3.4 Control Plane Services Life Cycle Management

As described in Section 3.2, CPSs represent different per slice instances of CP oriented service, based on the slice requirements, deployed on demand and destroyed when they no longer need.

In this sense, the CPS instances are dynamic architectural components, characterized by their own life cycle (as opposed to the other SliceNet architecture facilities) that should be managed at runtime by a proper **life-cycle manager**.

Based on the properties that characterize the CPS instances, the idea is to properly extend the P&P Manager (described architecturally in D2.4 [3] and under prototyping in Task T6.3) in order to make it able to control also the life-cycle of the CPS instances, by interacting with their own orchestrator.

P&P Manager is a component that resides on the SliceNet orchestration plane designed for handling the lifecycle of the P&P Control instances (described in D4.1 [4]). The lifecycle management of the CPS instances through the P&P Manager is doable since the two kind of instances (P&P Control and CPS) present strong similarities:

- **They are on-demand components**: the life-cycle of each CPS instance is strictly related to the life-cycle of the slice it belongs to, so that, each changes in the slice requirements, in terms of control features, affect the corresponding instances of CPS and,if the slice is destroyed also the CPS instances will be destroyed.
- **They are per-slice components**: a CPS instance offers its own features to the service consumers in the context of a given slice. In addition, the P&P Control instances of that slice is one of the service consumers for the CPS instance.
- **They are implemented as software containers** under the control of a proper orchestrator (e.g. Kubernetes).

With this in mind, the CPS instances, from the life-cycle point of view, present the same management problematics as the P&P Control instances that can be addressed by the P&P Manager. All of the life-cycle procedures performed by P&P Manager against the P&P control instances are detailed in D4.1 [4] and are extended in D6.3 [7] in order to meet the management requirements imposed by the CPSs.

In particular, the life-cycle management actions that the P&P Manager can perform on the CPS instances are **creation**, **configuration** and **termination**, as depicted in Figure 3.2.

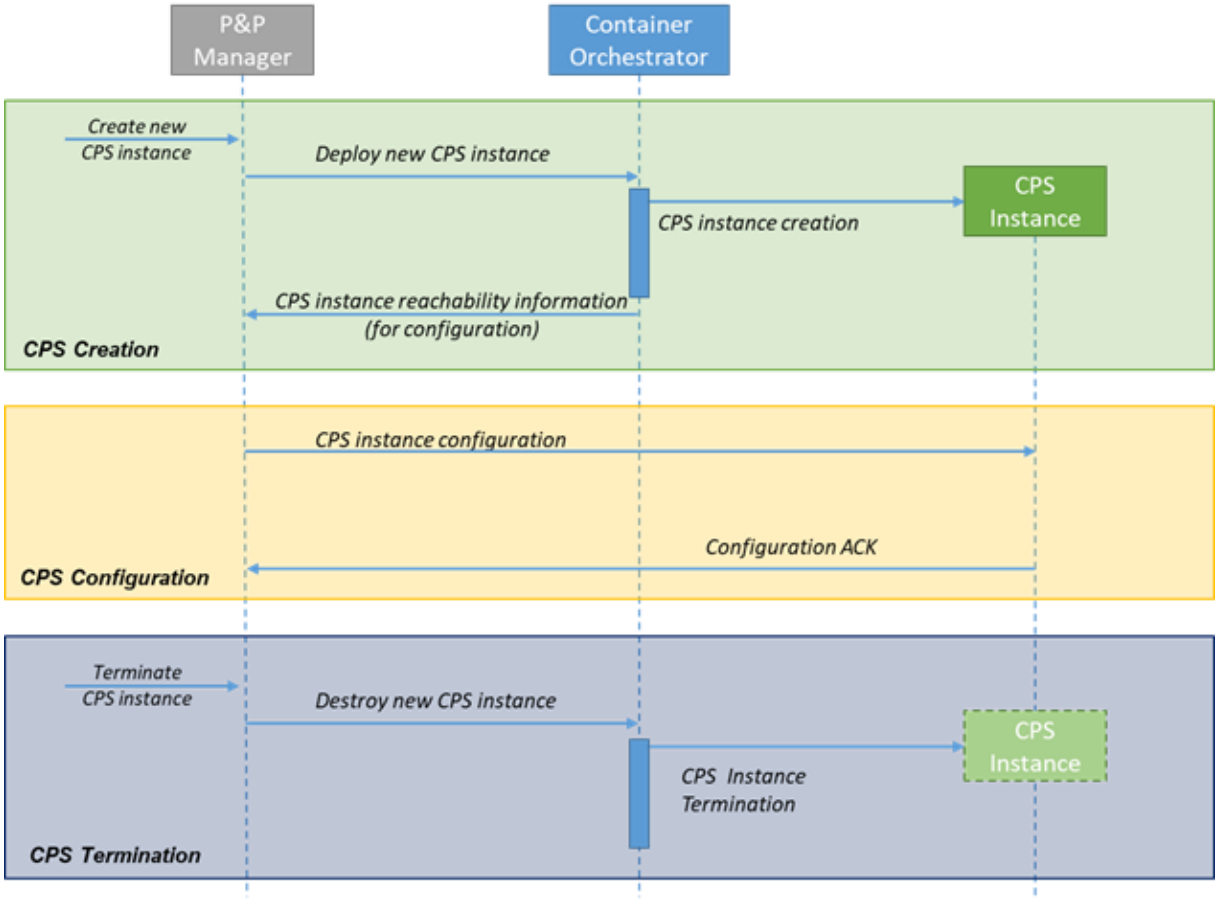

Figure 3.2 CPS Life-cycle management steps

3.4.1 CPS Creation

The P&P Manager triggers the CPS creation process by invoking the proper API on the interface provided by the CPS orchestrator. It is worth noting that a call to create a new CPS instance can happen for different reasons during the slice life-time.

New CPS instances are needed once a new slice is deployed. In this case, the creation of each CPS should happen at the same time of the creation of the slice it belongs to. For that reason, the P&P Manager should launch the creation process as soon as the Slice Service Orchestrator (SS-O) invokes the deployment of the slice (or immediately after the slice creation). To guarantee a prompt CPS creation, the P&P Manager should be triggered by the SS-O once it starts a new slice process creation.

New slice requirements, in terms of control feature, is also a valid reason to trigger the creation of new CPS instances as well as the replication of the same CPS can cope the service scalability demand if needed.

Starting from the SS-O request, the CPS creation consists of the following steps:

- The P&P Manager receives a request to create a new CPS instance. As described above, it can happen due a new slice creation or slice requirements change or even to meet service scalability requirements;
- The P&P Manager invokes the deployment of the new instance through the interface of the orchestrator;

- The orchestrator starts creating the new instance. Once the process terminates, the P&P Manager receives information about the CPS instance reachability and the creation process is completed.

3.4.2 CPS Configuration

Once the creation process terminated, the P&P Manager should configure the new CPS instance by providing relevant information about the slice and the components involved. Such information will include definitely the reachability of the CPSR, in order to make the CPS instance able to register itself as described in section 4.1.1.1.1. The configuration steps are as follows:

- The P&P Manager sends the configuration information to the new CPS instances;
- The CPS instance sends an ACK to confirm the configuration reception. Such ACK could be a simple 200 OK HTTP Code.

3.4.3 CPS Termination

The termination process should happen once the slice is going to be destroyed or, in general, once the CPS instance no longer meets the slice requirements. After the P&P Manager receives the termination request, the process consists of the following two steps:

- The P&P Manager invokes the destruction of the CPS instance through the interface of the orchestrator;
- The CPS orchestrator proceeds with the termination of the correspondent CPS instance.

3.5 Registration and retention flow

According to the Service Based Architecture principles, all the Control Plane Services and Adapters shall register to the Control Plane Service Register (CPSR) according to the flow depicted in Figure 3.3.

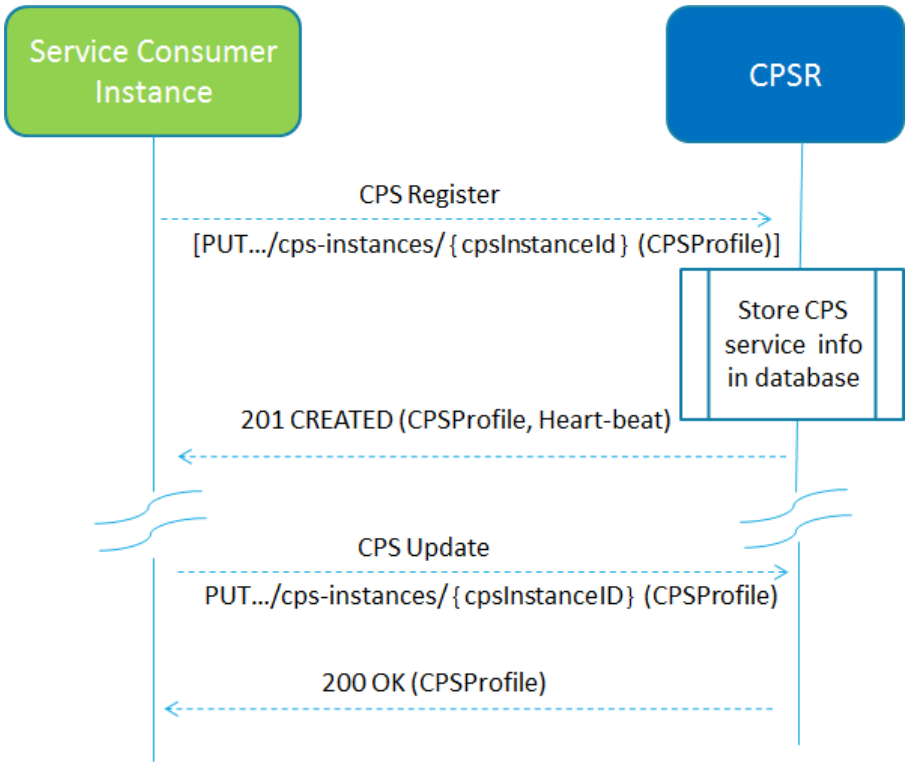

Figure 3.3 Registration and Retention flow

- Any CPS can register to the CPSR via CPSR Register function (see section 4.1.1.1.1) by providing the CPS profile to the CPSR. CPSR stores the received info and marks the requesting CPS as available to be discovered by other CPSs.
- Each CPS service successfully registered in CPSR shall contact the CPSR periodically (heart-beat), by invoking the CPS Update (see section 4.1.1.1.2) service operation, in order to show that the CPS is still operative. The time interval at which the CPSR shall be contacted is returned by the CPSR to the Service Consumer as a result of a successful registration (it is recommended to contact the CPSR before the heart-beat timeout)
- When the CPSR detects that a given CPS has not updated its profile for a specific amount of time (longer than the heart-beat interval), the CPSR considers the CPS as de-registered (see section 4.1.1.1.4) and removes it from its database.

4 Description of Single-Domain CP components

This section describes the Single-Domain CP components detailing the internal decomposition with workflow procedures and related APIs and interfaces.

4.1 Control Plane Services

4.1.1 Control Plane Service Register

The Control Plane Service Register (CPSR) is the main SW component used to implement the SBA principles in SliceNet Control Plane layer. It offers the SBI for registration and discovery management used for all the CP services. The producers register their services, and the consumers can ask where services are located.

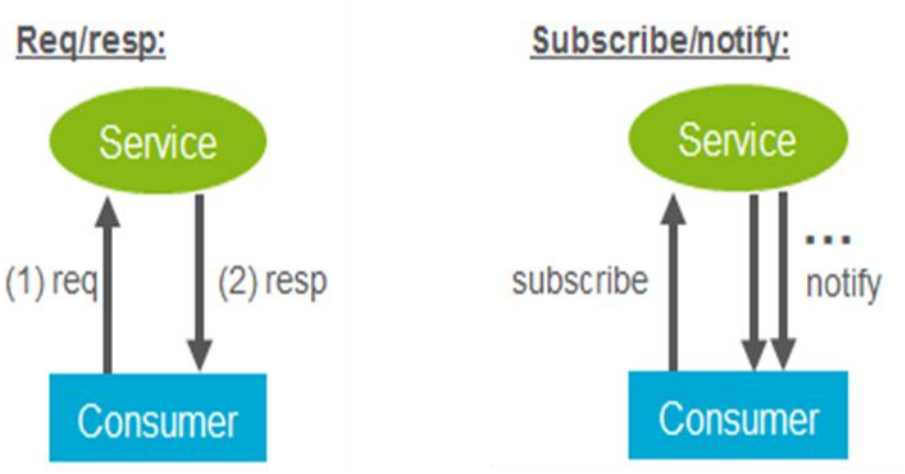

Figure 4.1 request/response and subscribe/notify

The CPSR provides the following main functionalities:

- CPS element registration and de-registration: to make the service consumer aware of the available Service producer instances and supported services;
- CPS element discovery: to enable a Service Consumer to discover Service Producer instance(s) which provide the expected service(s);
- CPS authorization: to ensure the Service Consumer is authorized to access the service provided by a specific Service Producer.

The service register is the key part of the service framework.

In addition to the above core features, the CPSR also supports the following advanced features:

- Maintain the CPS profile of available service producer instances and their supported services;
- Allows other CPS instances to subscribe to, and get notified about, the registration in CPSR of new CPS instances of a given type;

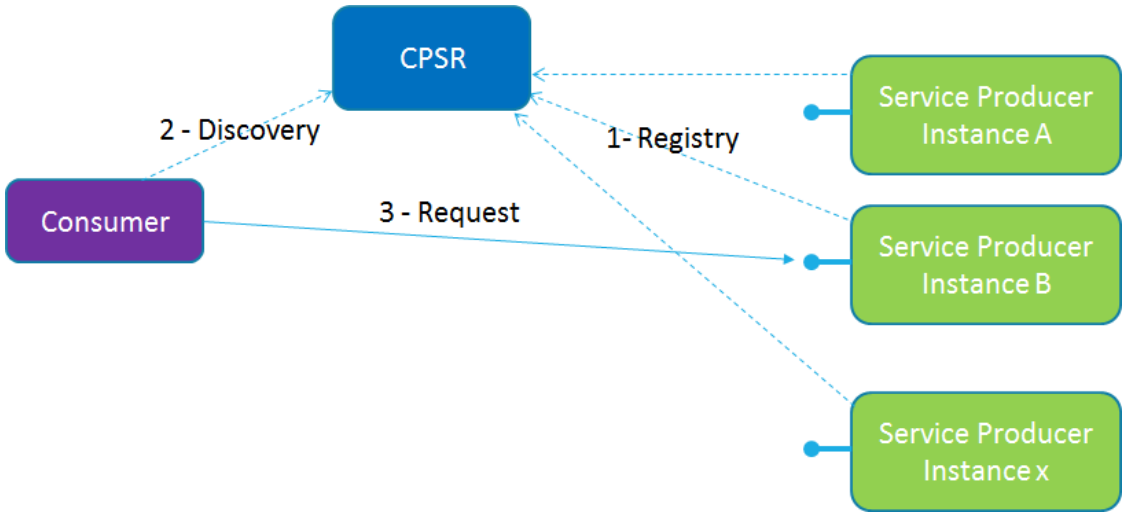

Figure 4.2 registry and discovery

Examples:

Figure 4.2 shows how the CPSR could be used by producers/consumers. Below we report some examples in "curl" commands how that could be done.

Registry:

```
curl -X PUT -H "Content-Type: application/json" -d '{<CPFProfile's json
data structure>}'http://localhost:8080/slicenet/ctrlplane/cpsr_cps/v1/cps-
instances/550e8400-e29b-41d4-a716-446655440000
```

Discovery:

```
curl -v -X GET -H "accept: application/json" -H "Content-Type:
application/json" http://localhost:8080/slicenet/ctrlplane/cpsr_cps/v1/cps-
instances?cpsType=COR_ADAPTER&limit=10
Answer: List of URI matching the request in json format
Example: [{"uri":"http://192.99.55.1:8081/coretim/cps-
xyz/v2"},{"uri":"http://192.99.33.1:8081/cpsr/cps-abc/v1"}]
```

The following CPS type, detailed in Table 4.1, are used during the registration of a service producer.

Table 4.1 CPS type

CPSType	
Enumeration value	**Description**
BKH_ADAPTER	Backhaul Adapter
COR_ADAPTER	Core Adapter
MEC_ADAPTER	Mobile Multi Access Edge Computing Adapter
RAN_ADAPTER	Radio Access Network Adapter
WAN_ADAPTER	Wide Area Network Adapter
DPP_ADAPTER	Data Plane Programmability Adapter
QOS_CP	Control Plane Service: Quality of Service
IPC_CP	Control Plane Service: InterPoP Connection
QOE_CP	Control Plane Service: Quality of Experience
PP_CP	Control Plane Service: Plug and Play
CNF_CP	Control Plane Service: Network Function Configuration
ID_CP	Control Plane Service: Interdomain

4.1.1.1 CPSR: Management Service

The CPRS allows a service producer instance in the CP architecture to register, update or deregister its profile.

It also allows a service provider to subscribe to be notified of newly registered CPS Instances along with their CPS services.

The service operations defined for the CPSR Management service are as follows:

- **CPS Register**: It allows an CPS Instance to register its CPS profile in the CPSR; it includes the registration of the general parameters of the CPS Instance, together with the list of services exposed by the CPS Instance.
- **CPS Update**: It allows an CPS Instance to replace, or update partially, the parameters of its CPS profile (including the parameters of the associated services) in the CPSR; it also allows to add or delete individual services offered by the CPS Instance.
- **CPS Deregister**: It allows an CPS Instance to deregister its CPS profile in the CPSR, including the services offered by the CPS Instance.
- **CPS StatusSubscribe**: It allows a CPS Instance to subscribe to change on the status of CPS Instances registered in CPSR.
- **CPS StatusNotify**: It allows the CPSR to notify subscribed CPS Instances of changes on the status of CPS Instances.
- **CPS StatusUnsubscribe**: It allows an CPS Instance to unsubscribe to changes on the status of CPS Instances registered in CPSR.

NOTE: The "change of status" of the CPS Status service operations can imply a request to be notified of newly registered CPS Instances in CPSR, or to be notified of profile changes of a specific CPS Instance, or to be notified of the deregistration of an CPS Instance.

4.1.1.1.1 CPS Registration operation

This service operation registers service provider (CPS entity) in the CPSR by providing the CPS profile of the requested CPS to the CPSR, and CPSR marks the requested CPS as available to be discovered by other CPSs. It is also used to register services associated to an existing CPS Instance.

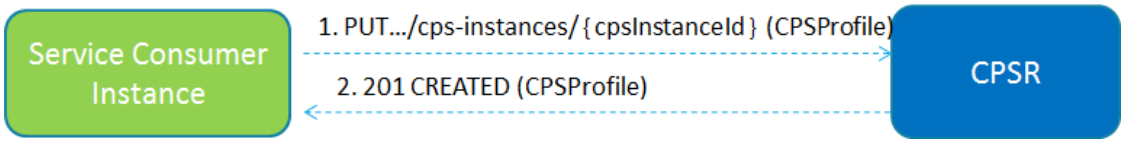

Figure 4.3 registration operation

With reference to Figure 4.3 the steps of register operation are described as follows:

1. The CPS Service should send a PUT request to CPSR with the details of its URI. The payload body of the PUT request contains a representation of the CPS Instance to be created.
2. On success, "201 Created" should be returned, the payload body of the PUT response contains the representation of the created resource and the "Location" header contains the URI of the created resource. Additionally, the CPSR returns a "heart-beat timer" containing the number of seconds expected between two consecutive heart-beat messages from a CPS instance to the CPSR.

If the registration of the CPS instance fails at the CPSR due to errors in the encoding of the CPS Profile JSON object, the CPSR should return "400 Bad Request" status code with the ProblemDetails IE providing details of the error.

If the registration of the CPS instance fails at the CPSR due to CPS internal errors, the CPSR returns "500 Internal Server Error" status code with the `ProblemDetails` providing details of the error.

4.1.1.1.2 CPS Update

This service operation updates the profile of a Service Consumer, registered in the CPSR, by providing the updated CPSprofile of the requesting CPS to the CPSR. The update operation may apply to the whole profile of the CPS (complete replacement of the existing profile by a new profile), or it may apply only to a subset of the parameters of the profile (including adding/deleting/replacing services to the CPSprofile).

To perform a complete replacement of the CPSProfile of a given CPSInstance, the Service Consumer should issue an HTTP PUT request.

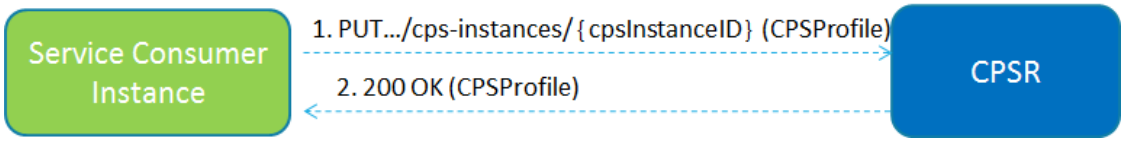

Figure 4.4 update operation

To perform a partial update of the CPS Profile of a given Service Consumer Instance, the Service Consumer shall issue an HTTP PATCH request. This partial update shall be used to add/delete/replace individual parameters of the CPS Instance, and also to add/delete/replace any of the services (and their parameters) offered by the CPS Instance.

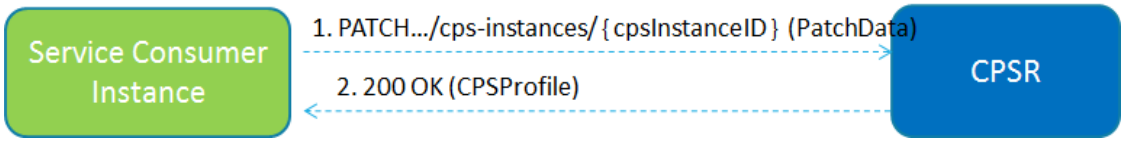

Figure 4.5 partial update operation

As shown in Figure 4.5 the steps of partial update operation are described as follows:

1. The Service Consumer sends a PATCH request to the resource URI representing the CPS Instance. The payload body of the PATCH request contains the list of operations (add/delete/replace) to be applied to the CPS Profile of the CPS Instance; these operations may be directed to individual parameters of the CPS Profile or to the list of services (and their parameters) offered by the Service Instances. In order to leave the CPS Profile in a consistent state, all the operations specified by the PATCH request body shall be executed atomically.
2. On success, "200 OK" should be returned, the payload body of the PATCH response contains the representation of the replaced resource.

4.1.1.1.3 CPS Heart-Beat

Each Service Consumer that has previously registered in CPSR should contact the CPSR periodically (heart-beat), by invoking the CPSUpdate service operation, in order to show that the CPS is still operative.

The time interval at which the CPSR shall be contacted is deployment-specific, and it is returned by the CPSR to the Service Consumer as a result of a successful registration.

When the CPSR detects that a given CPS has not updated its profile for a configurable amount of time (longer than the heart-beat interval), the CPSR considers the CPS as deregistered and its services can no longer be discovered by other CPSs via the CPS Discovery service.

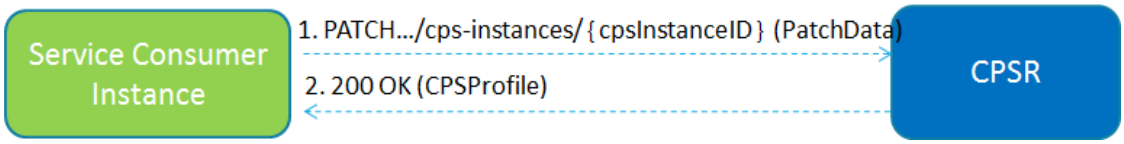

Figure 4.6 Heart-Beat

With reference to Figure 4.6 the steps of Heart beat operation are here described:

1. The Service Consumer sends a PATCH request to the resource URI representing the CPS Instance. The payload body of the PATCH request contains a "replace" operation on the "Status" attribute of the CPS Profile of the CPS Instance and set it to the value "REGISTERED".
2. On success, "200 OK" shall be returned, the payload body of the PATCH response contains the representation of the replaced resource.

4.1.1.1.4 CPS Deregistration operation

This service operation removes the profile of a CPS instance previously registered in the CPSR.

It is executed by deleting a given resource identified by a "CPS Instance ID". The operation is invoked by issuing a DELETE request on the URI representing the specific CPS Instance.

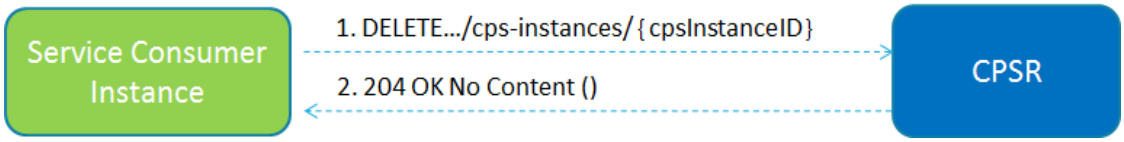

Figure 4.7 Deregistration operation

With reference to Figure 4.7 the step of deregister operation are here described:

1. The Service Consumer sends a DELETE request to the resource URI representing the CPS Instance. The request body would be empty.
2. On success, "204 No Content" is returned. The response body would be empty.

4.1.1.1.5 CPS Status Subscribe

This service operation is used to:

a) Subscription to newly registered CPS Instances

The subscription to notifications on newly registered CPS Instances is executed creating a new individual resource under the collection resource "subscriptions" (Figure 4.8). The operation is invoked by issuing a POST request on the URI representing the "subscriptions" resource.

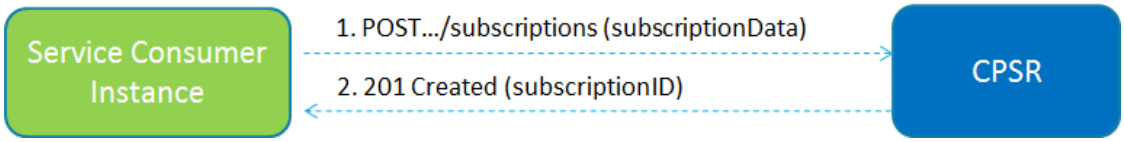

Figure 4.8 Subscription to new registered CPS instances

1. The Service Consumer shall send a POST request to the resource URI representing the "subscriptions" collection resource. The request body shall include the data indicating the type of notifications that the Service Consumer is interested in receiving; it also contains a callback URI, where the Service Consumer shall be prepared to receive the actual notification from the CPSR.
2. On success, "201 Created" is returned.

b) Subscription to changes of CPS profile

The subscription to notifications on changes of the profile, or deregistration, of a given CPS Instance is executed creating a new individual resource under the collection resource "cps-subscriptions (Figure 4.9). The operation is invoked by issuing a POST request on the URI representing the "cps-subscriptions" resource.

1. The Service Consumer sends a POST request to the resource URI representing the "cps-subscriptions" collection resource. The request body includes the data indicating which parts of the profile of the CPS Instance should trigger a notification, and/or the indication to be notified when the CPS Instance is deregistered; it also contains a callback URI, where the Service Consumer is prepared to receive the actual notification from the CPSR.
2. On success, "201 Created" is returned.

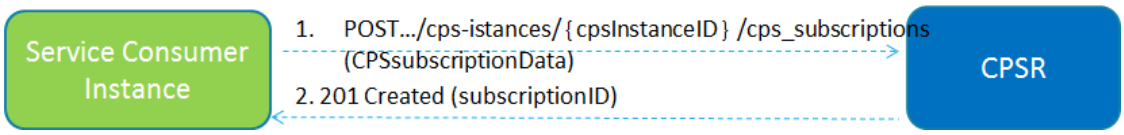

Figure 4.9 Subscription to change of CPS profile

4.1.1.1.6 CPS Status Notify

This service operation notifies each Service Consumer that was previously subscribed to receive notifications of newly registered CPS Instances, or notifications of changes of the CPS profile of a given CPS Instance, or notifications of deregistration of an CPS Instance from CPSR. The notification is sent to a callback URI that each Service Consumer provided during the subscription.

The operation is invoked by issuing a POST request to each callback URI of the different subscribed CPS Instance.

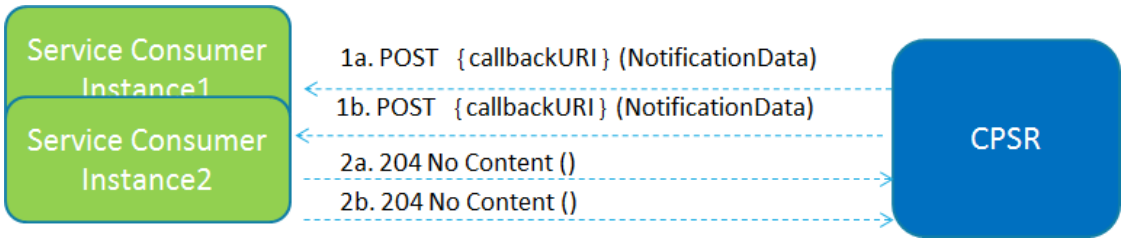

Figure 4.10 Status Notify

With reference to Figure 4.10 the step of status notify operation are here described:

1. The CPSR sends a POST request to the callback URI

   For notifications of newly registered CPS Instances, the request body includes the data associated to the newly registered CPS, and its services, according to the criteria indicated by the Service Consumer during the subscription operation. These data contain, among others, the CPSInstanceID of the CPS Instance, an indication of the event being notified ("registration"), and the services offered by the CPS Instance.

   For notifications of changes of the profile of a CPS Instance, the request body includes the CPSInstanceID of the CPS Instance whose profile was changed, an indication of the event being notified ("profile change"), and the new profile data.

   For notifications of deregistration of the CPS Instance from CPSR, the request body includes the CPSInstanceID of the deregistered CPS Instance, and an indication of the event being notified ("deregistration").

2. On success, "204 No content" is returned by the Service Consumer.

4.1.1.1.7 CPS Status UnSubscribe

This service operation removes an existing subscription to notifications.

It is executed by deleting a given resource identified either by a "subscriptionID" (for subscriptions to newly registered CPS Instances), or a "cpsSubscriptionID" (for subscriptions to changes of the profile, or deregistration, of a given CPS Instance). The operation is invoked by issuing a DELETE request on the URI representing the specific subscription.

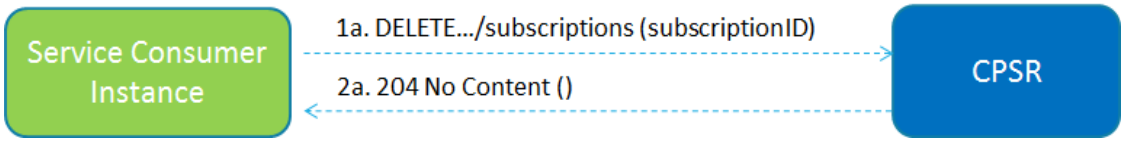

Figure 4.11 Status UnSubscribe (subscriptionID)

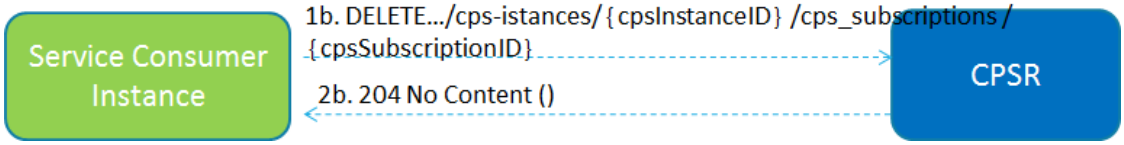

Figure 4.12 Status UnSubscribe (cpsSubscriptionID)

With reference to **Figure 4.11** and **Figure 4.12** the steps of unsubscribe operation are described as follows:

1. The Service Consumer sends a DELETE request to the resource URI representing the individual subscription. The request body would be empty.
2. On success, "204 No Content" is returned. The response body would be empty.

4.1.1.2 CPSR: Discovery (CPS Discovery)

The CPSR Discovery service allows a CPS Instance to discover services offered by other CPS Instances, by querying the CPSR.

The service operation defined for the CPSR Discovery service is:

CPS Discover: It provides to the CPS service consumer the IP address(es) or FQDN of the CPS Instance(s) or CPS Service(s) matching certain input criteria.

4.1.1.2.1 CPS Discover

The Service Consumer shall send an HTTP GET request to the resource URI "cps-instances" collection resource (Figure 4.13). The input filter criteria for the discovery request shall be included in query parameters.

On success, "200 OK" shall be returned. The response body shall contain a validity period, during which the search result can be cached by the Service Consumer, and an array of CPS profile objects, that satisfy the search filter criteria (e.g., all CPS Instances offering a certain CPS Service name). If the Service Consumer is not allowed to discover the CPS services for the requested CPS type provided in the query parameters, the CPSR shall return "403 Forbidden" response.

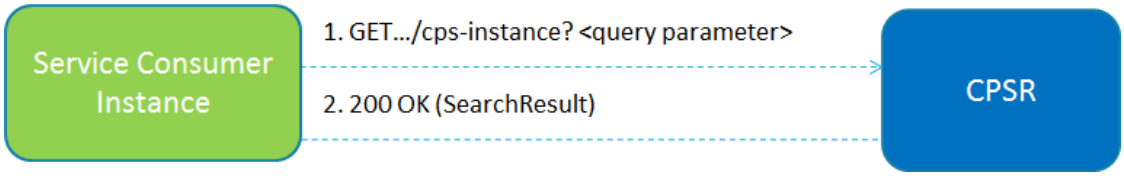

Figure 4.13 CPS Discovery

4.1.1.3 Security

The authorization access to CPS is based on OAuth 2.0 frameworks [34] . CPSR will be also the OAuth 2.0 Authorization server in the SliceNet Control plane.

The grant type used will be “Client Credential Grant” [34] Access token is based on JSON web Token [35] , secured with digital signatures based on JSON Web Signature [36] .

In terms of roles: CPSR is the OAuth 2.0 authorization server; CPS consumer is the OAuth 2.0 client; the CPS service producer is the OAuth 2.0 resource server.

Note that Resource Server (in Figure 4.14 CPSx) is authenticated itself during registration procedure by Authorization Server, then it could check the validity of token received and provide services to CPSy client.

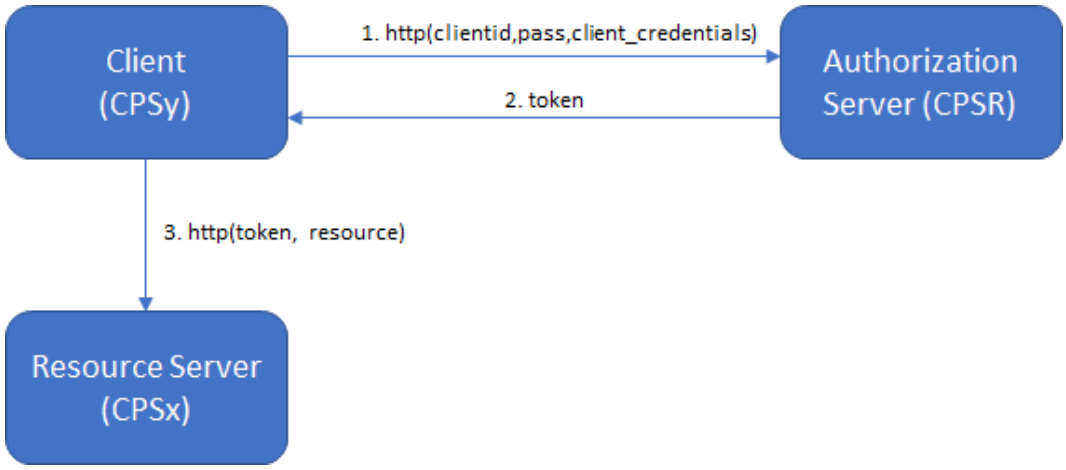

Figure 4.14 Authorization Flow

4.1.1.4 CPSR High Availability

The CPSR is a critical service in SliceNet, meaning that loss of CPSR availability in case of SW failure or hosting node crashes or extreme delays in response time during high volumes of requests, might cause overall SliceNet System failure.
In order to overcome such potential problems the following high availability architecture is proposed.

The CPSR high availability architecture is inspired to cloud-native concepts and requirements. According to this approach, the high availability requirements include both

- smooth resilience in case of failure, and
- smooth horizontal scalability in case of changing throughput demands,

so to provide a seamless behavior towards the external interfaces in all possible operational situations.

Specifically, high availability is enabled through the adoption of a clustering mechanism for the specific instances of the CPSR service, where an external Load Balancing function takes care to distribute incoming workload to the different cluster members according to configurable balancing algorithms.

The clustering operation is guaranteed by an independent clustering logic, present on all the single cluster member instances, which takes care to:

- Manage the joining of a member to the cluster;
- Manage the withdrawal of a member from the cluster;
- Manage the failure of a cluster member.

In case of the previous events the clustering logic will re-arrange the cluster topology so as to guarantee continued operativity of the CPSR service in terms of data availability and lossless fulfillment of existing transactions.

The decision to have new members joining, or existing members leaving the cluster will be taken by a specific cloud orchestration logic.

In order to achieve a cloud-native behavior based on a cluster architecture, a distributed DB architecture is also needed. Such DB can be either in-memory or based on permanent storage, but in both cases, it will be itself based on a clustering mechanism, and will appear to the CPSR cluster members as a single and consistent DB storage.

The picture below shows the mentioned CPSR architecture.

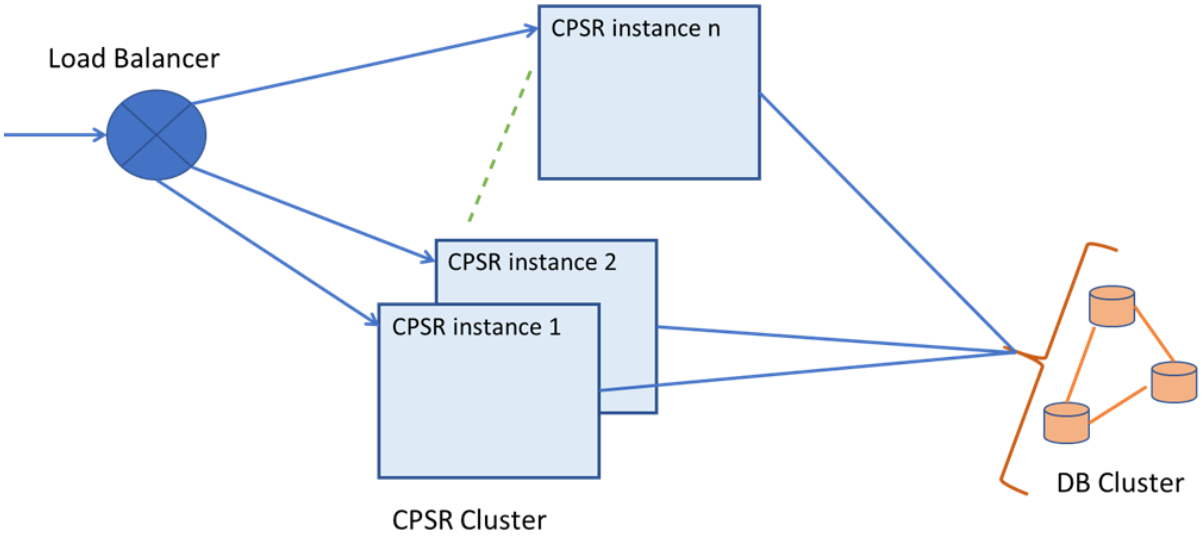

Figure 4.15 CPSR High Availability Architecture

A possible implementation of this architecture also leveraging on available Open Sources is proposed in section 6.1.2

4.1.2 InterPop Connection Control Service

The interPoP Connections (IPC) CPS replaces the original Forwarding Graph Enabler (FGE) component originally included in the SliceNet CP architecture described in D2.3. In particular, the IPC is responsible, for each slice instance, to deliver a proper interconnection of the slice Network Functions (i.e. mostly VNFs and MEC applications) deployed in different segments and domains, namely edge (e.g. MEC) and Core ones. While it is assumed that per-slice intraPoP Network Functions are properly interconnected by means of forwarding graph enforcement and provisioning features of the Resource Orchestrator (as described in deliverable D2.4 [3] , the IPC CPS allows geographically distributed slice Network Functions to be properly interconnected according to their end-to-end forwarding graph requirements.

Therefore, with respect to the original FGE component, the IPC CPS scope is focused to interPoP network provisioning to fulfil intra-domain slice requirements in terms of interconnection of constituent Network Functions across core and edge PoPs, with specific QoS and topology (i.e. forwarding graph) constraints.

For the provisioning of interPoP connections, the IPC CPS is expected to receive per-slice network interconnection requests from the Resource Orchestrator or the Slice Orchestrator and interact with the appropriate underlying network Adapters (i.e. Backhaul Adapter) for handling those requests. For this, with reference to Figure 3.1, the IPC CPS exposes the CP-IPC-S interface and related services at its northbound, and mostly leverages on the CP-BKHL-A interface and related services at its southbound.

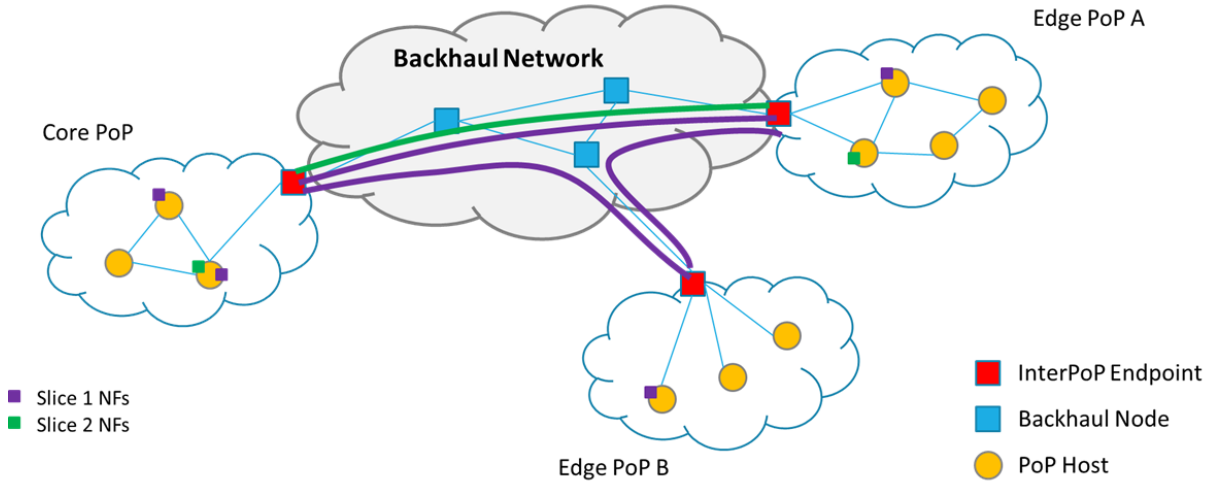

Figure 4.16 Backhaul Network

As depicted in Figure 4.16 the interPoP Connections consists of set of the pairs of endpoints on the PoPs to be interconnected for a given slice and the related constraints the interconnecting paths have to satisfy. In the figure above the interPoP Connections for a slice describe the pair of public IP addresses of the gateways (i.e. endpoints represented with red squares) on each PoP to interconnect, the constraints in terms of QoS to take in consideration along the path in the Backhaul interconnecting segment. For a slice there might be more pairs of PoPs to interconnect.

A general descriptor of an InterPoP Connection, received and processed by the IPC CPS, contains the slice identifier and the list of InterPoP paths to be interconnected through the Backhaul network:

- Slice Id: univocally identifies the slice through the backhaul network.
- InterPoP Paths: are the set of paths across the backhaul network. Each InterPoP path consists of:
    - a pair of endpoints: an endPoint on the PoP consists basically of an IP address and an encapsulation identifier (e.g. VLAN) to identify the per-slice incoming/outgoing traffic.
    - a set of constraints: bandwidth and latency to evaluate along the path interconnecting the endpoints.

Further details about data types are described in section 5.4.

The Inter-PoP Path descriptor is based on IP endPoints only so to abstract details of different underlaying backhaul technologies according to the Control Plane abstraction layers principles.

4.1.2.1 IPC Service based interface

The IPC CPS Service Based Interface is exposed as part of the SliceNet CP technology agnostic APIs, and offer the following operations to its service consumers (mostly SliceNet management and orchestration components):

- Provision InterPoP Connections, to create a new interPoP connection among two or more PoPs where the Network Functions (i.e. VNFs and MEC applications) of a given slice have been deployed and provisioned;
- Update InterPoP Connections, to dynamically update an existing interPoP connection, mostly intended for on-demand modification of one or more of the constraints related to the network connectivity for the given slice, like QoS parameters but also endpoint traffic identification attributes (e.g. IP addresses or VLAN identifiers) if applicable;

- Remove InterPoP Connections, to delete an existing interPoP connection, e.g. as a consequence of an overall slice decommissioning/termination operation;

These IPC CPS exposed operations derive from a substantial revision and grouping of the original technology agnostic FGE interface main operations described in D2.3. In particular, the following Table 4.2 the relate and map the IPC CPS listed above with the set of D2.3 FGE operations exposed through the technology agnostic APIs.

Table 4.2 Mapping between IPC CPS operations and D2.3 FGE operations

IPS CPS exposed operations	Original FGE exposed operations (ref. D2.3)
Provision InterPoP Connection	Provision Forwarding Graph
Update InterPoP Connection	Add nodes to Forwarding Graph Remove nodes from Forwarding Graph Provision link in Forwarding Graph Remove link in Forwarding Graph Configure routing scheme
Remove InterPoP Connection	Delete Forwarding Graph

4.1.2.1.1 Provision InterPoP Connections

This operation allows to properly interconnect two network functions in a slice between PoPs intra single-domain, through an interPoP Connections descriptor representing the Backhaul SDN network between infrastructure pillars.

Whenever required, e.g. if the slice orchestration or the resource orchestration components described in D2.4 (e.g. NFV and MEC orchestrator) [3] need to enforce a set of slice interPoP Connections across NFV infrastructures PoPs (e.g. managed by one or more Virtualized Infrastructure Managers), the IPC CPS offers this operation by interacting with infrastructure backhaul adapters at the SliceNet CP southbound to enforce the proper forwarding rules and thus enable the end-to-end (still at intra-domain level) forwarding graph for the whole set of Network Functions pertaining the given slice.

The Service Consumer shall send a POST request to the IPC CPS to the resource URI representing the inter-PoP Connections for a concerned Slice Id. The request body shall include the interPoP Connections, as detailed in Section 5.4

On success, "200 OK" shall be returned as result. The response body shall be empty.

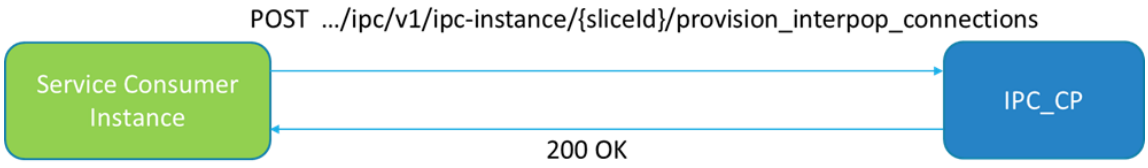

Figure 4.17 Provision InterPoP Connections

4.1.2.1.2 Update InterPoP Connections

During the lifecycle of a slice instance, the set of Network Functions (i.e. VNFs and MEC applications) deployed at core and edge locations may need to follow, in terms of location where they are running, the evolution of the vertical service (or use case) that the slice is supporting. Moreover, the slice

requirements in terms of performance (e.g. QoS requirements) may also vary and depend on the evolution of the vertical service, and therefore this could be reflected into the need of dynamic modifications of interPoP connections QoS and performance attributes. Therefore, in general, the lifecycle of a slice instance includes runtime modifications and structural changes that may result into:

1. deployment of new Network Functions in new locations (e.g. a new edge PoP),
2. deletion/termination of Network Functions running in a given location (e.g. an edge PoP close to a geographical area where there are no more end-users for the given slice),
3. migration of existing Network Functions from one location to another (e.g. across edge PoPs for mobility purposes),
4. dynamic modification of InterPoP connection performance and QoS constraints (e.g. bandwidth, latency, etc).

The first three modification options above (i, ii and iii) have an impact in the end-to-end topology of the slice, in terms of how the Network Functions are interconnected in a geographically distributed graph topology. However, these runtime modifications can be implemented from an IPC CPS perspective as a collection of Provision and Remove of interPoP connections, properly managed and coordinated by SliceNet slice and resource orchestration components. In other terms, we assume that for these i, ii and iii the SliceNet orchestration components have the required logic and the information (e.g. leveraging on instantiated services and slices detailed information, in terms of used resources, network functions, network connectivity information) to map an update of the end-to-end topology of the slice into a set of Provision or Remove of interPoP connections. For what concerns the option iv, that refers to dynamic updates of interPoP QoS constraints, the IPC CPS is conceived to expose a dedicated update operation, that will not lead to a modification of the end-to-end topology of the slice, but in case only to a modification of the connection path within the backhaul segment.

The Service Consumer, that in this case can be the slice orchestrator or the Resource orchestration in the SliceNet orchestration plane described in D2.4, shall send a PUT request to IPC_CP to the resource URI representing the inter-PoP Connections to be modified for a concerned Slice Id. The request body shall include the inter-PoP Connections to be updated along with the constraints to be modified, as detailed in Section 5.4

On success, "200 OK" shall be returned as result. The response body shall be empty.

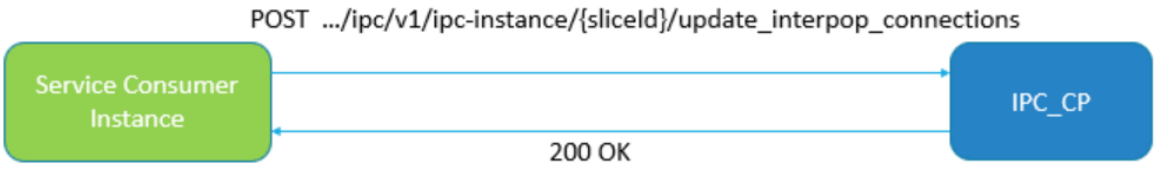

Figure 4.18 Update InterPoP Connections

4.1.2.1.3 Remove InterPoP Connections

This service operation is the counterpart of the “Provision InterPoP Connections” one and it is conceived to be used to delete whole slice inter-PoP Connections, thus coordinating the interactions with underlying network Adapters (i.e. Backhaul Adapter) to decommission the related forwarding rules. IPC receives requests from the Resource Orchestrator or any other authorized SliceNet functional component.

It is executed by deleting the given inter-PoP Connections for a Slice Id Figure 4.19). The operation is invoked by issuing a DELETE request on the URI representing the interPoP connections. The request body shall include the inter-PoP Connections to delete.

On success, "200 OK" shall be returned. The response body shall be empty.

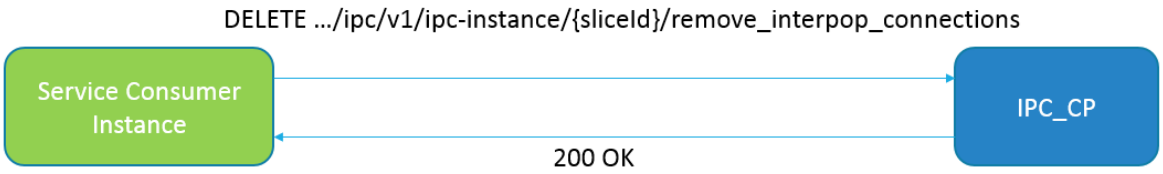

Figure 4.19 Remove InterPoP Connections

4.1.2.2 IPC internal architecture and functionalities

IPC Control is internally structured in several SW modules in charge to cover specific functionalities.

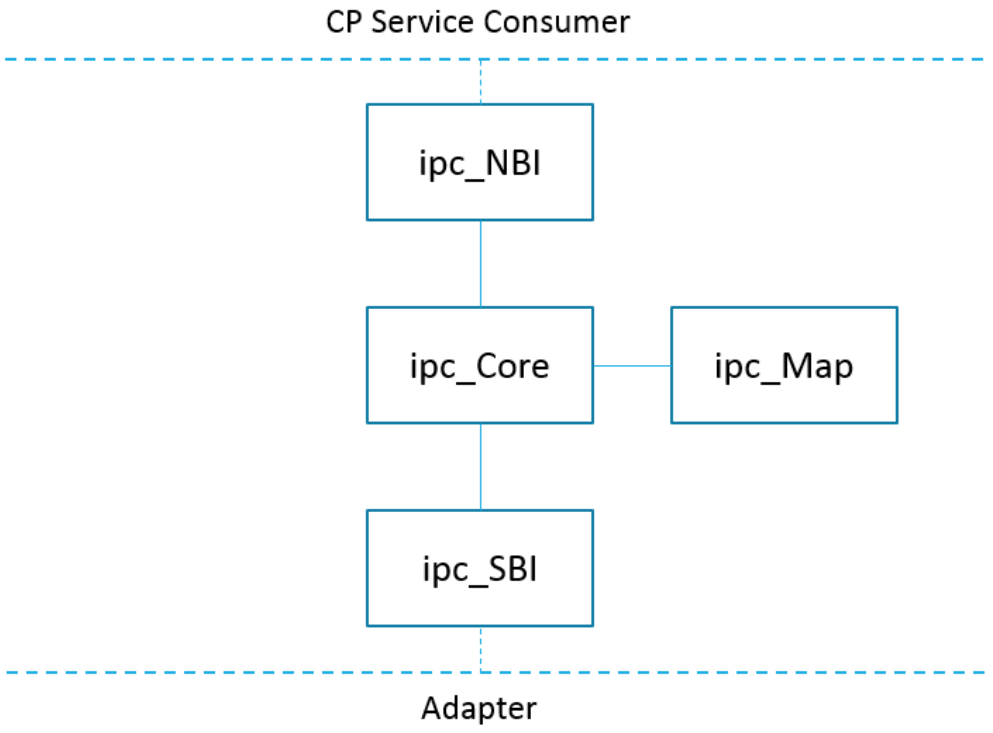

Figure 4.20 InterPoP Connections internal structure

- *ipc_NBI* handles the northbound API interface with other control plane service consumers for registering IPC instance to CPSR, for receiving the inter-PoP Connections related requests for a slice.
- *ipc_SBI* handles the southbound API interface towards the Backhaul adapter for a slice, for handling the intent-based operations.

- *ipc_Map* has the logic for mapping the inter-PoP Connections information to Intent information and vice versa.
- *ipc_Core* is the engine module which handles all service operations demanded to the IPC Control Service, interworking with the other IPC internal SW modules for registering the slice IPC instance identifier to CP Service Register, controlling the service operations from northbound to southbound interface. It is stateless, so it is supposed to do not store any data about slice and related service operations ongoing.

4.1.2.3 IPC use cases workflows

This section describes the interworking between IPC slice instances and other CPSs in order to handle the interPoP connections towards the underlying backhaul adapters. The actors involved in the IPC CPS workflows are: slice orchestration and (NFV / MEC) resource orchestration components (as main triggers of the workflows), the CPSR, the IPC CPS services, the backhaul adapters, the set of SDN controllers responsible for the configuration of the backhaul network (they could be in principle more than one).

The main pre-requisites are:

- IPC_CP instances up and running, there is 1-to-1 relationship between IPCs and Slices, it means one IPC instance for each slice;
- Backhaul Adapters are registered into CPSR;
- Slice components (e.g. NFV Network Services, VNFs, MEC applications)
- instantiated on all network segments, as performed by slice and (NFV / MEC) resource orchestration components;

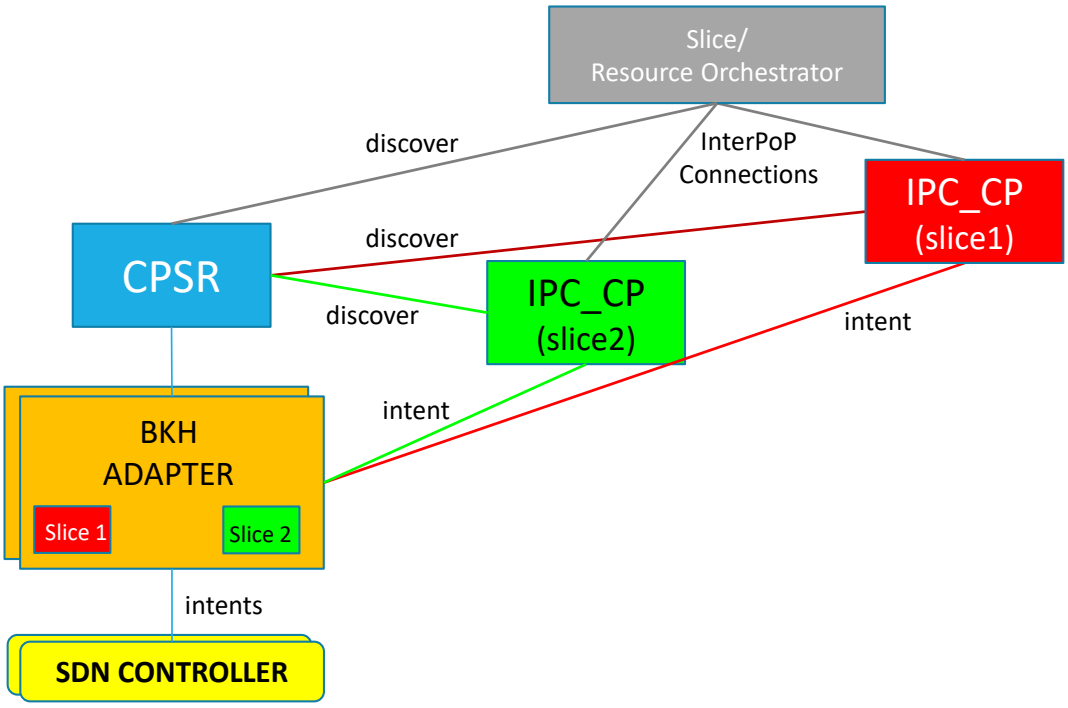

Figure 4.21 IPC use cases workflows

4.1.2.3.1 Provision InterPoP Connections

The InterPoP connection provisioning workflow is shown in Figure 4.22, and it describes the procedure to enable the interconnection of Network Functions deployed in multiple remote PoPs through a backhaul network. The workflow is triggered by the slice orchestration and (NFV / MEC) resource orchestration components soon after the individual sub-slices (as set of Network Functions in the different segments/PoPs) have been deployed, and it is implemented through the following main steps:

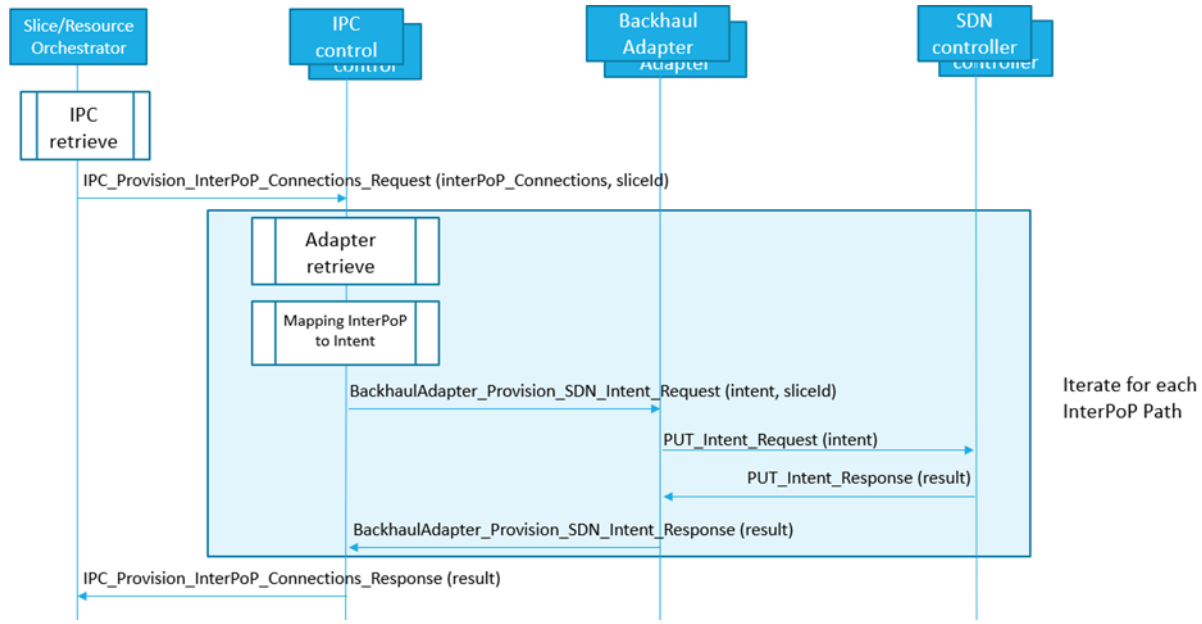

Figure 4.22 Provision InterPoP Connections flow

1. Slice Orchestrator/Resource Orchestrator retrieves IPC_CP address, via CPSR discovery function: inputs are NF_Type = "IPC_CP" and slice Id.
2. MP/RO calls IPC_CP API to provide InterPoP Connections for the specific slice Id.
3. IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
4. IPC_CP maps InterPoP Connections data to Intent data.
5. IPC_CP calls Backhaul Adapter API to provide SDN Intent
6. Backhaul Adapter calls Backhaul SDN Controller API for provision Intent
7. Operation result is returned back up to the Slice Orchestrator/Resource Orchestrator.

4.1.2.3.2 Update InterPoP Connections

The InterPoP connection update workflow is shown in Figure 4.23, and it refers to the procedure for the dynamic modification of a given slice interconnection among PoPs in support of an evolution of the slice characteristics, mostly in terms of QoS attributes. The workflow is triggered by the slice orchestration and (NFV / MEC) resource orchestration components whenever the slice requirements in terms of performances have to be dynamically modified, e.g. to increase its capacity in terms of bandwidth. This operation is normally issued as part of the cognition loops implementation within the SliceNet management and orchestration platform. The workflow is implemented through the following main steps:

1. The Slice Orchestrator or the Resource Orchestrator discovers the IPC CPS instance to invoke for the dynamic interPoP connection update operation, through a proper interaction with the CPSR.
2. The Slice Orchestrator or the Resource Orchestrator invokes the selected IPC CPS instance and issue an IPC_Update_InterPoP_Connections_Request. This request contains (in addition to the slice identifier) the list of interPoP connections to be updated, and for each of them it includes the new constraints to be applied and guaranteed in the backhaul segment, in terms of QoS performances
3. The IPC CPS service retrieves the original interPoP connection affected by the update operation, to ease the mapping to the SDN intents originally issued to the backhaul adapter(s)
4. In parallel, the IPC CPS service also retrieves the backhaul adapter(s) it was previously invoking for the provisioning of the interPoP connection affected by this update operation. Optionally, the IPC CPS can check the status of the adapter(s) directly on the CPSR.
5. The IPC CPS service maps the updated interPoP connection constraints, in terms of QoS requirements and attributes, to a new SDN intent to be issued to the backhaul adapter(s)
6. The IPC CPS service invokes the selected backhaul adapters to issue a BackhaulAdapter_Update_SDN_Intent_Request. This request contains (in addition to the slice identifier) the details of the new constraints to be applied to the original SDN intent identifier created at the provisioning phase. According to the specific nature and endpoints of the update request, more than one backhaul adapters may be required to be invoked, e.g. to cover the case of backhaul networks controlled by more than one SDN controller (e.g. in case of multi-vendor backhaul network)
7. The Backhaul adapter receives the SDN intent update request, and match the original intent identifier to retrieve the operations involved on the given SDN controller. In parallel, the adapter also produce a new intent according to the constraints received from the IPC CPS service. Then, it proceeds to delete the original intents, and to enforce the new ones in support of the update request received. For the sake of service continuity, the Backhaul adapter may follow a make-before-break approach, thus creating a new intent first, and deleting the old one later. This is subject to internal adapter policies.
8. At this point, the result of the SDN Intent update operation is forwarded back to the IPC CPS service
9. The steps from 3 to 8 are iterated for each interPoP connection to be updated and included in the request received at step 2.

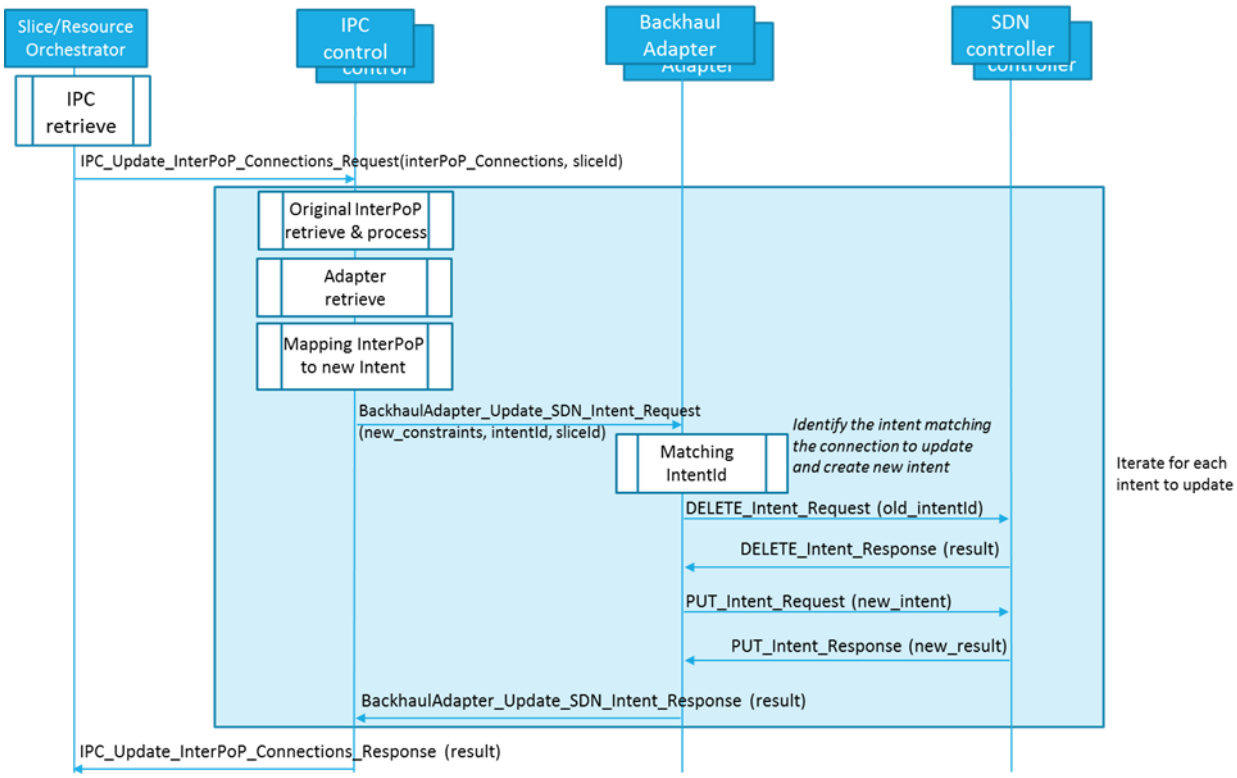

Figure 4.23 Update InterPoP Connection

4.1.2.3.3 Remove InterPoP Connections

The InterPoP connection termination workflow is shown in Figure 4.24, and it implements the procedure for removing an existing network connectivity service in the backaul segment that serves for the interconnection of slice Network Functions deployed in multiple PoPs. The workflow is triggered by the slice orchestration and (NFV / MEC) resource orchestration components as part of an overall slice decomissioning operation, and it is implemented through the following main steps:

- Slice Orchestrator/Resource Orchestrator retrieves IPC_CP address, via CPSR discovery function: inputs are NF_Type = "IPC_CP" and slice Id.
- Slice Orchestrator/Resource Orchestrator calls IPC_CP API to delete Inter PoP Connections for the specific slice Id.
- IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
- IPC_CP maps InterPoP Connections data to Intent data (e.g. mapping the IP addresses received in the InterPoP Path to the endpoints of an Intent).
- IPC_CP orders Backhaul Adapter via API to remove the Intent.
- Backhaul Adapter matches the received endpoints with intent ID.
- Backhaul Adapter requests Backhaul SDN Controller API to delete SDN Intent ID

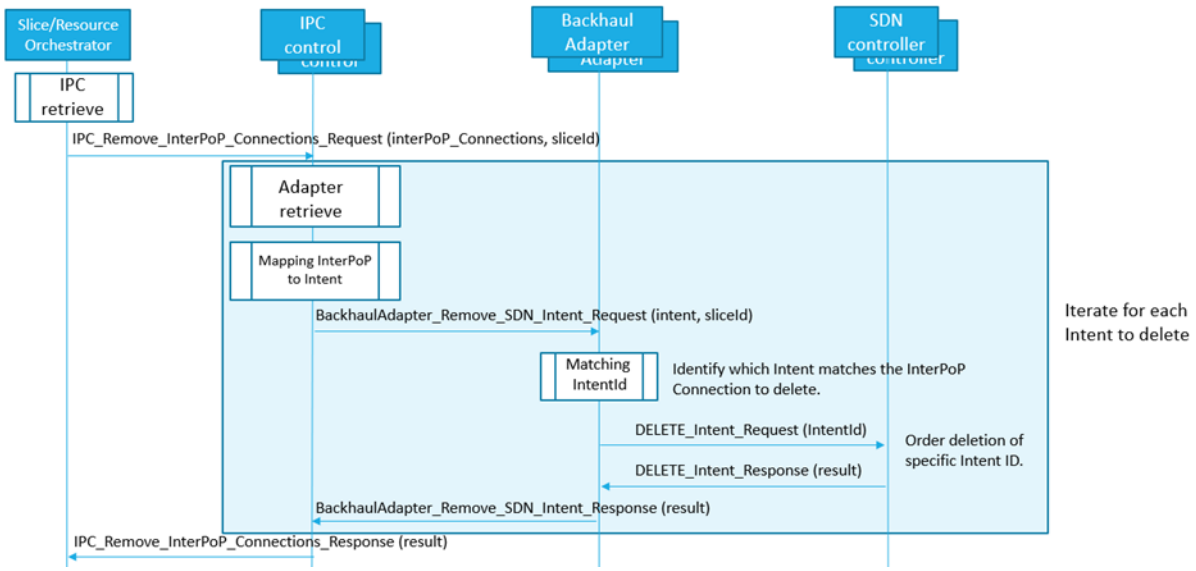

Figure 4.24 Remove InterPoP Connection Flow

4.1.3 QoS Control Service

QoS Control Service provides the SW component implementing the SliceNet distributed per-slice access for dynamic QoS setting. The QoS Control Service deploys the request to the network segments according to the input parameters in the exposed interfaces.

Figure 4.25 highlights the components that can be involved in the QoS control operations:

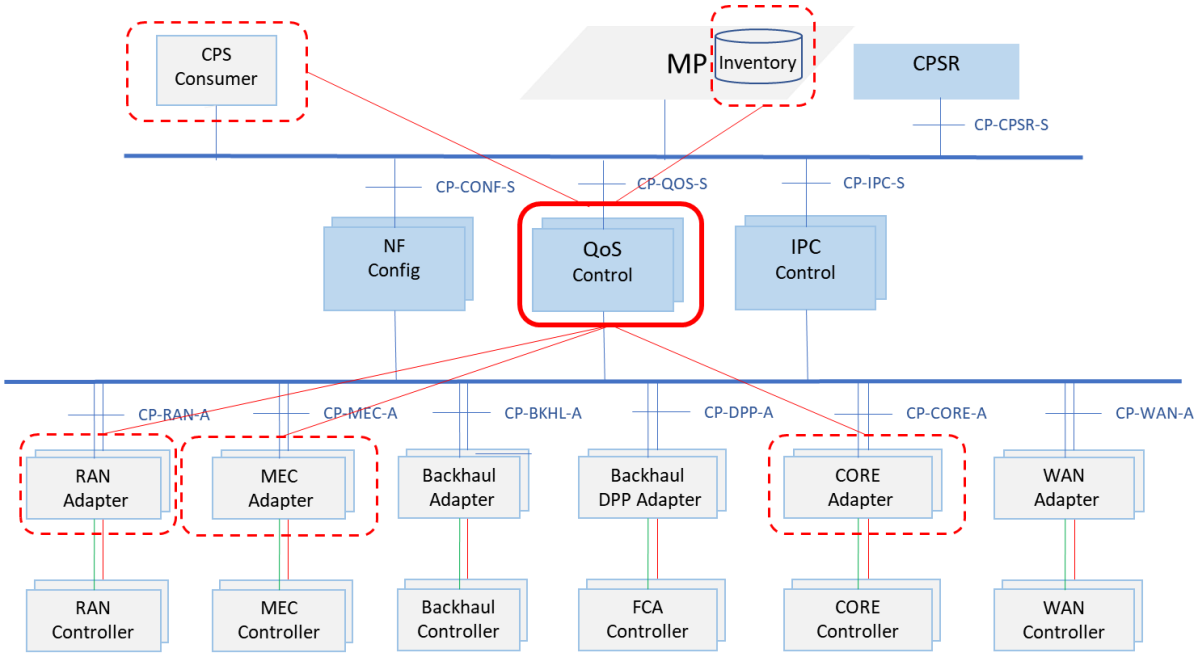

Figure 4.25 Control Plane components involved by QoS

4.1.3.1 QoS Service Based Interface

Operations offered by QoS control to apply specific slice-aware control logics:

- Set *QoS* constraints
- Set Priority

The request to set the QoS constraints can be on a per-slice basis or for an IMSI and optionally the EPS bearer ID associated with a slice. In principle, the request to set the QoS constraints is deployed to all network segments depending on the slice composition.

The request to set the priority can be, instead, on a per-slice basis only. Depending on the flow information retrieved by the INVENTORY, the priority can be set for all flows through the slice or for a specific flow.

The next sections describe SliceNet-specific use cases.

4.1.3.2 QoS use cases workflow

4.1.3.2.1 Set QoS Constraints

A CPS consumer (e.g. Orchestration Sub-Plane components) can request from the QoS control service to set new QoS constraints at runtime.

Currently, the request can be addressed to the RAN and/or Core segments.

The QoS configuration on a RAN segment cannot be differentiated per UE ID and/or EPS bearer ID. The request can be only on a per-slice basis.

On the contrary, the QoS configuration on the Core segment can be differentiated per UE ID (IMSI) and, optionally, by adding the EPS bearer ID.

The Table 4.3 shows input parameters with possible combinations:

Table 4.3 Set QoS Constraints

Optionality	Parameters	QoS_RAN_ SLICE	QoS_CORE_ IMSI	QoS_CORE_ BEARED	QoS_ALL_ Segments
Required	sliceId	X	X	X	X
Optional	segmentId	X (=ACCESS)	X (=CORE)	X (=CORE)	
Optional	userEqId	(X)*	X	X	(X)
Optional	epsBearerId	(X)*		X	(X)
Required	qosConstraints	X	X	X	X

X = present
(X)* = if present to be ignored
(X) = optionally present

All the other parameters combinations results as bad request.

The parameters combinations trigger different actions in QoS as explained below:

- **QoS_RAN_SLICE**: this triggers the increase, for example, of the BW in DL and/or BW in UL for a sliceID in the RAN segments. This operation could be also called by the CONFIG CP to re-configure the QoS slice regarding the QoS constraints.
- **QoS_CORE_IMSI**: All bearers associated to the IMSI is “routed” on other bearer matching the QoS constraints.
- **QoS_CORE_BEARER**: Only the specific bearer of the specific IMSI is routed according to the QoS request.

- **QoS_ALL_**Segments: it triggers always **QoS_RAN_SLICE** operation and optionally **QoS_CORE_IMSI** or **QoS_CORE_BEARER** depending on the presence of the UE id and the bearer id.

4.1.3.2.1.1 Set QoS constraints on RAN per slice

A caller (e.g. QoE/P&P) can require to QoS control service to set the QoS constraints of the RAN segments for an instantiated slice (Figure 4.26).

The caller needs to discover the QoS control service serving the slice.

The request to set the QoS constraints is sent to QoS control service instance specifying the RAN segment and the QoS parameters.

The QoS control service discovers the RAN adapter instances serving the instantiated slice. The request is deployed to the RAN-adapters which takes care to map the operation towards the RAN-controller.

RAN adapter API and required parameters are further explained in SliceNet Deliverable D4.2 [5]

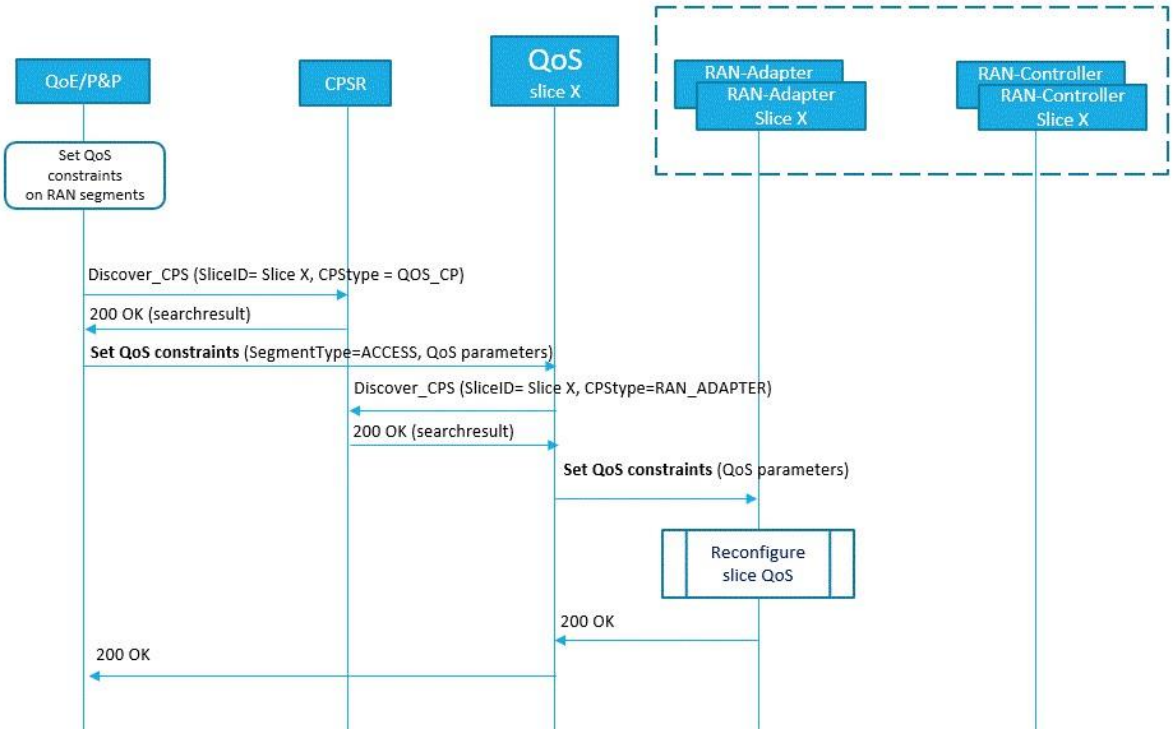

Figure 4.26 Set QoS Constraints on RAN per Slice

4.1.3.2.1.2 Set QoS constraints on CORE per IMSI

A caller (e.g. QoE/P&P) can request from the QoS control service to guarantee the QoS constraints for all EPS bearer IDs associated with an IMSI. The request is deployed to the MEC-CORE adapter.

The caller needs to discover the QoS control service serving the slice ID.

The request to set the QoS constraints is sent to QoS control service instance specifying the Core segment, the IMSI and the QoS parameters.

The QoS control service discovers the MEC-CORE adapters instance serving the instantiated slice. The request is deployed to the MEC-CORE adapters which takes care to redirect the traffic flow to bearers matching the required QoS constraints.

MEC-CORE adapter API and required parameters are further explained in D4.2 document.

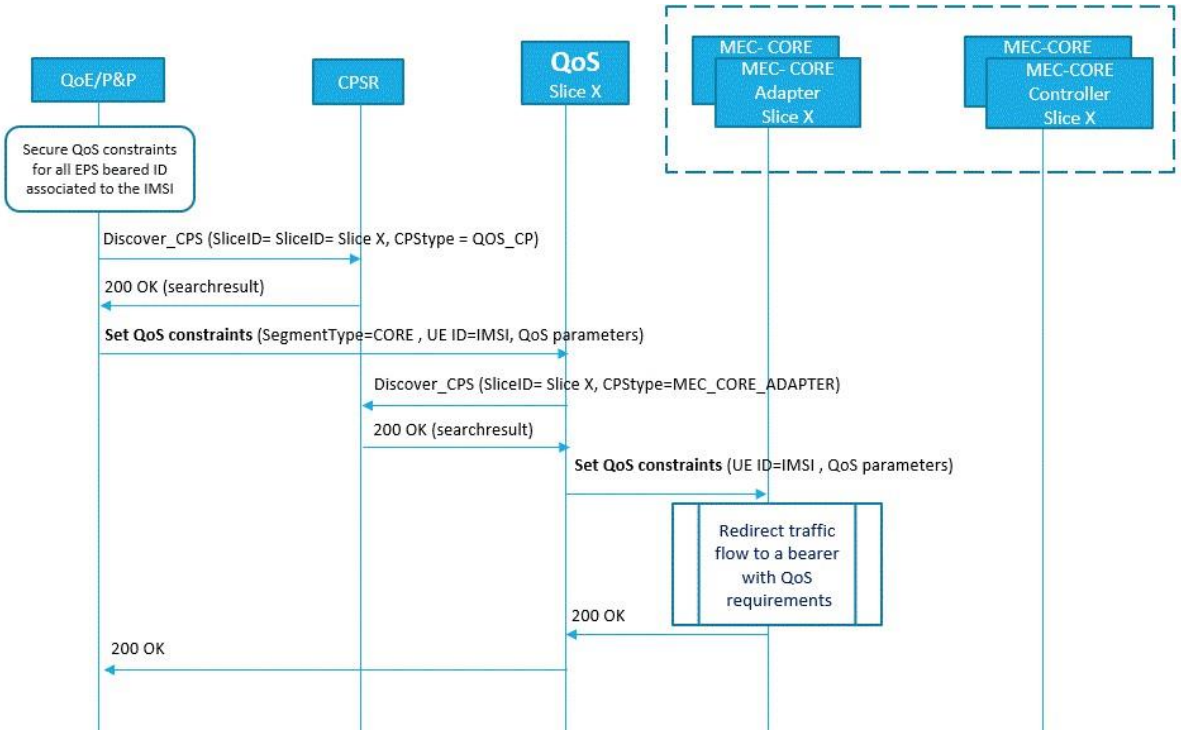

Figure 4.27 Set QoS Constraints on Core per IMSI

4.1.3.2.1.3 Set QoS constraints on CORE per Bearer ID

A caller (e.g. QoE/P&P) can require to QoS control service to secure the QoS constraints for a given bearer id associated to an IMSI. The request is deployed to the MEC-CORE adapter.

The caller needs to discover the QoS control service serving the slice ID.

The request to set the QoS constraints is sent to QoS control service instance specifying the Core segment, the IMSI, bearer id and the QoS parameters.

The QoS control service discovers the MEC-CORE adapter instance serving the instantiated slice. The request is deployed to the MEC-CORE adapter which takes care to redirect the traffic flow to bearers matching the required QoS constraints.

MEC-CORE adapter API and required parameters are further explained in SliceNet Deliverable D4.2 [5] .

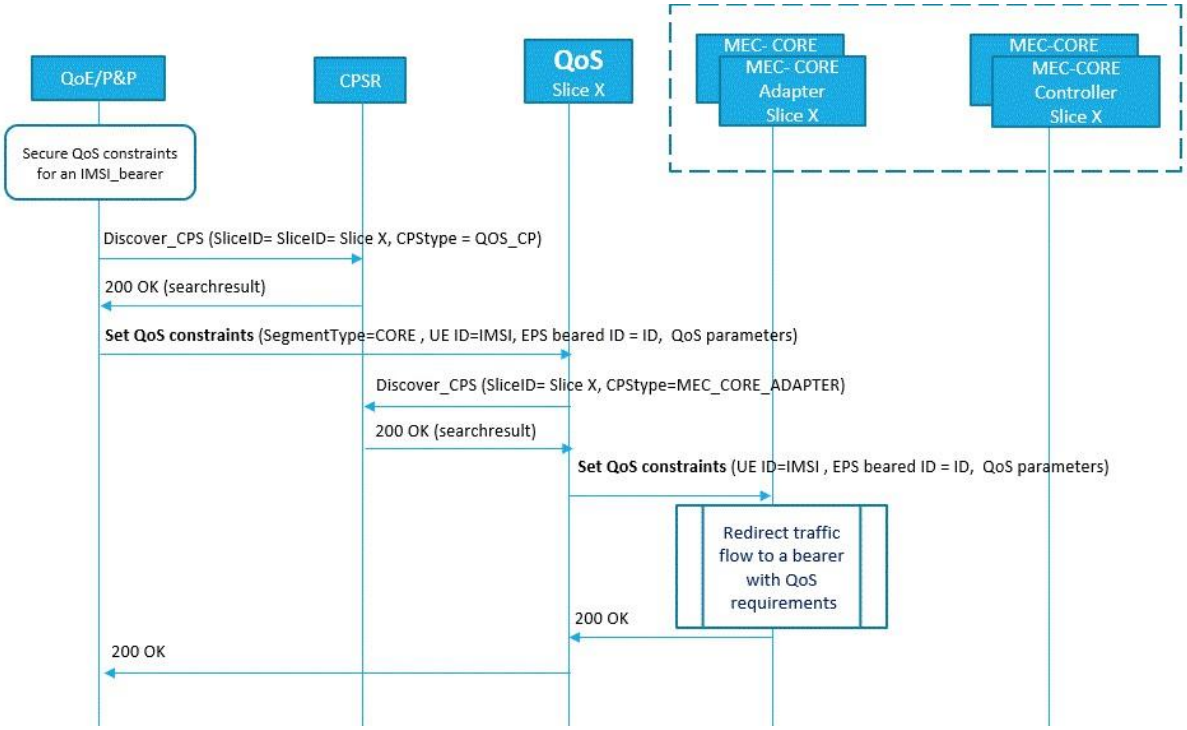

Figure 4.28 Set QoS constraints on CORE per Bearer ID

4.1.3.2.1.4 Set QoS constraints all segments

A caller (e.g. QoE/P&P) may require the QoS control service to secure the QoS constraints by specifying the slice ID and optionally the IMSI and bearer id.

If only the sliceID is given as input the request is deployed to the RAN segment only.

If the IMSI and optionally the bearer id are given the request is deployed to the Core segment.

MEC-CORE/RAN adapters API and required parameters are further explained in SliceNet Deliverable D4.2 [5] .

Figure 4.28 shows an example of the operation handling, having as input the sliceID, IMSI and bearer id.

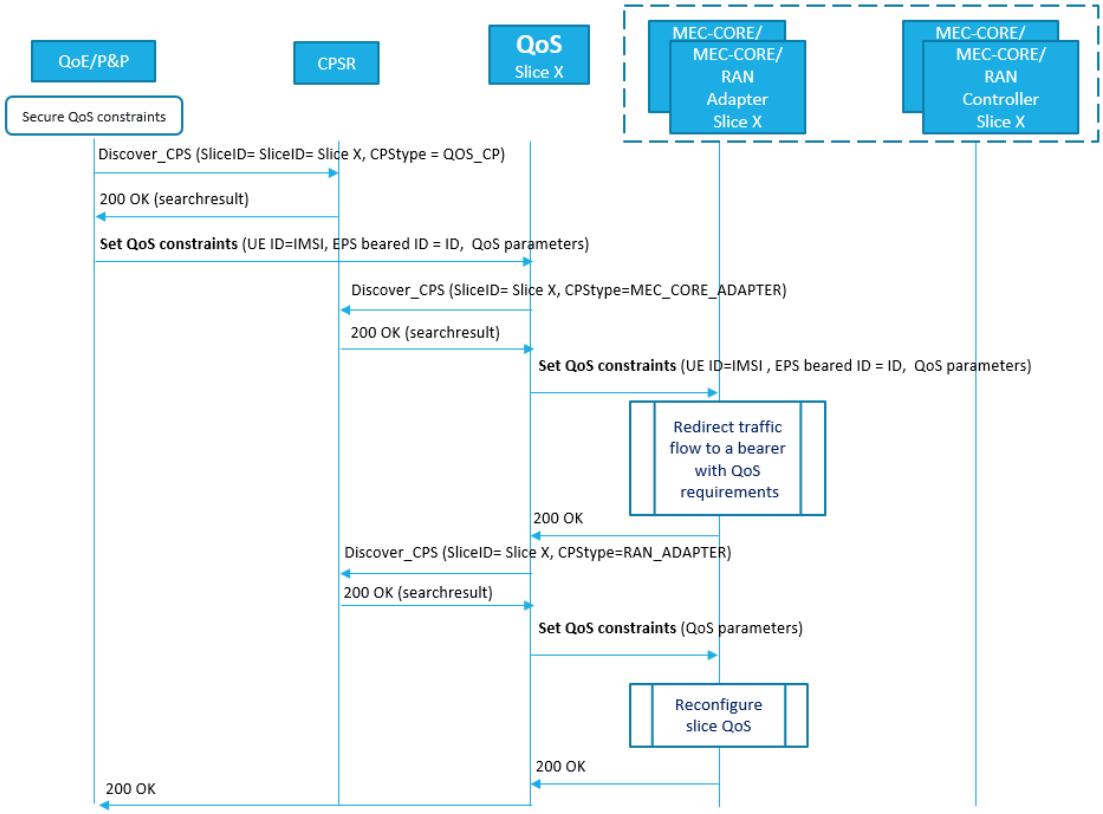

Figure 4.29 Set QoS constraints all segments

4.1.3.2.2 Set Priority

4.1.3.2.2.1 Set Priority per slice/flow

Figure 4.29 shows the workflow for changing the priority of a specific traffic crossing through the network. In summary, any authorized SliceNet functional component can discover a registered QoS Service through the CPS discovery function for changing the priority of a specific slice/flow. In order to do so, the said caller sends the priority value and the target slice id to that control plane service. QoS component is, therefore, in charge to collect required parameters (depending on the adapter) by requesting them from the Inventory. After that, QoS service uses the agnostic API exposed by the BKH_DPP_ADAPTER for setting the action (change priority) including those parameters. The BKH_DPP_ADAPTER maps the input and transforms it in an understandable format which is sent to the FCA controller. Finally, the FCA controller enforces the rule (change priority) in a specific location which changes the priority of all specified matching traffic. BKH_DPP_ADAPTER API and required parameters are further explained in Section 0.

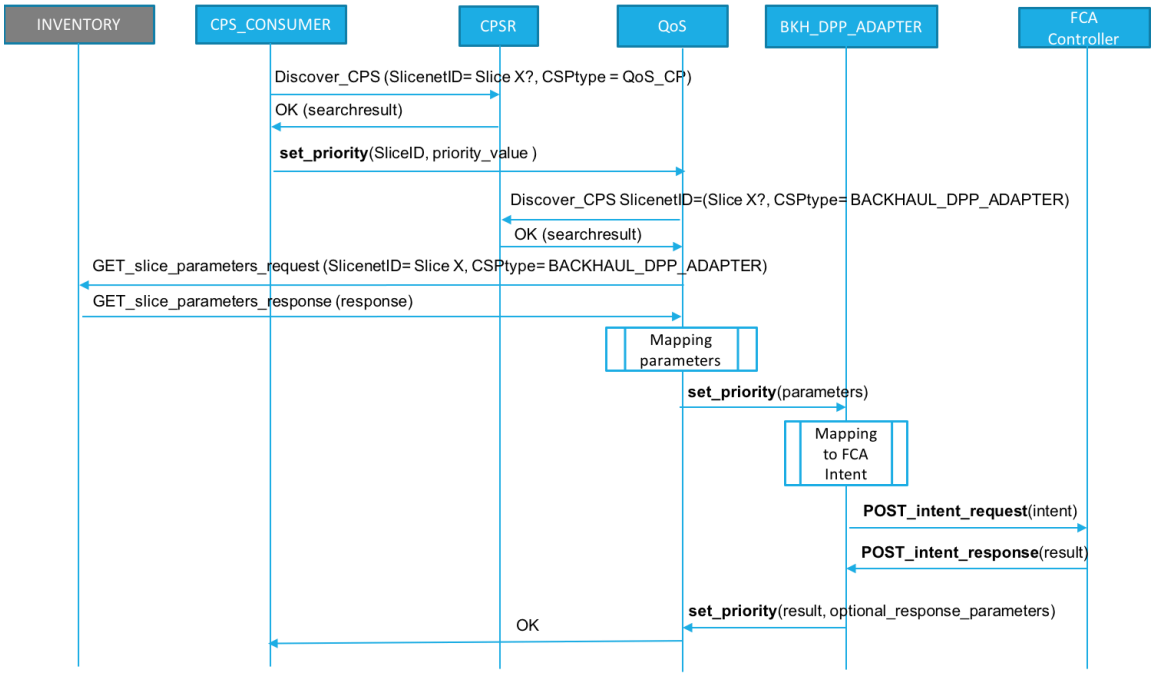

Figure 4.30 Set Priority per Slice/Flow

Following steps describe the workflow in more detail:

- A caller (CPS Consumer) retrieves Qos Control Plane Service address, via CPSR function.
  a) SliceID
  b) CPStype=QoS_CP
- This caller calls set_priority endpoint of the QoS service API to provide:
  a) SliceID
  b) priority_value
- QoS Service retrieves the Backhaul DPP Adapter address by using the CPSR discovery function.
  a) SliceID
  b) CPStype=BKH_DPP_ADAPTER

- QoS Service retrieves required parameters from the inventory in order to fulfil the request to be sent to the BKH_DPP_ADAPTER adapter.
    a) Required BKH_DPP_ADAPTER parameters are detailed in section 5.
- QoS Service maps those gathered parameters from the inventory.
- Qos Service calls the agnostic BKH_DPP_ADAPTER API in order to apply a priority value over a specified network traffic.
    a) SliceID
    b) required_parameters
- BKH_DPP_ADAPTER maps the retrieved data from its northbound interface to an intent.
- BKH_DPP_ADAPTER calls FCA controller API for provision Intent.
- FCA Controller enforces the intent to the target interface location on the data plane.
- Operation result is returned back to the initial service consumer which is acting as a caller

4.1.4 NF Config Control Service

Network functions and their dynamic management and life-cycle are expected to play a key role in the provisioning of slices. This is due to the fact that SliceNet approach follows principles defined in the context of:

- Sharing of the access medium (i.e. radio channels) based on practices defined for Multi-Operator Core Network (MOCN [40]) and for Dedicated Core Networks (DÉCOR [41]).
- Control and User Plane Separation of EPC nodes (CUPS [42]).
- SBA System Architecture for the 5G System [43]

In all these cases the key aspect is the modularity implied by the separate instantiation of control and network functions either to allow RAN sharing among several (virtual) operators like in DÉCOR and MOCN or to allow for a highly granular architecture with respect to the classification of user session among different slices as in the case of 5G SBA. SliceNet considers E2E Slicing as superset of the above mentioned setups that eventually can allow an E2E Slice to be delivered in a virtual operator fashion, i.e. each vertical is provided with a separate and dedicated core network to which shared (radio) access segments are attached without any restriction regarding the administrative domain they are belonging as long as the related NSP stakeholders support the federation of their resources under the principles of SliceNet.

By the introduction of its Control Plane [2] SliceNet has defined a way to accommodate either current (4G, NSA-5G) or future (SA-5G) technologies in an agnostic manner as long as the pillar [2] resources are attached under the abstraction layer through the appropriate adaptors and support of the initial and currently evolving adaptor NBI. It is expected that below the adaptation layer each NSP will be deploying NFs under the command of the orchestration modules for the provisioning of the slice dedicated core and access components (according to the definition of the slice along one or several NSPs). Consequently, every instantiated slice per NSP will be associated with a number of NFs either these are dynamically lifecycle managed or they are re-used from a shared pool. Each of these functions will be implementing a subset or the entire foreseen adaptor NBI interface of the pillar resource it belongs to.

Depending on the slice template/design that is instantiated a number of operations are expected to be available per slice. These operations are not limited to a predefined subset. Several operations can be defined and catalogued so that they can be included in a slice template. These operations are defined according to an extension of the configuration descriptor model that was used in 5G-PPP Phase 1 SELFNET project for the onboarding process of SDN/VNF and PNF applications [38] That model was including, among other, a configuration descriptor defining two parts, the exposed operation part that the management components are aware of and utilise through SELFNET

Application Manager [39] and the protocol section that was identifying the communication protocol (REST, CLI, Rabbit MQ, etc.) that should be used for applying the configuration to the related application. For SliceNet this descriptor is augmented so that the protocol section is defined as a set of interactions to be applied sequentially to form a workflow. The information model foresees that the arguments (supplied as key:value pairs) of the operation are used in the customisation of each one of the interactions (e.g. the body of a POST request) of the workflow. Optionally, temporary variables can be defined that are assigned specific values that can be retrieved from each of the executed transactions. Thereafter, temporary variables can be used in subsequent requests. For any of the NF Config CP Service instance(s) created for a slice, a bootstrap configuration with such descriptors is provided so that the contained workflow execution engine can customise the processing per invoked operation. During slice runtime, those management operations may be required to be applied per slice according to P&P support, cognitive processing or local (NSP) automated FCAPS actuation. Following CP principles, unless slice exposure level is quite extensive, the required configuration is applied through the NF Config Control service. The NF Config Control service is implemented to analyse the requested operation and execute it in the context of a defined workflow among those with which it has been configured during instantiation. In this way the NF Config CP Service allows extensibility and dynamicity to be applied per slice based on the features that can be used from the underlying pillar technologies.

4.1.4.1 NF Config Control Service Based Interface

The NF Config CPS exposes as many operations as the number of those that have been defined in the slice template instance and therefore there is no specific enumeration of the operations apart from a generic REST approach presented in paragraph 5.2

4.1.4.1.1 NF Config CPS Workflow

As the NF Config CPS is provided as a generic workflow execution engine an abstract use case is presented as a representative flow sequence.

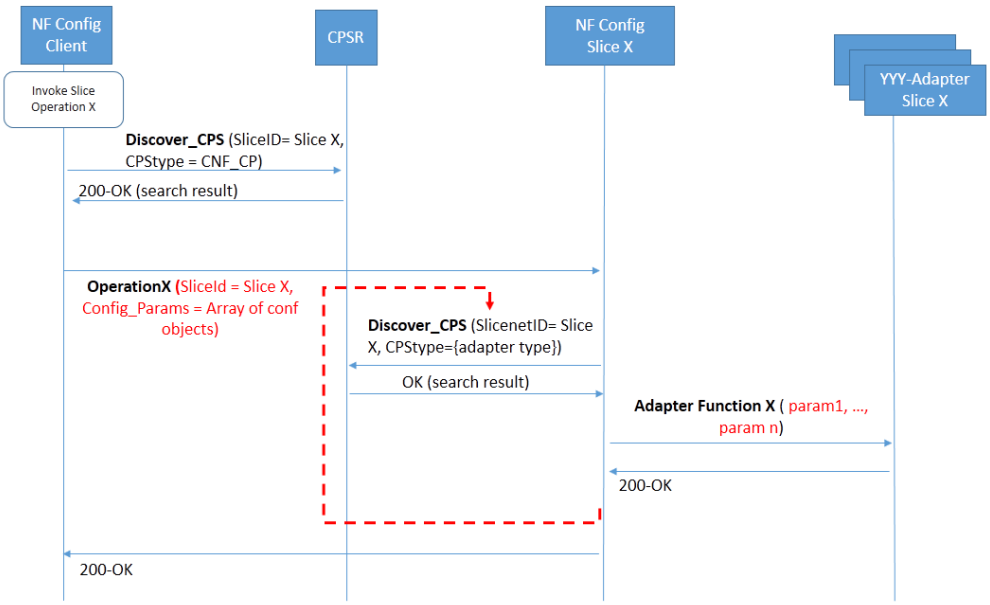

Figure 4.31 NF Config Workflow

- An NF Config client component (P&P, Cognitive or FCAPS Management component) identifies an operation that has been defined for a slice
- The CPSR is contacted to resolve the NF Config CPS that has been allocated for the specific slice
- The operation is invoked by providing the required parameter configuration set
- The NF Config CPS analyses the steps that are foreseen for applying the specific operation for the specific slice
- The NF Config CPS iterates over all the steps and after resolving the required adapter for each step it applies the configuration and invokes the adapter interface
- Once all the steps have been performed feedback is provided to the caller. Optionally a callback can be supported in this context so as to allow for asynchronous operation.

4.2 Control Plane Adapters

4.2.1 Backhaul DPP Adapter

The Backhaul DPP Adapter (BKH_DPP_ADAPTER) is a software component that provides a SliceNet centralized interface for completing a set of actions over traffic flowing across a specific point, the backhaul. That network traffic can either represents a specific flow or a slice which contains aggregated flows covering the same scope, traffic description is further explained in section 5. There is 1-to-many relationship between BKH_DPP_ADAPTER and slices, it means just one BKH_DPP_ADAPTER instance for covering several flow/flows. In order to deploy an action, BKH_DPP_ADAPTER will receive requests from the QoS Control Service or any other authorized SliceNet functional component (e.g P&P, MP) and interact with the appropriate underlying network controller (FCA Controller) for handling those requests. This component is in charge of providing the agnostic technology abstraction layer capabilities for the consumer by doing a mapping of the technology agnostic API to technology depending APIs provided by the underlying network controller.

4.2.1.1 Backhaul DPP Adapter internal architecture and functionalities

Backhaul DPP Adapter is internally structured in several SW modules in charge to cover specific functionalities, as is depicted in Figure 4.32.

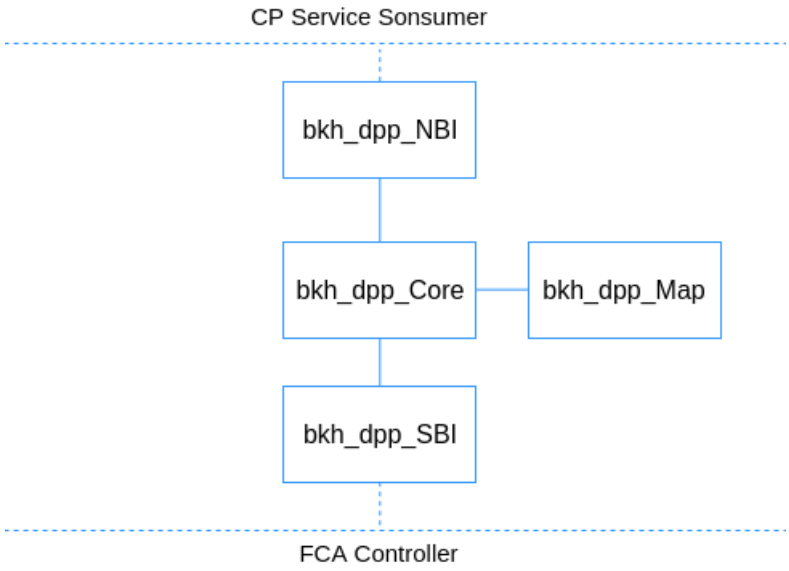

Figure 4.32 Backhaul DPP Adapter

- *bkh_dpp_NBI* handles the northbound API interface with other control plane service consumers for registering BKH_DPP_ADAPTER instance to CPSR, for receiving the action related requests for a slice.
- *bkh_dpp_SBI* handles the southbound API interface towards the Backhaul FCA Controller for a slice, for handling the intent-based operations.
- *bkh_dpp_Map* has the logic for mapping the intent related information to FCA Controller API information and vice versa.
- *bkh*_dpp_*Core* is the engine module which handles all service operations demanded to the BKH_DPP_ADAPTER, interworking with the other BKH_DPP_ADAPTER internal SW modules for registering the slice BKH_DPP_ADAPTER instance identifier to CP Service Register, controlling the service operations from northbound to southbound interface. It is stateless, so it is supposed to do not store any data about slice and related service operations ongoing.

4.2.2 Backhaul Adapter

The Backhaul Adapters provide a first level of abstraction over the transport network pillar functionalities which is further exploited and abstracted by CP Services which expose specific services by their own Service Based Interface (SBI). The Backhaul Adapters translate the Controllers northbound interface into a technology agnostic Interface, thus enabling a common SliceNet CP information model and control logics, hiding the slice detailed composition in terms of network segments and vendors technology.

The Backhaul Adapter maps the sliceID to the ONOS criteria parameter V-LAN id to allow the SDN controller isolation of traffic within backhaul network.

The Backhaul Adapters expose the northbound interface for interconnecting multiple Point-of-Presences (PoP) on infrastructure segments intra domain for a slice.

4.2.2.1 Backhaul Adapter Service Based Interface

Operations offered by Backhaul Adapter to apply specific slice aware control logics:

- Provision SDN Intent
- Update SDN Intent
- Remove SDN intent
- SDN Topology Exposure

Pre-requisites:

- Backhaul adapter instance up and running, 1-to-1 relationship with SDN backhaul controller for each Slice ID.
- Backhaul Adapters registered into CPSR
- Sub-slices instantiated on all network segments, performed by Management Plane (MP)

4.2.2.1.1 Provision SDN Intent

An SDN intent may be expressed as a connectivity service between two endpoints with given QoS requirements and with a given routing scheme (e.g. an high level firewall rule to block specific user traffic at a border of a network domain). This operation therefore exposes what to do on top of the SDN controllers, without specifying how to do it.

The Service Consumer shall send a PUT request to BKH_ADAPTER to the resource URI representing the Intent for a concerned Slice Id. The request body shall include the Intent object.

On success, "200 OK" shall be returned as result. The response body shall be empty.

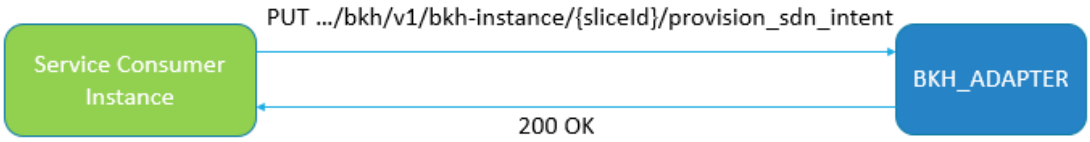

Figure 4.33: Provision SDN Intent

4.2.2.1.2 Update SDN Intent

A runtime update of a provisioned Intent is required to fulfil the dynamicity in the life-cycle of the slice instances, in particular to satisfy the needs to change the quality of service constraints for an intent-based connection between two end-points of backhaul SDN network.

The Service Consumer shall send a PUT request to BKH_ADAPTER to the resource URI representing the Intent to be updated for a concerned Slice Id. The request body shall include the modified Intent object.

On success, "200 OK" shall be returned as result. The response body shall be empty.

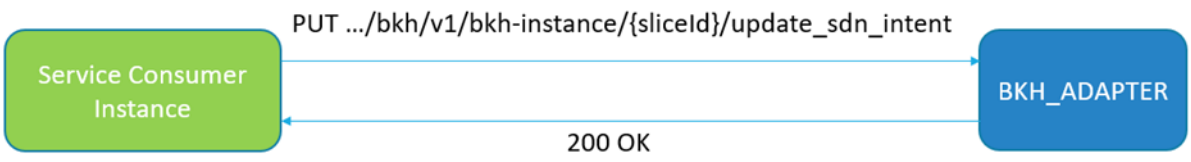

Figure 4.34 Update SDN Intent

4.2.2.1.3 Remove SDN Intent

This operation is the counterpart of the above “Provision SDN intent”. It follows the same declarative approach and it has to be used to enforce the decommissioning of a given SDN intent by means of the underlying Backhaul SDN controller. The decommissioning logic is mandated to the SDN controller, aiming to reduce conflict at the SliceNet CP control services level and increase the success of the requests.

The Service Consumer shall send a DELETE request to BKH_ADAPTER to the resource URI representing the Intent for a concerned Slice Id. The request body shall include the Intent object.

On success, "200 OK" shall be returned as result. The response body shall be empty.

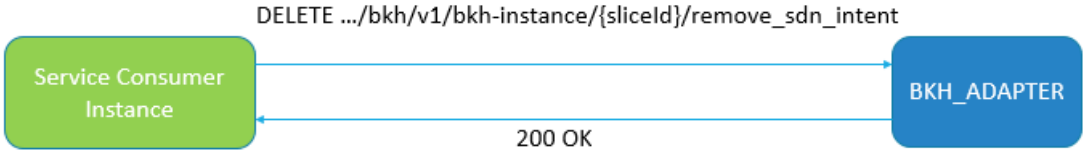

Figure 4.35 Remove SDN Intent

4.2.2.1.4 SDN Topology Exposure

This operation is conceived to expose a logical view of the backhaul/transport network segment topology to other SliceNet CP control services for a slice. Following the SliceNet CP abstraction principles, the SDN topology view offered through this operation is independent from specific SDN controller(s) technologies and models, and it consists of listing the endpoints (and their attributes) of the PoPs on the infrastructure network segments which have been interconnected through the Backhaul SDN network controller.

The slice topology exposed by the Backhaul Adapter consists of the list of pairs of endpoints (and their attributes) used for Intent provisioning towards the Backhaul SDN Controller.

The CP Service Consumer shall send a GET request to BKH_ADAPTER for a concerned Slice Id. The request body shall be empty.

On success, "200 OK" shall be returned as result and the response body shall contain the topology with the list of inter-PoP endpoints.

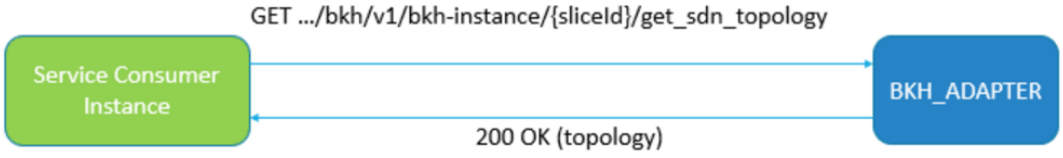

Figure 4.36: Get SDN Topology

4.2.2.2 Backhaul Adapter internal architecture and functionalities

Only one Backhaul Adapter instance (per SDN Controller vendor technology) is registered into CPSR, it interworks with more CP control service instances for handling more slices on the northbound side and with concerned SDN controller on the southbound side.

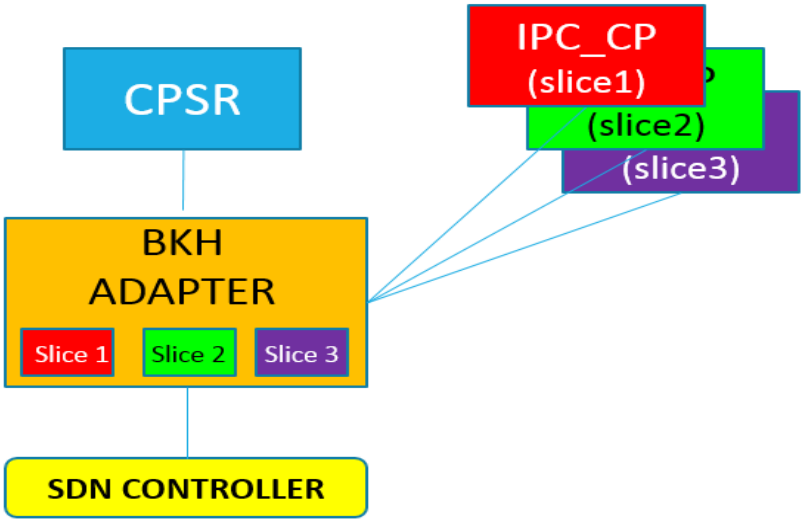

Figure 4.37 Backhaul Adapter interworking

The Backhaul Adapter is internally structured in several SW modules in charge to cover specific functionalities.

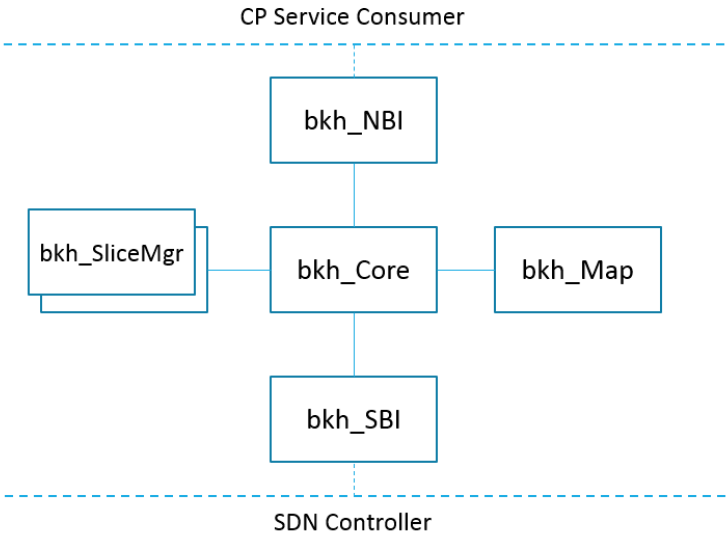

Figure 4.38 Backhaul Adapter internal structure

- *bkh_NBI* handles the northbound API interface with other control plane service consumers for registering BKH_ADAPTER instance to CPSR, for receiving the intent related requests for a slice.

- *bkh_SBI* handles the southbound API interface towards the Backhaul SDN Controller for a slice, for handling the intent-based operations.
- *bkh_Map* has the logic for mapping the intent related information to SDN Controller API information and vice versa.
- *bkh_SliceMgr* manages the information regarding each slice instance inside the Backhaul Adapter, keeps track at slice instance level of data for interworking with the CP Service Consumer instance (e.g. IPC_CP slice instance) on NBI side and the Backhaul SDN Controller on SBI side.
- *bkh_Core* is the engine module which handles all service operations demanded to the BKH_ADAPTER, interworking with the other BKH_ADAPTER internal SW modules for registering the slice BKH_ADAPTER instance identifier to CP Service Register, controlling the service operations from northbound to southbound interface. It is stateless, so it is supposed to do not store any data about slice and related service operations ongoing.

4.2.2.3 Backhaul Adapter use cases workflows

There is one Backhaul Adapter for each SDN Controller type (e.g. ONOS, OpenDayLight). The following workflows are intended for an adapter using ONOS on its southbound interface, they might be slightly different in case of another SDN controller.

4.2.2.3.1 Provision SDN Intent

- IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
- IPC_CP calls Backhaul Adapter API to provide SDN Intent for Slice Id.
- BKH_ADAPTER gets list of available hosts from Backhaul SDN controller.
- BKH_ADAPTER maps the endpoints received from IPC_CP into corresponding Backhaul SDN controller Host IDs.
- BKH_ADAPTER calls Backhaul SDN Controller API for Intent provisioning.
- Operation result is returned to IPC_CP.

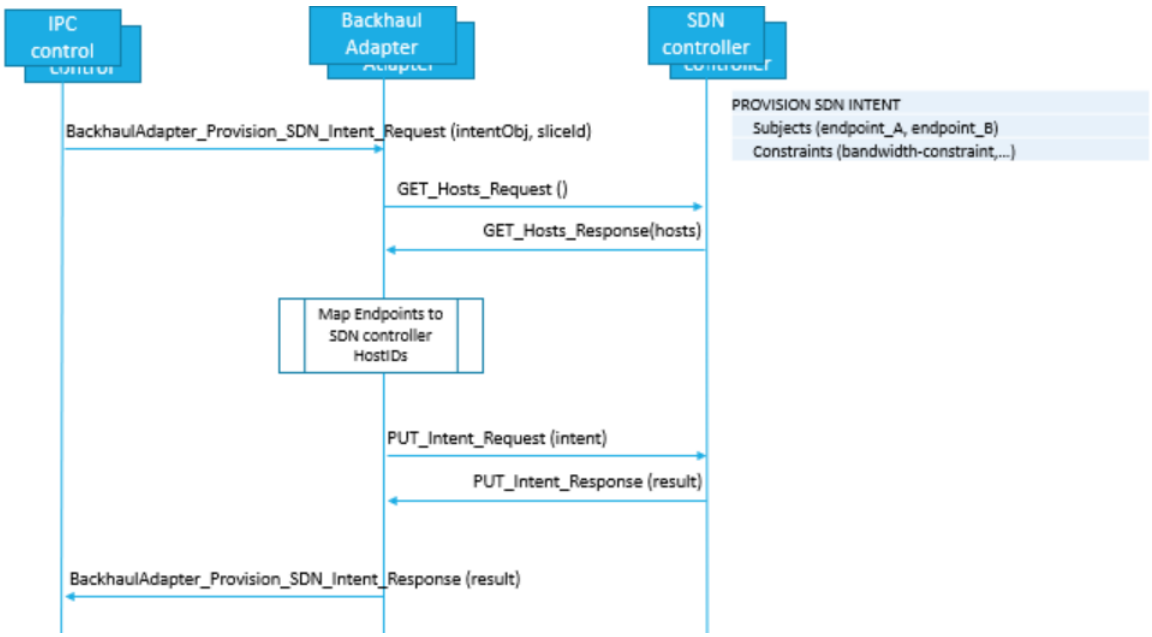

Figure 4.39 Provision SDN Intent flow

4.2.2.3.2 Update SDN Intent

- IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
- IPC_CP calls Backhaul Adapter API to Update an SDN Intent for Slice Id.
- BKH_ADAPTER gets list of available hosts from Backhaul SDN controller.
- BKH_ADAPTER maps the endpoints received from IPC_CP into corresponding Backhaul SDN controller Host IDs.
- BKH_ADAPTER gets list of available intents from Backhaul SDN controller.
- BKH_ADAPTER identifies the corresponding intent to be updated.
- BKH_ADAPTER calls Backhaul SDN Controller API to delete the existing Intent a create it again with updated data.
- Operation result is returned to IPC_CP.

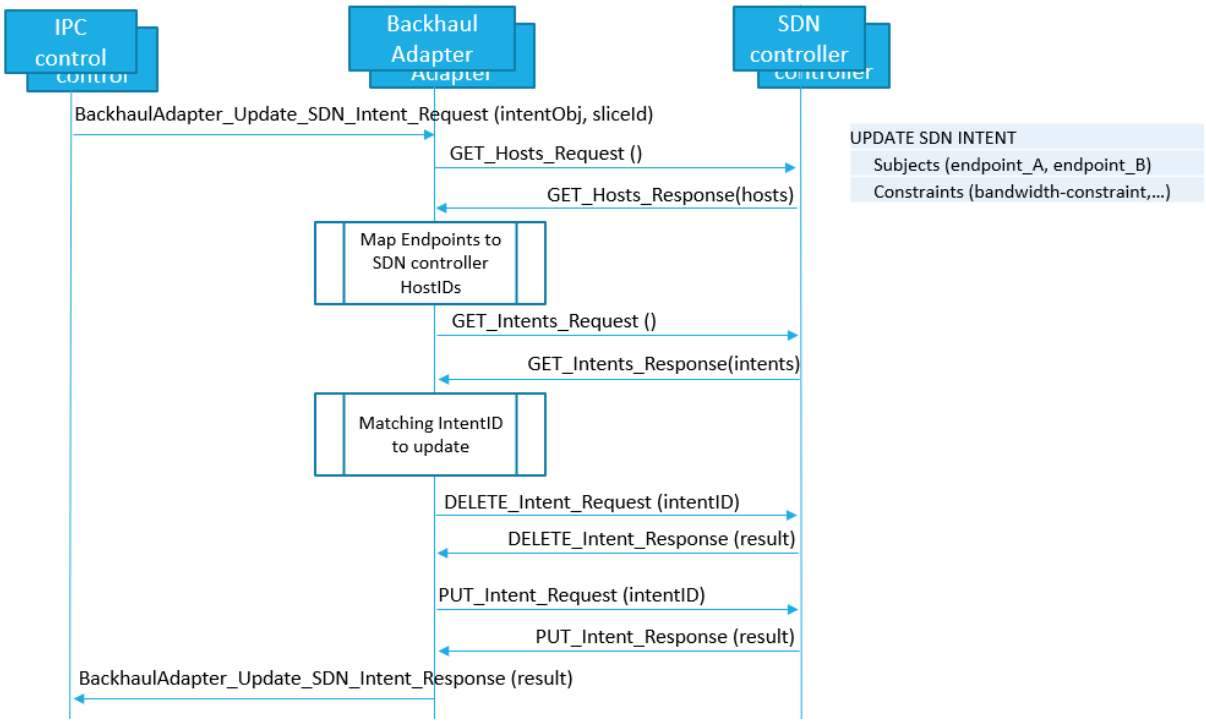

Figure 4.40 Update SDN Intent flow

4.2.2.3.3 Remove SDN Intent

- IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
- IPC_CP calls Backhaul Adapter API to delete SDN Intent for Slice Id.
- BKH_ADAPTER gets list of available hosts from Backhaul SDN controller.
- BKH_ADAPTER maps the endpoints received from IPC_CP into corresponding Backhaul SDN controller Host IDs.
- BKH_ADAPTER gets list of available intents from Backhaul SDN controller.
- BKH_ADAPTER identifies the corresponding intent to be removed.
- BKH_ADAPTER calls Backhaul SDN Controller API for Intent removal
- Operation result is returned to IPC_CP.

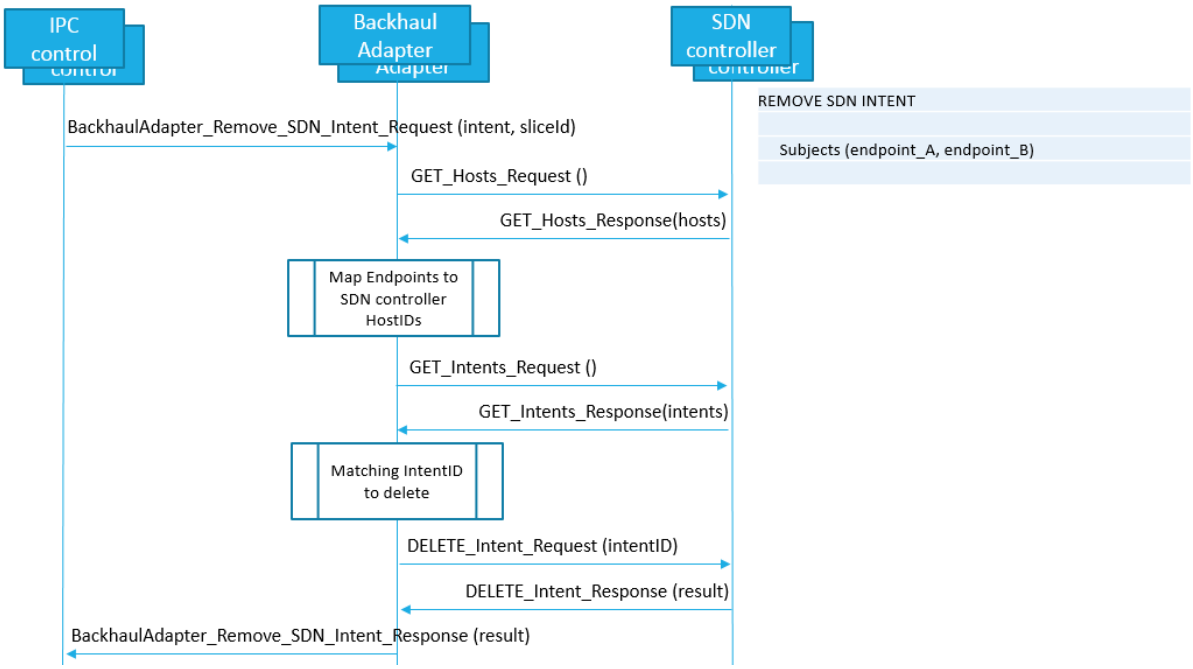

Figure 4.41 Remove SDN Intent flow

4.2.2.3.4 SDN Topology Exposure

- IPC_CP retrieves Backhaul Adapter address, via CPSR function: inputs are NF_Type="BKH_ADAPTER" and slice Id.
- IPC_CP calls Backhaul Adapter API to get the inter-PoP network topology.
- BKH_ADAPTER gets the list of Intents from Backhaul SDN Controller.
- BKH_ADAPTER selects all returned Host IDs having the VLAN ID attribute matching the required Slice Id.
- For selected Host IDs, the BKH_ADAPTER gets the corresponding IP addresses (endpoints) from Backhaul SDN Controller.
- BKH_ADAPTER returns to IPC_CP the obtained list of intents for the required slice Id.

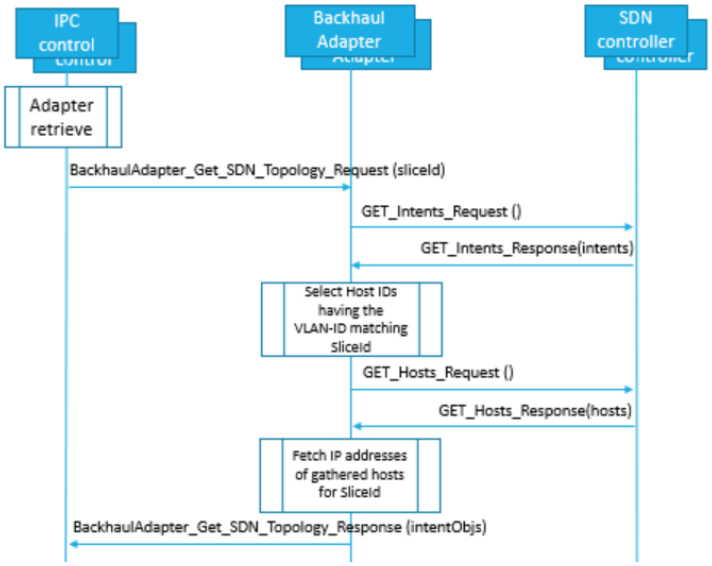

Figure 4.42 Get SDN Topology flow

5 Interfaces

Note that IPv6 is supported as described by the following Control Plane Services and CPSR interfaces.

5.1 CP-CPSR-S

The following sub-chapters describe the interfaces of "Control Plane Service Register" according the specifications stored under gitlab at the following link:

https://gitlab.com/slicenet/wp4/blob/develop/CPSR/Interface/cpsr.yaml

5.1.1 CPSR main data structure

5.1.1.1 Resources and methods overview

Table 5.1 CPSR Resources and methods overview

CPS Management			
Resource name	**Resource URI**	**HTTP method or custom operation**	**Description**
cps-instances (Store)	{apiRoot}/slicenet/ctrlplane/cpsr_cps/v1/cps-instances	GET	Read a collection of CPS Instances.
cps-instance (Document)	{apiRoot}/slicenet/ctrlplane/cpsr_cps/v1/cps-instances/{cpsInstanceID}	GET	Read the profile of a given CPS Instance.
		PUT	Register in CPSR a new CPS Instance, or replace the profile of an existing CPS Instance, by providing an CPS profile.
		PATCH	Modify the CPS profile of an existing CPS Instance.
		DELETE	Deregister from CPSR a given CPS Instance.
Subscriptions (Collection)	{apiRoot}/slicenet/ctrlplane/cpsr_cps/v1/subscriptions	POST	Creates a new subscription in CPSR to newly registered CPS Instances.
Subscription (Document)	{apiRoot}/slicenet/ctrlplane/cpsr-sm/v1/subscriptions/{subscriptionID}	DELETE	Deletes an existing subscription from CPSR.
cps-subscriptions (Collection)	{apiRoot}/slicenet/ctrlplane/cpsr_cps/v1/cps-instance/{cpsInstanceID}/cps-subscriptions	POST	Creates a new subscription in CPSR to changes of the profile of a given CPS Instance.
cps-subscription (Document)	{apiRoot}/slicenet/ctrlplane/cpsr_cps/v1/cps-instance/{cpsInstanceID}/cps-subscriptions/{cpsSubscriptionID}	DELETE	Deletes an existing subscription from CPSR.
Notification	{callbackUri}	POST	Notify about newly

Callback			created CPS Instances, or about changes of the profile of a given CPS Instance.

Table 5.2 CPS Discovery

CPS Discovery			
Resource name	**Resource URI**	**HTTP method or custom operation**	**Description**
CPS-instances (Store)	{apiRoot}/slicenet/ctrlplane/cpsr-disc/v1/cps-instances	GET	Retrieve a collection of CPS Instances according to certain filter criteria.

5.1.1.2 CPRS Main Data Type

CPS Profile

Table 5.3 CPS profile

CPSProfile				
Attribute name	**Data type**	**P**	**Cardinality**	**Description**
cpsInstanceId	string	M	1	Unique identity of the CPS Instance (UUID format).
cpsType	CPSType	M	1	Type of Control Plane Function (See table CPSType for details)
cpsStatus	CPSStatus	M	1	Status of the CPS Instance (REGISTERED, SUSPENDED)
slicenetId	String	O	0..1	slicenetId (UUID format)
fqdn	string	C	0..1	FQDN of the CPS where the service is hosted (see NOTE 1)
ipv4Addresses	array(String)	C	0..N	IPv4 address(es) of the CP Function (NOTE 1)
ipv6Addresses	array(String)	C	0..N	IPv6 address(es) of the CP Function (NOTE 1)
ipv6Prefixes	array(String)	C	0..N	IPv6 prefix of the CP Function (NOTE 1)
capacity	integer	O	0..1	Static capacity information, expressed as a weight relative to other CPS instances of the same type.
load	integer	O	0..1	Dynamic load information, ranged from 0 to 100, indicates the current load percentage of the CPS.
cpsServices	array(CPSService)	O	0..N	List of CPS Service Instances
Note 1: At least one of the addressing parameters (fqdn, ipv4address, ipv6adress and ipv6Prefix) shall be included in the CPS Profile. **Note 2:** M=Mandatory, C=Constraint; O=Optional;				

Table 5.4 CPS Types

CPSType	
Enumeration value	**Description**
BKH_ADAPTER	Backhaul Adapter
COR_ADAPTER	Core Adapter
MEC_ADAPTER	Mobile Multi Access Edge Computing Adapter
RAN_ADAPTER	Radio Access Network Adapter
WAN_ADAPTER	Wide Area Network Adapter
DPP_ADAPTER	Data Plane Programmability Adapter
QOS_CP	Control Plane Service: Quality of Service
ICP_CP	Control Plane Service: Interpop Connection
QOE_CP	Control Plane Service: Quality of Experience
PP_CP	Control Plane Service: Plug and Play
CNF_CP	Control Plane Service: Network Function Configuration
ID_CP	Control Plane Service: Interdomain

CPS Service

Table 5.5 CPS Service

CPSService				
Attribute name	**Data type**	**P**	**Cardinality**	**Description**
serviceInstanceId	string	M	1	Unique ID of the service instance within a given CPS Instance (UUID format)
serviceName	string	M	1	Name of the service instance (e.g. interface name "qos-local", "qos-public“)
version	string	M	1	Version of the service instance (e.g. "v1")
schema	string	M	1	Protocol schema (e.g. "http", "https")
slicenetId	String	O	0..1	slicenetId (UUID format)
fqdn	Fqdn	O	0..1	FQDN of the NF where the service is hosted (see NOTE 1)
ipEndPoints	array(IpEndPoint)	O	0..N	IP address(es) and port information of the Network Function (including IPv4 and/or IPv6 address) where the service is listening for incoming service requests (see NOTE 1) See details on table IPEndPoint
apiPrefix	string	O	0..1	Optional path segment(s) used to construct the {apiRoot} variable of the different API URIs (will do a table for that. e.g. ctrlplane/cpsr
defaultNotificationSubscriptions	array(DefaultNotificationSubscription)	O	0..N	Notification endpoints for different notification types.
allowedCPSTypes	array(CPSType)	O	0..N	Type of the CPSs allowed to access the service instance (all allowed if not present)
allowedSlices	array(string)	O	0..N	Allowed slices to access the service

				instance (all allowed if not present)
capacity	integer	O	0..1	Static capacity information, expressed as a weight relative to other services of the same type
load	integer	O	0..1	Dynamic load information, ranged from 0 to 100, indicates the current load percentage of the CPS.
NOTE 1: If the fqdn and ipEndpoint attributes are not present, the FQDN and IP address related attributes from the CPSProfile shall be used to construct the API URIs of this service.				

Table 5.6 CPS Service Ip End Point

IpEndPoint				
Attribute name	**Data type**	**P**	**Cardinality**	**Description**
ipv4Address	string	C	0..1	IPv4 address (NOTE 1)
ipv6Address	string	C	0..1	IPv6 address (NOTE 1)
ipv6Prefix	string	C	0..1	IPv6 prefix (NOTE 1)
transport	string	O	0..1	Transport protocol
port	integer	O	0..1	Port number
NOTE 1: At most one occurrence of either ipv4Address, ipv6Address or ipv6Prefix shall be included in this data structure.				

5.1.1.3 Registration Models

CPSR is designed to support several registration models that CPSs can choose. It is up to CPS to select the appropriate registration model to use depending on the chosen deployment model. The heart-beat procedure is impacted by the choice as it is tied to "profile" data structures. The three main models are reported below; a mixture of them is also possible.

Registration model A: Many services instances over one single server

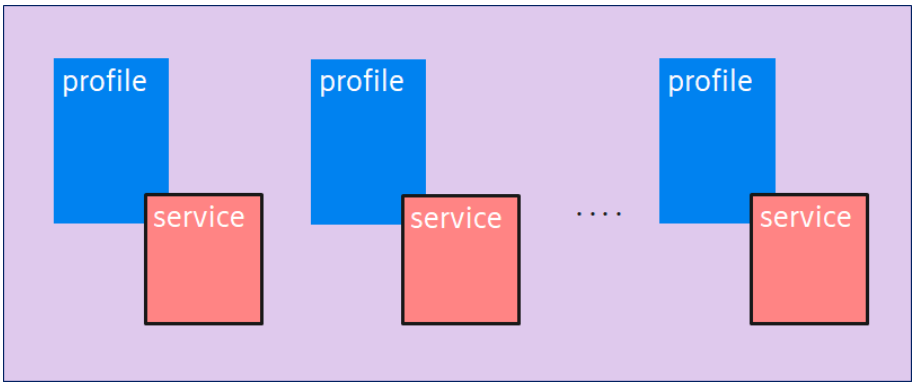

Figure 5.1 Registration model A: Many services instances over one single server

This model can be selected when each CPS instance (the couple of profile/service per slice) is stateful. In this model each service instance needs to maintain its heart-beat process up.

Registration model B: Many services instances over one single server

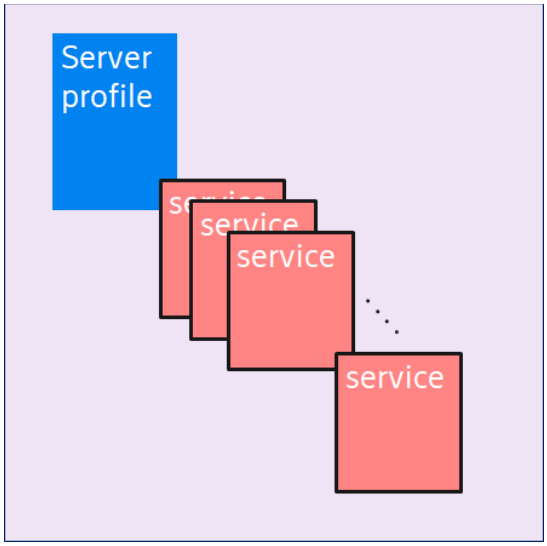

Figure 5.2 Registration model B: Many service instances over one single server

This model can be used when there are limited resources from the infrastructure, each service is stateless. In this model it is enough to have one single heart-beat process up, supporting all services.

Registration model C: One service over a single server

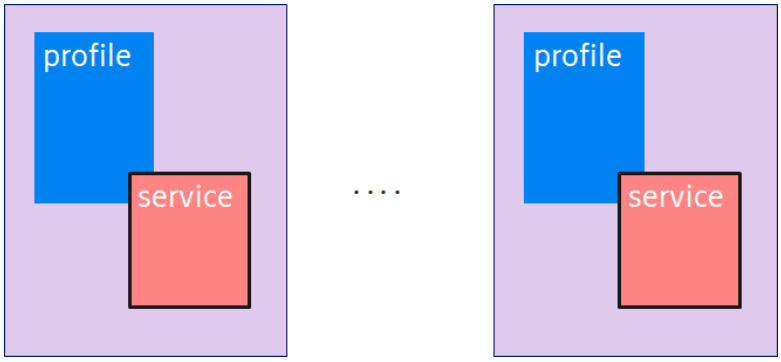

Figure 5.3 Registration model C: One service instance over a single server

This model is suggested when there are special security and isolation needs but it requires large amount of resources from infrastructure.

5.2 CP-CONF-S

The NF Config CPS is exposing a north bound interface as a REST endpoint (defined in https://gitlab.com/slicenet/information-model/blob/master/api/function.json).
It supports addition, update and removal of configuration regarding the supported operations with each operation identified for a specific slice as part of the path of the URL at which the REST request is performed as explained in Table 5.7

Table 5.7 CP CONF-S supported operations

Endpoint	/function/{sliceid}/{operation}"
Request	POST/PUT/DELETE
Request Body	A JSON array with configuration items as presented in the next table

Consumes	application/json
Produces	—-
Response Body	—-
Return Codes	200, 404 if the operation does not exist

The body of the request is structured as presented in Table 5.8

(https://gitlab.com/slicenet/information-model/blob/master/api/model/Configuration.json):

Table 5.8 Body request structure

Field	Type	Details
name	String	The name of one of the parameters of the operation. It is the same with the key defined in the operation descriptor.
value	String	The current value to be applied for the parameter under the above name.
unit	String	An optional unit identifier for numeric parameters.

5.3 CP-QOS-S

The following sub-chapters describe the interfaces “Set Qos constraints” and “Set Priority” according the specifications stored under gitlab at the following links:

https://gitlab.com/slicenet/wp4/blob/develop/QOS/interfaces/qos.yaml

5.3.1 Set QoS constraints

The operation allows the indication of quantitative parameters to be enforced for adequate servicing of user sessions in the network segments.

The sliceId is a mandatory field; it represents the unique identifier of the Slice associated to the QoS instance.

segmentId is optional, if it is not defined the QoS parameters will be enforced on all active sessions (or on the specific one) for all network segments (i.e. access, Core, backhaul).

userEqId and epsBearerId are optional, if they are not defined, QoS enforces QoS parameters on all active User sessions for the specific sliceId.

QoS parameters are mandatory, they are defined as a JSON schema.

All input paramters are described in the tables below.

[PUT | POST] ../qos/v1/qos-instance/{sliceId}/set_qos_constraints/{parameters}

Example usage:

url: http://localhost:8080/slicenet/ctrlplane

URL parameters:

Table 5.9 Set QoS constraints, parameters

Field name	Type	Example	Mandatory	Description
sliceId	UUID	6b699c12-d154-49ed-8922- f578e108f818	YES	Unique ID of the slice associated to QOS.
segmentId	String	ACCESS	NO	Identify the type of the adapter,

				ACCESS or CORE adapter kind
userEqId	String	49150123456789	NO	dentify the User Equipment Identity as IMSI
epsBearerId	integer	5	NO	Identify the EPS Bearer Identity

JSON parameters:

Table 5.10 Set QoS constraints, JSON parameters

Field name	Type	Example	Mandatory	Description
qosDirName	String	UPL	YES	The setting directive for QoS parameters can be: UPL=Upload directive; DWL= Download directive
qosDirValue	String	50	YES	The value of setting directive
qosUnitScale	String	MB	YES	The measurement scale of the setting directive

Request example:

In the following an example is reported for the setting of QoS parameter UPL (UpLink) to 50 MB given the <sliceID> UUID value, segmentId (ACCESS), userEqId (IMSI) and epsBearerId (5).

```
curl -X PUT http://localhost:8082/slicenet/ctrlplane/qos/v1/qos-
instance/<sliceID>/set_qos_constraints?segmentId=ACCESS&userEqId=4915012345
6789&epsBearerId=5 -H "Content-Type: application/json" -
d'{"qosDirName":"UPL","qosDirValue":"50","qosUnitScale":"MB"}'
```

Success response: **200 OK**

Error response: **400 Bad Request**

5.3.2 Set Priority

The operation allows the settings of the priority value in the north bound interface of the QoS component that matches the definition of the north bound interface of the BKH_DPP_Adapter described in section 0.

sliceId is mandatory, it represents the unique identifier of the Slice associated to the QoS instance.

priorityValue is mandatory, it represents the priority value and can be set as an integer between 1 and 9.

QoS calls set_priority towards BKH_DPP_Adapter that will send back the result of the request to QoS.

All input parameters are described in the table below.

[PUT] ../qos/v1/qos-instance/{sliceId}/set_priority/{parameters}

Example usage:

url: http://localhost:8080/slicenet/ctrlplane

URL parameters:

Table 5.11 Set Priority, parameters

Field name	Type	Example	Mandatory	Description
sliceId	UUID	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to QoS.
priorityValue	Integer	1,….9	YES	Identify the priority value Identity

Request Example

In the following an example is reported for the setting of the priority parameter (1) given the <sliceID> UUID value.

```
curl -X PUT http://localhost:8082/slicenet/ctrlplane/qos/v1/qos-
instance/<sliceID>/set_priority?priorityValue=1
```

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
```

5.4 CP-IPC-S

The following subchapters describe the interfaces “provision interpop connections” and “remove interpop connections” according to the specifications stored in gitlab at the following link:

https://gitlab.com/slicenet/wp4/blob/develop/IPC/interfaces/ipc.yaml

5.4.1 Provision InterPoP Connections

This operation interconnects two network functions in a slice between PoPs intra single domain, through an interPoP Connections descriptor representing the Backhaul SDN network between infrastructure pillars.

[PUT] ../ipc/v1/ipc-instance/{sliceId}/provision_interpop_connections

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/ipc/v1/ipc-instance/{sliceId}/provision_interpop_connections

URL parameters:

operationId: provisionInterpopConnections

The interPoP Connections object (Table 5.12) will contain the specific slice identifier and the list of interPoP paths to interconnect.

Table 5.12 Provision interPoP Connections parameters

Field name	Type	Example	Mandatory	Description
sliceId	string	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC.
InterPoP_Connections	array	[InterPoP_Path1, …, InterPoP_PathN]	YES	List of inter-PoP Paths.

Each interPoP_Path (Table 5.13) is an element of list in the InterPoP connections, it contains a set of parameters: the endPoints to interconnect and related constraints.

Table 5.13 InterPoP Path parameters

Field name	Type	Example	Mandatory	Description
endPoints	object	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to interconnect through the SDN backhaul network.
constraints	array	bandwidth=1GBPS	NO	Weights applied to a set of network resources, such as bandwidth, latency...

The endPoints object (Table 5.14) contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes to interconnect.

Table 5.14 endPoint Parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

The *constraints* paramater (Table 5.15) contains the information about the bandwidth, latency to use along the path interconnecting the endPoints.

Table 5.15 constraints Parameter

Field name	Type	Example	Mandatory	Description
bandwidth	float	1 GBPS	NO	Constraint that evaluates links based on available bandwidths.
latency	float	1 ms	NO	Constraint to keep under specified latency through a path

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```

5.4.2 Update InterPoP Connections

This operation allows dynamic modification of QoS constraints for the InterPoP Connections (e.g. bandwidth, latency).

[PUT] ../ipc/v1/ipc-instance/{sliceId}/update_interpop_connections

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/ipc/v1/ipc-instance/{sliceId}/update_interpop_connections

URL parameters:

operationId: updateInterpopConnections

The interPoP Connections object (Table 5.16) contains the specific slice identifier and the list of interPoP paths to update.

Table 5.16 Update interPoP Connections parameters

Field name	Type	Example	Mandatory	Description
sliceId	string	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC.
InterPoP_Connections	array	[InterPoP_Path1, …, InterPoP_PathN]	YES	List of inter-PoP Paths.

Each interPoP_Path (Table 5.17) is an element of list in the InterPoP connections, it contains a set of parameters: the interconnected endPoints and the updated constraints to use.

Table 5.17 InterPoP Path parameters

Field name	Type	Example	Mandatory	Description
endPoints	object	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to interconnect through the SDN backhaul network.
constraints	array	bandwidth=1GBPS	NO	Weights applied to a set of network resources, such as bandwidth, latency...

The endPoints object (Table 5.18) contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two interconnected nodes.

Table 5.18 endPoint Parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format

ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

The *constraints* parameter (Table 5.19) contains the information about the bandwidth, latency to use along the path interconnecting the endPoints.

Table 5.19 constraints Parameter

Field name	Type	Example	Mandatory	Description
bandwidth	float	2 GBPS	NO	Constraint that evaluates links based on available bandwidths.
latency	float	0.5 ms	NO	Constraint to keep under specified latency through a path

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```

5.4.3 Remove InterPoP Connections

This operation deletes the slice inter-PoP Connections.

[DELETE] ../ipc/v1/ipc-instance/{sliceId}/remove _interpop_connections

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/ipc/v1/ipc-instance/{sliceId}/remove _interpop_connections

URL parameters:

operationId: removeInterpopConnections

The interPoP Connections object (Table 5.20) contains the specific slide identifier and the list of interPoP paths to disconnect.

Table 5.20 remove InterPoP Connections Parameters

Field name	Type	Example	Mandatory	Description
sliceId	string	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC.
InterPoP_Connections	array	[InterPoP_Path1, …, InterPoP_PathN]	YES	List of inter-PoP Paths to disconnect

Each interPoP_Path (**Table 5.21**) to remove contains the endPoints to disconnect.

Table 5.21 InterPoP Path parameters

Field name	Type	Example	Mandatory	Description
endPoints	object	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to interconnected through the SDN backhaul network.

The endPoints object contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes to disconnect .

Table 5.22 endPoint Parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```


5.5 CP-RAN-A

This is the Service Based Interface exposed by the RAN Adapter, details are described in the SliceNet Deliverable D4.2 [5]

5.6 CP-MEC-A

This is the Service Based Interface exposed by the MEC Adapter, details are described in the SliceNet Deliverable D4.2 [5]

5.7 CP-CORE-A

This is the Service Based Interface exposed by the CORE Adapter, details are described in the SliceNet Deliverable D4.2 [5]

5.8 CP-WAN-A

This is the Service Based Interface exposed by the WAN Adapter, details are described in the SliceNet Deliverable D4.4 [6]

5.9 CP-BKHL-A

The following subchapters describe the interfaces “provision SDN intent”, “remove SDN intent” and “get SDN topology” according the specifications stored under gitlab at the following links:

https://gitlab.com/slicenet/wp4/blob/develop/BKH/Interface/bkh.yaml

5.9.1 Provision SDN Intent

This operation provides intent based data to connect two PoP endpoints on backhaul network.

[PUT] ../bkh/v1/bkh-instance/{sliceId}/provision_sdn_intent

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/bkh/v1/bkh-instance/{sliceId}/provision_sdn_intent

URL parameters:

The Intent object contains the specific slide identifier and the data related to the Intent to be provisioned.

Table 5.23 intent object parameters

Field name	Type	Example	Mandatory	Description
sliceid	String	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC
intentObj	object	intent	YES	info for intent to provide

An IntentObj contains the endPoints to interconnect and related constraints.

Table 5.24 IntentObj structure parameters

Field name	Type	Example	Mandatory	Description
endPoints	array	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to interconnect through the SDN backhaul network.

constraints	array	bandwidth=1 GBPS	NO	Weights applied to a set of network resources, such as bandwidth, latency.

The endPoints object contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes to interconnect via Intent.

Table 5.25 endPoints structure parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

The *constraints* parameter contains the information about the bandwidth, latency to use for the Intent.

Table 5.26 constraints Parameter

Field name	Type	Example	Mandatory	Description
bandwidth	float	1 GBPS	NO	Constraint that evaluates links based on available bandwidths.
latency	float	1 ms	NO	Constraint to keep under specified latency through a path

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```


5.9.2 Update SDN Intent

This operation allows to update the parameters for intent between two PoP endpoints on backhaul network.

[PUT] ../bkh/v1/bkh-instance/{sliceId}/update_sdn_intent

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/bkh/v1/bkh-instance/{sliceId}/update_sdn_intent

URL parameters:

The Intent object (Table 5.27) contains the specific slide identifier and the data related to the Intent to be updated.

Table 5.27 intent object parameters

Field name	Type	Example	Mandatory	Description
sliceid	String	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC
intentObj	object	intent	YES	info for intent to update

An IntentObj (Table 5.28) contains the interconnected endPoints and the constraints to update.

Table 5.28 IntentObj structure parameters

Field name	Type	Example	Mandatory	Description
endPoints	array	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to interconnect through the SDN backhaul network.
constraints	array	bandwidth=1 GBPS	NO	Weights applied to a set of network resources, such as bandwidth, latency.

The endPoints object (Table 5.29) contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes interconnected via Intent.

Table 5.29 endPoints structure parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

The *constraints* parameter (Table 5.30) contains the information about the bandwidth, latency to use for the Intent.

Table 5.30 constraints Parameter

Field name	Type	Example	Mandatory	Description
bandwidth	float	2 GBPS	NO	Constraint that evaluates links based on available bandwidths.
latency	float	0.5 ms	NO	Constraint to keep under specified latency through a path

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```

5.9.3 Remove SDN Intent

This operation deletes intent based connection between two PoP endpoints on backhaul network.

[DELETE] ../bkh/v1/bkh-instance/{sliceId}/remove_sdn_intent

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/bkh/v1/bkh-instance/{sliceId}/remove_sdn_intent

URL parameters:

The Intent object contains the specific slide identifier and the data related to the Intent to be removed.

Table 5.31 remove SDN intent parameters

Field name	Type	Example	Mandatory	Description
sliceid	String	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC
intentObj	object	intent	YES	Intent to remove

An IntentObj contains the endPoints to disconnect.

Table 5.32 IntentObj structure

Field name	Type	Example	Mandatory	Description
endPoints	array	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs to disconnect through the SDN backhaul network.

The endPoints object contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes relate to the Intent to be removed.

Table 5.33 endPoint structure parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

Success response:

```
HTTP/1.1
200 OK
```

Error response:

```
HTTP/1.1
400 BadRequest
403 Forbidden
404 Not Found
500 Internal Server error
```

5.9.4 Get SDN Topology

This operation exposes the backhaul network topology consisting of the endpoints of the PoPs on the infrastructure network segments which have been interconnected through the Backhaul SDN network controller.

[GET] ../bkh/v1/bkh-instance/{sliceId}/get_sdn_topology

Example usage:

url: http://localhost:8081/slicenet/ctrlplane/bkh/v1/bkh-instance/{sliceId}/get_sdn_topology

Result:

operationId: GetSdnTopology

For a specified Slice Id it is returned the list of installed intents on backhaul SDN controller.

Table 5.34 get SDN topology parameters

Field name	Type	Example	Mandatory	Description
sliceid	String	6b699c12-d154-49ed-8922-f578e108f818	YES	Unique ID of the slice associated to IPC
intentObjs	object	[Intent1, …, IntentN]	YES	List of intents for the specified sliceid

Each intent is an element of list in the intentObject, it contains a set of parameters: the interconnected endPoints and related constraints.

Table 5.35 Intent parameters

Field name	Type	Example	Mandatory	Description
endPoints	object	[endPointA,endPointB]	YES	A pair of routing devices on the PoPs interconnected through the SDN backhaul network.
constraints	array	bandwidth=1GBPS	NO	Weights applied to a set of network resources, such as bandwidth, latency...

The endPoints object contains the pair of IP addresses (either IPv4 or IPv6 format) and associated attributes of two nodes to interconnect.

Table 5.36 endPoint Parameters

Field name	Type	Example	Mandatory	Description
ipv4Address_nodeA/ ipv6Address_nodeA	string ipv4/ ipv6	"192.168.0.3"	YES	Source IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeA	string/ ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeA	string	“111”	YES	VLAN identifier of NodeA
ipv4Address_nodeB/ ipv6Address_nodeB	string ipv4/ ipv6	"192.168.12.4"	YES	Destination IP address either in IPv4 or IPv6 format
ipv6Prefix_nodeB	string ipv6	prefix/length in bits	NO	The leftmost fields of the IPv6 address contain the prefix, which is used for routing IPv6 packets.
vlan_nodeB	string	“222”	YES	VLAN identifier of NodeB

The constraints parameter contains the information about the bandwidth, latency used for the Intent.

Table 5.37 constraints Parameter

Field name	Type	Example	Mandatory	Description
bandwidth	float	1 GBPS	NO	Constraint that evaluates links based on available bandwidths.
latency	float	1 ms	NO	Constraint to keep under specified latency through a path

5.10 CP-BKHL-DPP-A

The following subchapters describe the interfaces of the control plane backhaul data plane programmability adapter according to the specifications stored under gitlab in the following link:

cp-bkh-dpp-a gitlab interface design

This is the list of Operations offered by Backhaul DPP Adapter to apply specific actions to the flow/flows:

- [PUT | DELETE] priority(parameters)
- [PUT | DELETE] min_bw(parameters)
- [PUT | DELETE] max_bw(parameters)

Pre-requisites:

- BKH_DPP_ADAPTER configured, registered, up and running
- FCA_CONTROLLER configured, registered, up and running

5.10.1 Parameters

Offered operations described in previous section have to contain a set of parameters, which are internally processed by the backhaul DPP adapter in order to create an understandable intent, which is then provided to the FCA Controller. Those parameters are:

- SliceID → For having control over slicing operations.
- Traffic Definition. → Which kind of traffic represents the flow/flows.
- Target machine (Management IP address) → IP address where the action is enforced. (E.g “target_ip_address”:”192.168.1.1”)
- Target interface. → Interface name where the action is enforced. (E.g “target_interface_name”:”eth0”)
- Target interface direction → Action will apply to the ingress/egress traffic.
- Action value → Depend on the use case. (E.g “priority_value:3”)

Following table gives information about the optionality of parameters depending on the three different operations offered by the backhaul dpp adapter (set priority, set minimum bandwidth guaranteed and set a maximum of the allowed bandwidth).

Table 5.38 parameters

Opt	Parameter	PRIO	MIN_BW	MAX_BW
Required	sliceID	X	X	X
Required	traffic_definition	X	X	X
Required	target_ip_address (Management IP Address)	X	X	X
Required	target_interface_name	X	X	X
Required	target_interface_direction	X	X	X
Optional	priority_value	X		
Optional	max_bwd		(X)	X
Optional	min_bwd		X	

X = present
(X)* = if present to be ignored
(X) = optionally present

5.10.1.1 Traffic Definition

Since BKH_DPP_ADAPTER is a component acting on traffic data plane, a specific traffic definition must be provided to it when setting a new action (E.g set priority). As explained before, this list of resource parameters can represent more than one flow by using wildcard values, achieving thus, slicing data plane actions.

The following table describes all possible parameters:

Table 5.39 traffic definition parameters

Field name	Example	Compulsory	Description
sliceId	“87E7EB8F”	YES	SliceID value of this resource.
encapsulationID1	“00000445”	NO	First encapsulation ID in Hexadecimal value (Just needed if there is encapsulation)
encapsulationID2	null	NO	Second encapsulation ID in Hexadecimal value (Just needed if there is encapsulation)
encapsulationType1	“gtp”	NO	First encapsulation protocol name (Just needed if there is encapsulation)
encapsulationType2	null	NO	Second encapsulation protocol name (Just needed if there is encapsulation)
encapsulationLayer	1	YES	Number of the encapsulation layer which this flow is representing.
srcIP	"192.168.0.3"	NO	Source ip address
dstIP	"192.168.12.4"	NO	Destination ip address
l4Proto	“17”	NO	Protocol number at Transmission layer
srcPort	“5004”	NO	Source port
dstPort	“5004”	NO	Destination port

5.10.1.2 Parameters example

This section provides an example of a JSON schema which represents a real set of parameters received in the northbound interface of the backhaul dpp adapter when a CP Service reaches the set priority function.

```
{
  "sliceID":"87E7EB8F",

  "traffic_definition":{
        "sliceID":"87E7EB8F",
        "encapsulationLayer":1,
        "encapsulationType": "gtp",
        "encapsulationID": "00000445"
  },

  "target_ip_address":"192.168.1.1",
  "target_interface_name":"eth0",
  "target_interface_direction":"egress",

  "priority_value":6
}
```

Therefore, all egressing traffic flowing through the interface eth0 of the specified ip address machine will change its priority to 6 if and only if is a GTP network traffic containing the specified encapsulation identifier.

5.10.2 Backhaul DPP Adapter Set/Remove Priority

[POST | DELETE] ../dpp/v1/dpp-instance/priority/{parameters}

Figure 5.4 set/remove priority

The priority is defined as a number from 0..7 that indicates how much priority will the slice have. The higher the number is, the higher priority will be applied to the traffic of that slice. The priority has impact in delay, probability of packet loss (reliability) and jitter.

Required parameters when inserting a priority over a specific network traffic is defined in Section 5.10.1 On the other hand, for deleting an existing rule which is already applying a priority action just the sliceID parameter will be needed.

5.10.3 Backhaul DPP Adapter Set/Remove Min BW

[POST | DELETE] ../dpp/v1/dpp-instance/min_bw/{parameters}

Figure 5.5 set/remove minimum bandwidth guaranteed

This method will set the minimum guaranteed bandwidth provided to a given network slice. All the traffic within that slice will share that bandwidth and compete against it. However, all the traffic of other slices will not interfere with this warrantied bandwidth. The bandwidth is indicated as a parameter in bits per second.

Required parameters when inserting a minimum bandwidth guaranteed over a specific network traffic is defined in section 5.10.1. On the other hand, for deleting an existing rule which is already applying such action just the sliceID parameter will be needed.

5.10.4 Backhaul DPP Adapter Set/Remove Max BW

[POST | DELETE] ../dpp/v1/dpp-instance/max_bw/{parameters}

Figure 5.6 set/remove maximum bandwidth allowed

This method will set the max bandwidth provided to a given network flow/flows. All the traffic within that slice will never pass this limitation. The bandwidth is indicated as parameter in bits per second.

Required parameters when inserting a maximum bandwidth limit over a specific network traffic is defined in section 5.10.1. On the other hand, for deleting an existing rule which is already applying such action just the sliceID parameter will be needed.

6 Prototype

Figure 6.1 shows the reference architecture used to prototype different components within this deliverable; detailed realization description is reported through next sub sections.

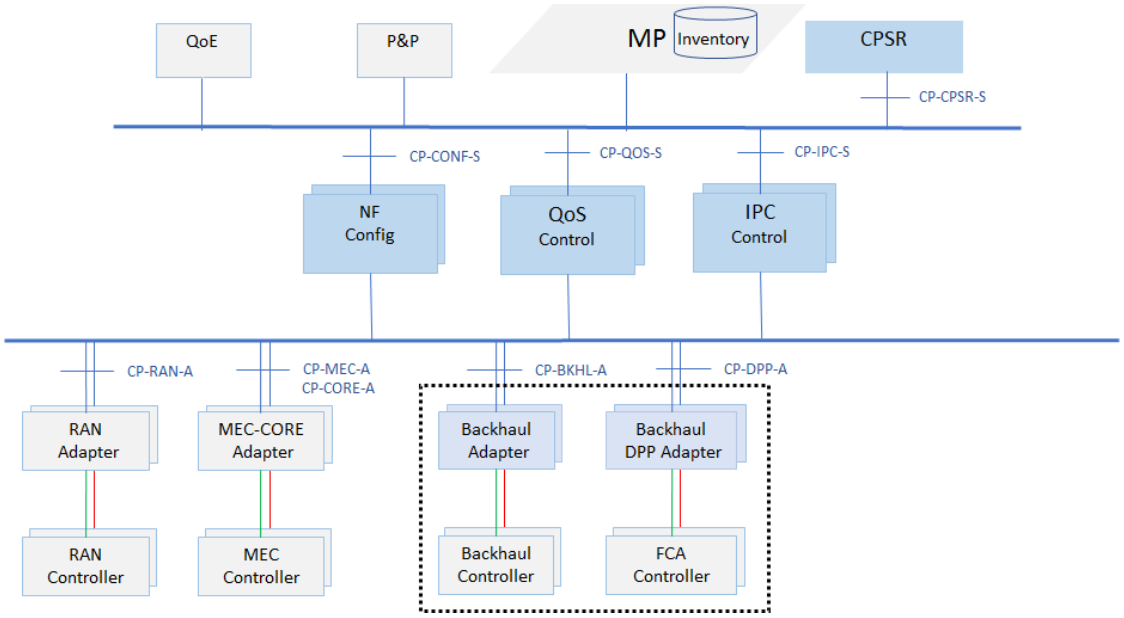

Figure 6.1 Prototype reference architecture

Protocols used for the SBI are: HTTP for application layer, TCP for transport, JSON for serialization, RESTful framework for the API style, OpenAPI for Interface Description Language (IDL).

Java has been used as major implementation language.

Swagger is the tool used to generate the models and classes described by OpenAPI language, it is able to generate both client and server sketch implementations that it is described by API.

Each component contains inside at least one server implementation and one or more clients implementation.

The component could be deployed as war object or as a docker object. it is possible also to deploy it inside a single Virtual Machine if needed.

When the docker is used, it must contain inside a Web server that is able to run one or more war objects. The basic web server used is Jetty but we have done some tests with WildFly server that allows to support scalability and clustering if needed.

All the components except CPSR are stateless without database dependency.

Instead, CPSR has need a database. To be independent from specific database technology implementation Hibernate framework has been used.

The binary software has been stored in the SliceNet GIT repository.

6.1 CPSR

CPSR realization is based on Java language. Its server interface described in terms of openAPI 3.0 language is automatic generated by swagger tools (https://swagger.io/) as Java code. After the generation the major methods are completed with proper additional Java code.

The server is using a database to store its data. Abstraction from database technology is operated via Hibernate framework (http://hibernate.org) that can work as relational database RDBMS (working with SQL language) or as NoSQL technology[1]. In addition, it can create independence from specific DB vendors. Java HSQLDB (http://hsqldb.org) configured as in memory database, or Infinispan (http://infinispan.org) can be used as DB depending on the wanted CPSR Architecture deployment: "no High Availability" or "High Availability" mode as further described in this section.

Classes generated by Swagger are mapped to database tables manually. Access to database is done in Java via JPA.

The WAR deploy method is used also to be independent from web server technology.

CPSR prototype implementation does not include any security rule, so no OAuth 2.0 is needed.

CPSR component is developed from scratch. It is distributed as container and it is available in gitlab in binary form.

6.1.1 CPSR, no High Availability Architecture

Jetty (https://www.eclipse.org/jetty) is used in this case and only HTTP is supported. HTTP/2 is not supported as and additional package (ALPN) would be needed.

Figure 6.2 CPSR Prototype reference architecture with no High Availabilityshows the overall components used to obtain a CPSR running instance

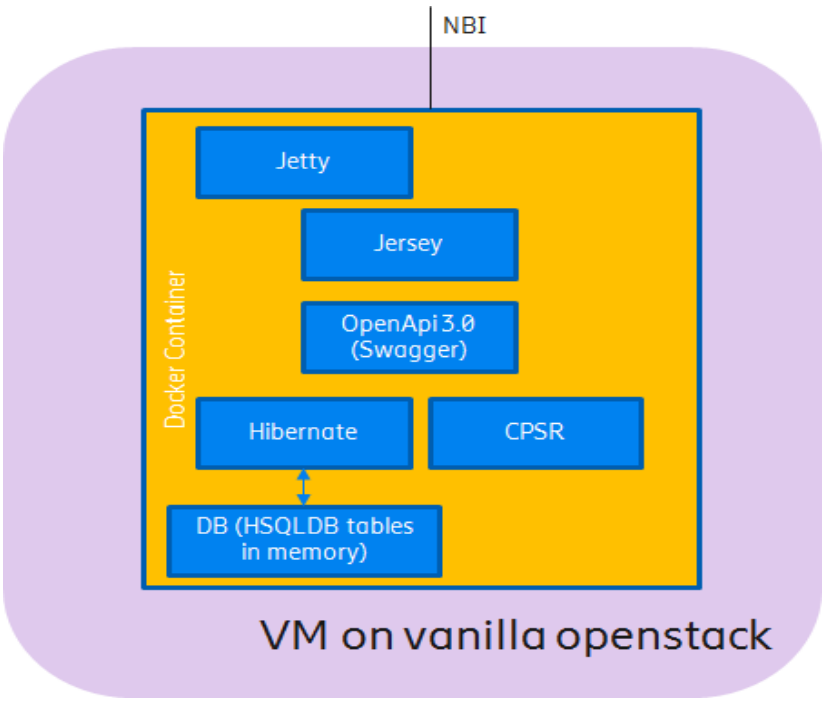

Figure 6.2 CPSR Prototype reference architecture with no High Availability

[1] Hibernate provide two main technology: sql (Hibernate ORM) and nosql (Hibernate OGM)

6.1.2 CPSR, High Availability Architecture

Wildfly (http://wildfly.org) is used in this case so to realize the CPSR High Availability Architecture as illustrated in section 4.1.1.4.
WildFly High Availability services supports two features which ensure high availability of critical Java applications like CPSR:

- **fail-over**: allows a client interacting with a Java application to have uninterrupted access to that application, even in the presence of node failures.
  If the application makes use of WildFly fail-over services, a client interacting with an instance of that application will not be interrupted even when the node on which that instance executes crashes. Behind the scenes, WildFly makes sure that all of the user data that the application make use of (HTTP session data, EJB SFSB sessions, EJB entities and SSO credentials) are available at other nodes in the cluster, so that when a failure occurs and the client is redirected to that new node for continuation of processing (i.e. the client "fails over" to the new node), the user's data is available and processing can continue.
- **load balancing**: allows a client to have timely responses from the application, even in the presence of high-volumes of requests, Load balancing enables the application to respond to client requests in a timely fashion, even when subjected to a high-volume of requests. Using a load balancer as a front-end, each incoming HTTP request can be directed to one node in the cluster for processing. In this way, the cluster acts as a pool of processing nodes and the load is "balanced" over the pool, achieving scalability and, consequently, availability. Requests involving session-oriented servlets are directed to the same application instance in the pool for efficiency of processing (sticky sessions). Using mod cluster has the advantage that changes in cluster topology (scaling the pool up or down, servers crashing) are communicated back to the load balancer and used to update in real time the load balancing activity and avoid requests being directed to application instances which are no longer available.

Wildfly also supports HTTP/2 natively.

Figure Figure 6.3 shows the components used to obtain CPSR running instances in “High Availability mode” leveraging Wildfly open source.
Infinispan (http://infinispan.org) or other clustering capable DB like Mongodb (https://www.mongodb.com) can be used.

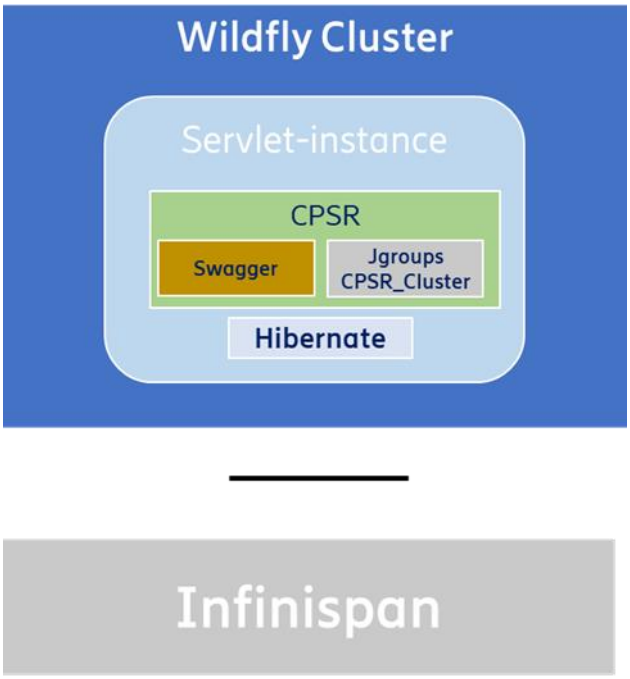

Figure 6.3 CPSR Prototype reference architecture with High Availability

6.2 QoS Control Service

QoS source code is mantained in GitLab at following path https://gitlab.com/slicenet/wp4.

QoS CP service application fits SBA (Service Based Architecture) and it is based on Container-based virtualization using Docker platform: it’s an open platform that enables to separate our applications from the infrastructure allowing to ship, test and deploy our QoS application quickly.

Following figure represents the adopted architecture by QoS Application.

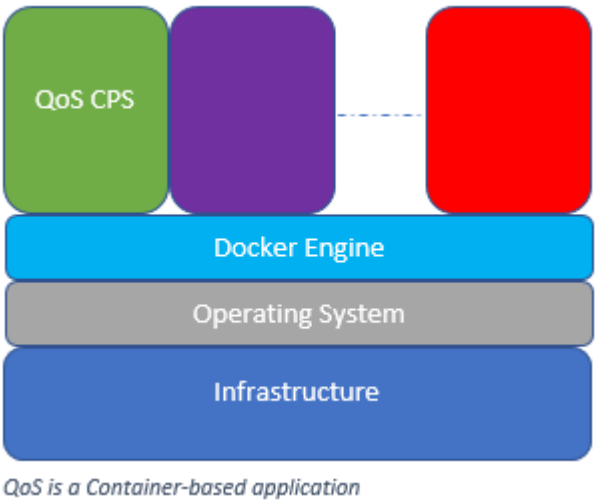

Figure 6.4 QoS prototype architecture

QoS CPS application docker image has been built using a base image with minimal Ubuntu OS and customized adding following applications (See next figure):

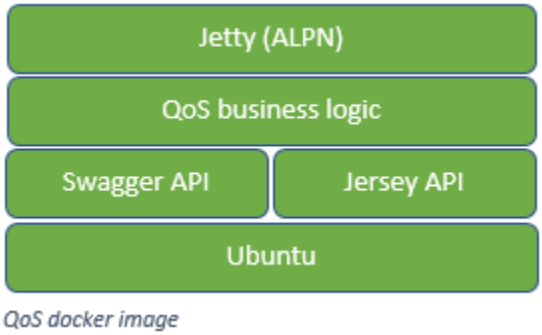

Figure 6.5 QoS prototype docker image

- Jetty Web Server with ALPN to support HTTP and HTTP/2 protocols
- Jersey API to support the development of RESTFUL Web Server and corresponding client in Java
- Swagger API (OpenAPI 3.0 Spec) to support the data exchange between client/Server application using JSON/YAML languages
- QoS business logic is a service Java based

6.2.1 QOS prototype and test environment description

The main purpose of the QOS module is to increase the abstraction level when delivering the Quality Of Service constraints to the SliceNet network over the network adapters placed in the RAN and in the Core segment network.

The program is developed in Java language; it handles JSON messages, over HTTP protocol and RESTful interfaces, that responds to a predefined algorithm. The test environment was tided-up running an instance of the CPSR module in a virtual machine (Oracle VirtualBox).

To simulate a real use case and to test the QOS functionalities, two different STUB servers, the RAN adapter and the CORE adapter, were also developed and put in service in the Eclispe IDE.

Requests to the QoS were coming from a RESTful web interface (postman style) via the Google Chrome browser.

In the current prototype QOS instantiation is performed manually, in the final product it will be performed by Management Plane layer that will trigger the Life Cycle Management of QOS according to the Slice instances managed in the system.

6.3 IPC

IPC source code is maintained in GitLab at following path https://gitlab.com/slicenet/wp4.

IPC CP service application fits SBA (Service Based Architecture) and it is based on Container-based virtualization using Docker platform: it's an open platform that enables to separate our applications from the infrastructure allowing to ship, test and deploy our IPC application quickly.

Figure 6.6 represents the adopted architecture by IPC Application.

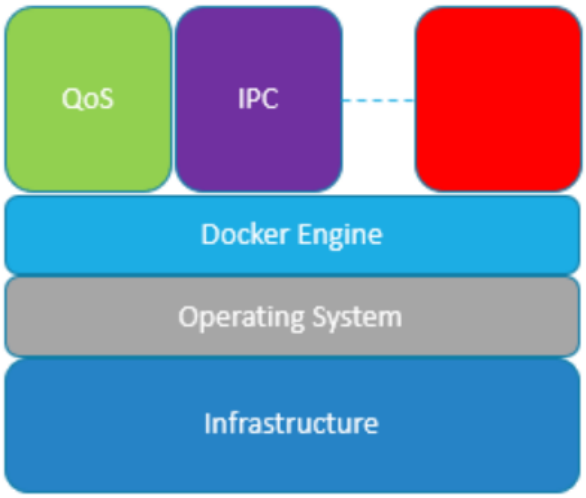

Figure 6.6 IPC prototype architecture

IPC CPS application docker image has been built using a base image with minimal Ubuntu OS and customized adding following applications (See **Figure 6.7**):

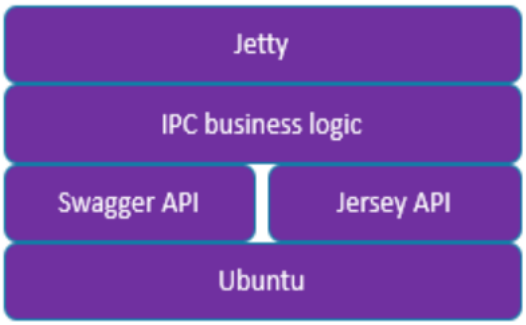

Figure 6.7 IPC prototype docker image

- Jetty Web Server to support HTTP protocol

- Jersey API to support the development of RESTFUL Web Server and corresponding client in Java
- Swagger API (OpenAPI 3.0 Spec) to support the data exchange between client/Server application using JSON/YAML languages
- IPC business logic is a service Java based

6.3.1 IPC prototype and test environment description

The program is developed in Java language; it handles JSON messages, over HTTP protocol and RESTful interfaces, that responds to a predefined algorithm. The test environment was tided-up running an instance of the CPSR module in a virtual machine (Oracle VirtualBox).

To simulate a real use case and to test the IPC functionalities a prototype of the Backhaul Adapter is also used and put in service together with IPC prototype. Requests to the IPC were coming from a RESTful web interface (postman style) via the Google Chrome browser.

In the current prototype IPC instantiation is performed manually, in the final product it will be performed by Management Plane layer that will trigger the Life Cycle Management of IPC according to the Slice instances managed in the system.

6.4 NF Config Control Service

The NF Config CPS has been developed as an extension of the Application Manager VNF/PNF App Configuration Service Class (96[39]) and particularly the part of the code for REST communications. However, it has been disintegrated from the initial Application Manager Service so that it can be deployed and managed under the overall approach for SliceNet CPS Management. The prototype (https://gitlab.com/slicenet/wp4/tree/develop/NF-CONFIG) is based on an OSGi Karaf Blueprint archetype and provides support for REST services using Apache CXF and Jetty Engine.

In the current form the module, when launched in Karaf, registers with CPSR but it runs in a stateless mode and has to be configured every time it starts as foreseen in paragraph 3.4.2. Configuration is done by posting operation descriptors as JSON content to its management endpoint (e.g. https://gitlab.com/slicenet/wp4/blob/develop/NF-CONFIG/descriptor.json) structured according to a specified schema (https://gitlab.com/slicenet/wp4/blob/develop/NF-CONFIG/descriptorSchema.json). The descriptor is analysed and for every operation is found, a REST endpoint is created to make available the specific configuration option per slice. The part of the communication section that defines the workflow of the operation is used within each object launched per operation to apply the required steps. Text enclosed in curly brackets either in URLs or in request bodies indicates the use of a variable defined in one of the following ways:

- A key:value parameter in the operation request body
- A Config CPS instance specific parameter (e.g. Slice ID)
- An intermediate value that is defined in the context of the execution of the workflow
    - JSON-Path (https://github.com/json-path/JsonPath) is used to collect the value from the response collected in one of the steps in the workflow process and then it is assigned to one of the variables of the operation processing space

The NF Config CPS will be adapted according to integration requirements identified in the following period when management and orchestration modules will be provided by the corresponding tasks. It is also expected that the descriptor schema may need to be adapted according to the evolution of the overall SliceNet information model.

6.5 Adapters

6.5.1 Backhaul Adapter

BKH ADAPTER source code is maintained in GitLab at following link: https://gitlab.com/slicenet/wp4.

BKH ADAPTER service application fits SBA (Service Based Architecture) and it is based on Container-based virtualization using Docker platform: it's an open platform that enables to separate our applications from the infrastructure allowing to ship, test and deploy our Adapter application quickly.

Figure 6.8 below represents the adopted architecture by BKH ADAPTER Application.

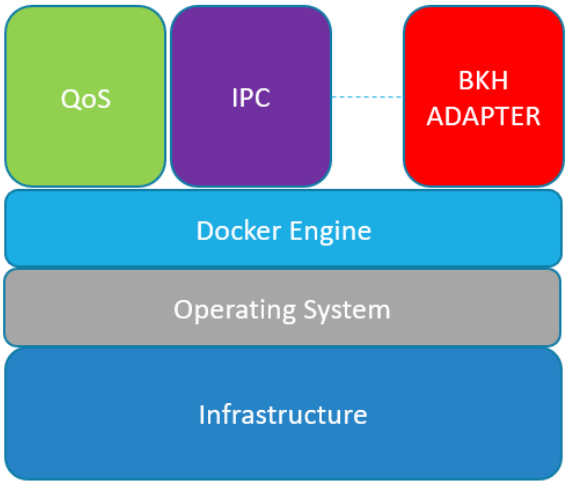

Figure 6.8 BKH ADTAPTER prototype architecture

BKH ADAPTER application docker image has been built using a base image with minimal Ubuntu OS and customized adding following applications (See Figure 6.9):

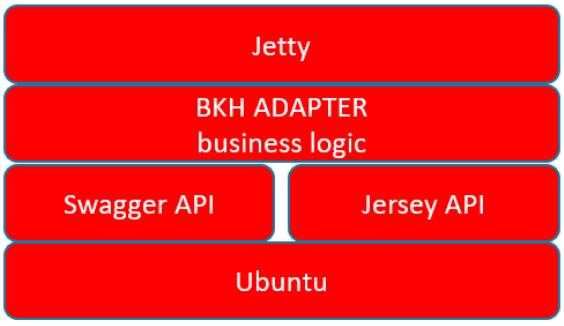

Figure 6.9 BKH ADAPTER prototype docker image

- Jetty Web Server to support HTTP protocol
- Jersey API to support the development of RESTFUL Web Server and corresponding client in Java
- Swagger API (OpenAPI 3.0 Spec) to support the data exchange between client/Server application using JSON/YAML languages
- BKH ADAPTER business logic is a service Java based

6.5.1.1 BKH ADAPTER prototype and test environment description

The program is developed in Java language; it handles JSON messages, over HTTP protocol and RESTful interfaces, that responds to a predefined algorithm. The test environment was tided-up running an instance of the CPSR module in a virtual machine (Oracle VirtualBox).

To simulate a real use case and to test the BKH ADAPTER functionalities a prototype of the IPC is also used and put in service together with BKH ADAPTER prototype.

Requests to the BKH ADAPTER were coming from a RESTful interface on northbound interface and were sent to ONOS SDN Controller ONOS (version 1.14.0) on the southbound interface.

In the current prototype BKH ADAPTER instantiation is performed manually.

To simulate the realistic virtual network has been used Mininet (**Mininet 2.2.2 on Ubuntu 14.04 LTS - 64 bit**) running **real kernel, switch and application code**, on a single machine (VM, cloud or native).

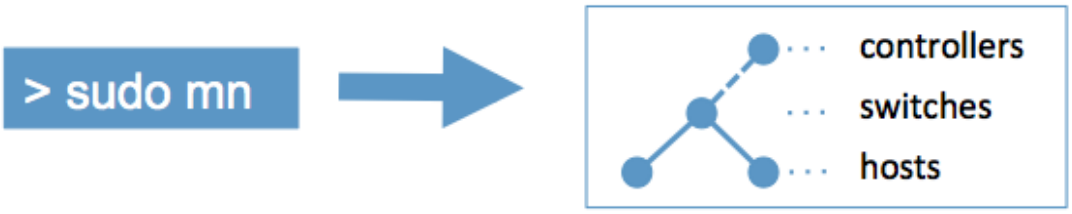

Figure 6.10 Mininet and Backhaul Adapter prototype

6.5.2 Backhaul DPP Adapter

Backhaul Data Plane Programmability Adapter (BKH_DPP_ADAPTER) component has been prototyped and implemented by using Python 3 coding language in its main source code. The last release of such component can be found in the SliceNet git repository and contains all instructions about how to configure it and test it. As described in section 4.2.1.1, the BKH_DPP_ADAPTER is mainly composed by a northbound interface for both, registering to the CPSR and receiving slice actions; a southbound interface for handling intent-base operations to the FCA Controller; a stateless engine module which is in charge of controlling service operations from the northbound interface to southbound interface by using internal SW modules; a mapping module which contains the logic for transform provisioned parameters from the northbound interface to an intent-based operations. Those intents are based in a (JavaScript Object Notation) JSON data-interchange format and provide to the FCA Controller all required information for executing an action in a specific location of the network. The following schema represents an example of the architecture of the intent-based object which is produced as an outcome of the BKH_DPP_ADAPTER in its southbound interface.

```
{
  "Resources": [
  {
      "resourceId": "87E7EB8F",
      "encapsulationType1": "gtp",
      "encapsulationID1": "00000445",
      "encapsulationLayer": 1,
      "packetStructure": "/ip:20",
      "completePacketStructure": "/mac:14/ip4:20/udp:8/gtp:8/"
    }
  ],
  "Intent": {
    "actionType": "INSERT",
```

```
    "actionName": "SET_PRIORITY"
  },
  "Params": [
    {
      "paramName": "target_interface_name",
      "paramValue": "enp0s25"
    },
    {
      "paramName": "target_ip_address",
      "paramValue": "192.168.1.1"
    }
    {
      "paramName": "target_interface_direction",
      "paramValue": "egress"
    }
  ]
}
```

6.6 Network Controllers

The following network controllers have been used for prototyping activities, but they are not in the scope of this deliverable,

6.6.1 ONOS

ONOS [14] has been used as Backhaul Controller for Backhaul Adapter prototyping as described in Section 6.5.1.

6.6.2 FCA Controller

El FCA Controller consumes Intent-based requests provided by the Backhaul DPP adapter with the purpose of selecting the specific network endpoint related to the concrete location of the network where the intent will be enforced. This location is the Node that is under the control of a Flow Control Agent (FCA) that is responsible to enforce such an intent into the control plane of such a machine. To do so, the FCA Controller will place the intents in a queue and process them by routing them to the proper machines. Notice that the receiver can be only one machine if the intent is associated to a unique point of the network or multiple machines if the request is associated to a global network policy across the whole administrative domain. Also, this component will provide reliability in the delivery of the message so that it will retransmit them to the destinations in case of connectivity problem in order to make sure there is a consistent state of the control plane. This controller also provides fault tolerance against failures of the Flow Control Agent by leaving the message in the queues in case the control agent are not ready to consume them and thus allowing a recovery of the state when they are ready again. This FCA Controller is a centralized logical entry point to the infrastructure control which is physically distributed across all the machines controlled.

It is noted that the previous version of FCA was developed in 5G PPP Phase 1 project SELFNET, as specified in SELFNET D3.4 [37] . The current version reported in SliceNet has been further developed and extended to have the following additional functions:

- SliceNet FCA is able to act as a standalone actuator (without the need to employ an external traffic control device such as TrustNode used in SELFNET);
- SliceNet FCA is able to enforce additional traffic control actions including Change Priority, Minimum Bandwidth Guaranteed, Maximum Bandwidth Allowed, in addition to Drop a Flow in SELFNET;
- Multiple SliceNet FCAs can be distributed to different locations in 5G networks and controlled by an FCA Controller.

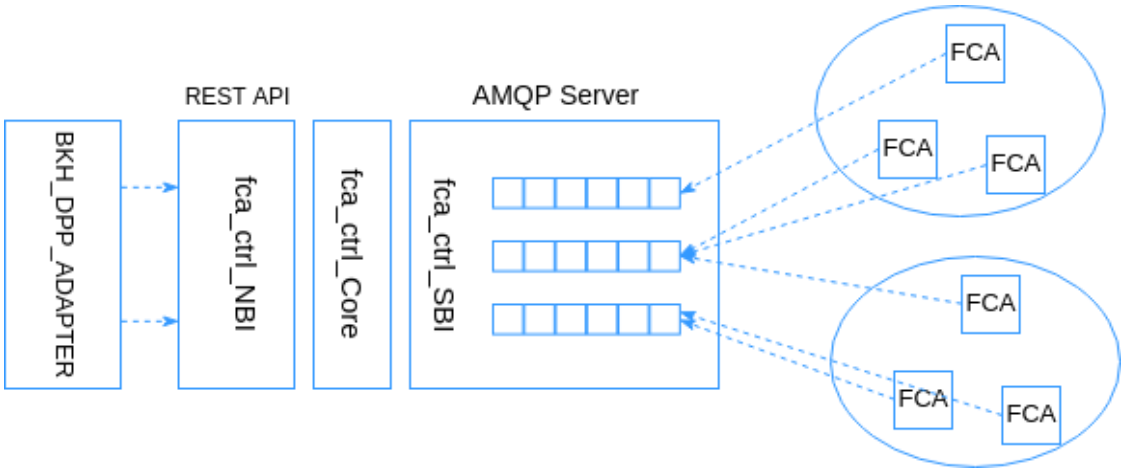

Figure 6.11 FCA Controller Architecture

- *fca_ctrl_NBI* handles the northbound API interface with other control plane service consumers for receiving intent-base massages which are describing the action to apply, the traffic which will be affected and the deployment location.
- fca_ctrl_Core controls the interworking between northbound interface and southbound interface handling those messages and processing them for being southbound compliant.
- fca_ctrl_SBI handles the southbound API interface which in this case, follows a different approach for data communication which is based on Advanced Message Queuing Protocol (AMQP) instead of the Representational State Transfer API used in upper layer components for requesting and transmitting data elements. Therefore, the FCA Controller will act as a provider/publisher of AMQP messages which will be sent to the Flow Control Agents (FCAs) deployed and ready for consuming these king of messages. It is important to highlight that just those FCAs interested and binded to the proper routing key will be able for consuming and therefore enforce the incoming actions.

7 Conclusions

This deliverable presents the design and prototype implementation of the SliceNet Control Plane Services and Backhaul Adapters for Single Domain in the context of a Service Based Architecture framework.

In addition, this delivery includes the realization of the Control Plane Service Register (CPSR) in order to enable the Service Based Architecture framework which is used to accommodate the required SliceNet Control Plane components. As the CPSR is a critical service for SliceNet, a related High Availability architecture is described coupled with a possible implementation leveraging on existing open sources.

One instance of each CPS is created to be serving one Slice instance; this one-to-one relationship is regarded as a key enabler for Slice performance and security isolation as well as for the overall system scalability when it comes to the number of Slices instances.

The components software prototypes have been also documented in this deliverable, in terms of reference open source frameworks and implementation choices.

All the delivered components have been implemented from scratch mostly as a containerized application.

Finally, the lifecycle management of the Control Plane Services instances, performed manually in the first stage with Kubernetes command line tools, is now fully automated through the CPS Life Cycle Manager delivered by “SliceNet Deliverable D6.3” [7]

As part of the next steps, the Control Plane Services and Backhaul Adapters components will be further integrated within the SliceNet platform in the context of WP8 activities. The interactions with SliceNet Control Plane Service consumers as well as with RAN, MEC and Core Infrastructure is one of main targets for integration. Moreover, the requirements from SmartGrid, SmartCity and eHealth use cases will be further investigated with the use case teams to identify additional functions for the delivered components required to fulfil the SliceNet verticals’ needs.

8 References

[1] SliceNet Deliverable D2.2, “Overall Architecture and Interfaces Definition”, SliceNet Consortium, January 2018

[2] SliceNet Deliverable D2.3, “Control Plane System Definition, APIs and Interfaces”, SliceNet Consortium, April 2018

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

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

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

[6] SliceNet Deliverable D4.4, “Multi-Domain Network Slicing Control and Negotiation”, SliceNet consortium, planned for February 2019.

[7] SliceNet Deliverable D6.3, “Management for the Plug & Play Control Plane”, SliceNet consortium, May 2019

[8] A. Mohammadkhan, K. K. Ramakrishnan, A. S. Rajan, and C. Maciocco, “CleanG: A Clean-Slate EPC Architecture and Control Plane Protocol for Next Generation Cellular Networks”, Proc. ACM Workshop on Cloud-Assisted Networking (CAN) 2016, Irvine, CA, USA, Dec. 2016.

[9] A. M. Nayak , P. Jha, and A. Karandikar, “A Centralized SDN Architecture for the 5G Cellular Network”, Jan. 2018. [Online].Available at https://www.ee.iitb.ac.in/~karandi/publications/preprint/akshatha_pranan_karandikar_ieee5gforum.pdf

[10] Y. Li, Z. Yuan, and C. Peng, “A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks”, Proc. 23rd Annual International Conference on Mobile Computing and Networking (Mobi Com’17), Snowbird, UT, USA, Oct. 2017.

[11] ONF TR-526, “Applying SDN Architecture to 5G Slicing”, Apr. 2016.

[12] Floodlight Project web site. Available at http://www.projectfloodlight.org/floodlight/.

[13] ODL OpenDayLight web site. Available at https://www.opendaylight.org/.

[14] ONOS web site. Available at https://onosproject.org/.

[15] RyU web site. Available at https://osrg.github.io/ryu/.

[16] Open Networking Foundation (ONF) web site. Available at https://www.opennetworking.org/.

[17] Network Configuration Protocol RFC. Available at https://tools.ietf.org/html/rfc6241.

[18] P. Marsch, I. Da Silva, O. Bulakci, M. Tesanovic, S. E. El Ayoubi, T. Rosowski, A. Kaloxylos, and M. Boldi, “5G Radio Access Network Architecture: Design Guidelines and Key Considerations,” IEEE Communications Magazine, vol. 54, no. 11, pp. 24–32, Nov. 2016.

[19] 3GPP TR38.801, “Technical Specification Group Radio Access Network; Study on New Radio Access Technology: Radio Access Architecture and Interfaces, Release 14”, Apr. 2017.

[20] 3GPP TR 38.804 Study on new radio access technology: Radio Interface Protocol Aspects (Release 14), Mar. 2017.

[21] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, “OpenRAN: A Software-defined Ran Architecture via Virtualization,” ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 549–550, Aug. 2013.

[22] I. F. Akyildiz, P. Wang, and S.-C. Lin, “SoftAir: A Software Defined Networking Architecture for 5G Wireless Systems,” Computer Networks, vol. 85, pp. 1–18, 2015.

[23] A. Gudipati, D. Perry, L. E. Li, and S. Katti, “SoftRAN: Software Defined Radio Access Network,” in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN ’13). ACM, 2013, pp. 25–30.

[24] T. Chen, H. Zhang, X. Chen, and O. Tirkkonen, "SoftMobile: Control Evolution for Future Heterogeneous Mobile Networks," IEEE Wireless Communications, vol. 21, no. 6, pp. 70–78, Dec. 2014.
[25] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: A Programmable Wireless Dataplane," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12). ACM, 2012, pp. 109–114.
[26] W. Wu, L. E. Li, A. Panda, and S. Shenker, "PRAN: Programmable Radio Access Networks," in Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets-XIII). ACM, 2014, pp. 6:1–6:7.
[27] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. P. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16). ACM, 2016, pp. 427–441.
[28] C.Y. Chang and N. Nikaein, "RAN Runtime Slicing System for Flexible and Dynamic Service Execution Environment," IEEE Access, vol. 6, pp. 34018–34042, 2018.
[29] V. G. Nguyen and Y. H. Kim, "Slicing the Next Mobile Packet Core Network," in 2014 11th International Symposium on Wireless Communications Systems (ISWCS), Aug. 2014, pp. 901–904.
[30] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a Service to Ease Mobile Core Network Deployment over Cloud," IEEE Network, vol. 29, no. 2, pp. 78–88, Mar. 2015.
[31] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A High Performance Packet Core for Next Generation Cellular Networks," in Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). ACM, 2017, pp. 348–361.
[32] TR 23.707 Architecture enhancements for dedicated core networks; Stage 2 (Release 13), 3GPP, Dec. 2014.
[33] TR 23.711 Enhancements of Dedicated Core Networks selection mechanism (Release 14), 3GPP, Sep. 2016.
[34] IETF RFC 6749: The OAuth 2.0 Authorization Framework.Available at https://tools.ietf.org/html/rfc6749.
[35] IETF RFC 7519: JSON Web Token (JWT) Available at: https://tools.ietf.org/html/rfc7519.
[36] IETF RFC 7515: JSON Web Signature (JWS) Available at: https://tools.ietf.org/html/rfc7515.
[37] 5G PPP Phase 1 project SELFNET Deliverable D3.4 Report and Prototype Implementation of the NFV & SDN Sensors and Actuators related to the Self-Optimizing Use Case, DOI: 10.18153/SLF-671672-D3_4, available at: https://bscw.selfnet-5g.eu/pub/bscw.cgi/d74902-5/*/*/*/*/*/DOI-D3.4.html.
[38] 5G PPP Phase 1 project SELFNET D3.1: Report and Prototype Implementation of the NFV & SDN Repository, https://bscw.selfnet-5g.eu/pub/bscw.cgi/d74922-5/*/*/*/*/*/DOI-D3.1.html
[39] 5G PPP Phase 1 project D6.2 Report and Prototypical Implementation of the NFV & SDN Application Manager, https://bscw.selfnet-5g.eu/pub/bscw.cgi/d99388-5/*/*/*/*/*/DOI-D6.2.html
[40] 3GPP TS 23.251 Multi-Operator Core Network (MOCN)
[41] 3GPP TR 23.707 Dedicated Core Networks (DECOR)
[42] 3GPP TS 29.244 Control and User Plane Separation of EPC nodes (CUPS)
[43] 3GPP TS 23.501 SBA System Architecture for the 5G System
[44] SliceNet 2nd year Review Report: https://bscw.slicenet.eu/sec/bscw.cgi/55087
[45] SliceNet Deliverable D8.4, "SliceNet System Integration and Testing (Iteration II)"

