

---

# 5G EVE

**5G European Validation platform for Extensive trials**

## Deliverable D4.4

### Report on benchmarking of new features and on the experimental portal (2<sup>nd</sup> version)

### *Project Details*

<b>Call</b>	H2020-ICT-17-2018
<b>Type of Action</b>	RIA
<b>Project start date</b>	01/07/2018
<b>Duration</b>	36 months
<b>GA No</b>	815074

### *Deliverable Details*

<b>Deliverable WP:</b>	WP4
<b>Deliverable Task:</b>	Tasks T4.5
<b>Deliverable Identifier:</b>	5G_EVE_D4.4
<b>Deliverable Title:</b>	Report on benchmarking of new features
<b>Editor(s):</b>	Paolo Giaccone (CNIT) Jaime Garcia-Reinoso (UC3M)
<b>Author(s):</b>	Jaime Garcia-Reinoso, Winnie Nakimuli, Pablo Serrano, Gines Garcia-Aviles, Jonathan Almodovar (UC3M) Giada Landi, Juan Brenes, Elian Kraja, Francesca Moscatelli, Leonardo Agueci (NXW) Ramon Perez (TELCARIA) Kostas Trichias, Athanasios Gkiolias (WINGS) Paolo Giaccone, Stefano Salsano, Nicola Blefari Melazzi, Marco Bonola, Luca Chiaraviglio, Paolo Lungaroni, Andrea Mayer, Davide Palmisano, Daniela Renga, Martino Trevisan (CNIT) Raymond Knopp, Florian Kaltenberger, Lionel Gauthier, Tsu-han Wang, Cedric Roux (ECOM) Luis M. Contreras, Lourdes Luque (TID)
<b>Reviewer(s):</b>	Sandra Martín Fernandez (ASTI) George Agapiou (OTE)
<b>Contractual Date of Delivery:</b>	30/06/2020
<b>Submission Date:</b>	29/06/2020
<b>Dissemination Level:</b>	PU
<b>Status:</b>	Final
<b>Version:</b>	1.0
<b>File Name:</b>	5G EVE_D4.4_Final

***Disclaimer***

*The information and views set out in this deliverable are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.*

*Deliverable History*

<b>Version</b>	<b>Date</b>	<b>Modification</b>	<b>Modified by</b>
<i>V0.1</i>	<i>22/03/2020</i>	<i>First ToC</i>	<i>Paolo Giaccone</i>
<i>V0.2</i>	<i>06/06/2020</i>	<i>Complete version of the Portal</i>	<i>Jaime Garcia-Reinoso</i>
<i>V0.3</i>	<i>07/06/2020</i>	<i>First complete version</i>	<i>Paolo Giaccone</i>
<i>V0.4</i>	<i>13/06/2020</i>	<i>After integrating ASTI comments</i>	<i>Paolo Giaccone</i>
<i>V0.8</i>	<i>16/06/2020</i>	<i>After integrating OTE comments</i>	<i>Paolo Giaccone</i>
<i>V0.12</i>	<i>23/06/2020</i>	<i>Final integration TID</i>	<i>Paolo Giaccone</i>
<b><i>V1.0</i></b>	<i>29/06/2020</i>	<i>Final version</i>	<i>M. Boldi, K. Trichias</i>

# Table of Content

<b>LIST OF ACRONYMS AND ABBREVIATIONS .....</b>	<b>6</b>
<b>LIST OF FIGURES.....</b>	<b>8</b>
<b>LIST OF TABLES.....</b>	<b>12</b>
<b>EXECUTIVE SUMMARY .....</b>	<b>14</b>
<b>1 INTRODUCTION .....</b>	<b>15</b>
<b>2 UPDATES TO THE EXPERIMENTAL 5G EVE PORTAL .....</b>	<b>16</b>
2.1 PORTAL BACKEND COMPONENTS.....	16
2.2 PORTAL GUI .....	41
2.3 5G EVE PORTAL USER GUIDE .....	47
<b>3 STATEFUL DATA PLANES TO COORDINATE VNFS.....</b>	<b>84</b>
3.1 INTRODUCTION.....	84
3.2 RELATED WORKS.....	85
3.3 STARE ARCHITECTURE AND PROTOCOL .....	85
3.4 STARE MIDDLEWARE .....	87
3.5 THE STARE P4 APPLICATION IN THE NETWORK SWITCHES.....	88
3.6 EXPERIMENTAL VALIDATION.....	91
3.7 STARE PROTOCOL AND MESSAGE SPECIFICATIONS.....	94
3.8 THE NORTHBOUND INTERFACE OF THE MIDDLEWARE.....	97
3.9 NF IMPLEMENTATION.....	99
3.10 STARE MIDDLEWARE IMPLEMENTATION.....	100
3.11 ALTERNATIVE IMPLEMENTATION WITH OPENFLOW SWITCHES .....	102
<b>4 MICROSERVICES-INSPIRED VNF RE-DESIGN.....</b>	<b>105</b>
4.1 MOTIVATION .....	105
4.2 APPROACH .....	105
4.3 ESTIMATED GAINS .....	106
<b>5 FLEXIBLE ETHERNET AND PREFERRED PATH ROUTING .....</b>	<b>108</b>
5.1 LATENCY AND JITTER WITH FLEXIBLE ETHERNET DATA PLANE .....	108
5.2 EXPERIMENTAL EVALUATION OF PREFERRED PATH ROUTING (PPR).....	109
<b>6 FULLY CLOUD-NATIVE IMPLEMENTATION OF OPENAIRINTERFACE RAN AND CORE.....</b>	<b>119</b>
6.1 CURRENT OAI FUNCTIONAL DECOMPOSITION OPTIONS (RADIO-ACCESS NETWORK).....	119
<b>7 SEGMENT ROUTING OVER IPV6: LINUX DATA PLANE IMPROVEMENTS FOR PERFORMANCE MONITORING.....</b>	<b>128</b>
7.1 SRV6 AND SRV6 PM (PERFORMANCE MONITORING) .....	128
7.2 PERFORMANCE MONITORING METHODOLOGIES AND THEIR STANDARDIZATION .....	129
7.3 SRV6 LOSS MEASUREMENT.....	130
7.4 LINUX IMPLEMENTATION .....	131
7.5 RESULTS.....	138
7.6 ANALYSIS OF RESULTS .....	142
<b>8 CONCLUSIONS .....</b>	<b>143</b>
<b>ACKNOWLEDGMENT .....</b>	<b>144</b>
<b>ANNEX A – UPDATED EXAMPLE OF REQUESTS RECEIVED BY THE DCM FROM THE ELM.....</b>	<b>145</b>
<b>REFERENCES .....</b>	<b>148</b>

## List of Acronyms and Abbreviations

<b>Acronym</b>	<b>Description</b>		
<b>AP</b>	Access Point	<b>IBN</b>	Intent Based Networking
<b>API</b>	Application Programming Interface	<b>IBNS</b>	Intent Based Networking System
<b>BLUO</b>	Browse-and-Lookup, Uploading and Onboarding	<b>ID</b>	Identifier
<b>CB</b>	Context Blueprint	<b>IAM</b>	Identity and Access Manager
<b>CD</b>	Context Descriptor	<b>IDMS</b>	Intent Driven Management System
<b>CLI</b>	Command Line Interface	<b>IL</b>	Instantiation Level
<b>CNF</b>	Cloud-native Network Functions	<b>I/W</b>	Interworking Layer
<b>CPU</b>	Core Processing Unit	<b>JPA</b>	Java Persistency API
<b>CRUD</b>	Create Read Update Delete	<b>JSON</b>	JavaScript Object Notation
<b>CSS</b>	Cascading Style Sheets	<b>KPI</b>	Key Performance Indicator
<b>CtxB</b>	Context Blueprint	<b>LM</b>	Loss Measurement
<b>DB</b>	Database	<b>LTE</b>	Long Term Evolution
<b>DCM</b>	Data Collection Manager	<b>MANO</b>	Management and Orchestration
<b>DCS</b>	Data Collection and Storage	<b>MAC</b>	Multi Access Control
<b>DF</b>	Deployment Flavour	<b>MEC</b>	Multi-Access Edge Computing
<b>DoS</b>	Deny of Service	<b>MSO</b>	Multi-Site Orchestrator
<b>DV</b>	Data Visualization	<b>MMTC</b>	Massive Machine Type Communication
<b>E2E</b>	End to End	<b>MPLS</b>	MultiProtocol Label Switching
<b>EBB</b>	Experiment Blueprint Builder	<b>NBI</b>	North Bound Interface
<b>eBPF</b>	Extended Berkeley Packet Filter	<b>NFV</b>	Network Function Virtualization
<b>EEM</b>	Experiment Execution Manager	<b>NF</b>	Network Function
<b>ELM</b>	Experiment Lifecycle Manager	<b>NFVO</b>	Network Function Virtualization Orchestrator
<b>eMBB</b>	Enhanced Mobile BroadBand	<b>NSD</b>	Network Service Descriptor
<b>eNB</b>	eNodeB	<b>OAI</b>	OpenAirInterface
<b>EPC</b>	Evolved Packet Core	<b>OAM</b>	Operation and Maintenance
<b>ETSI</b>	European Telecommunication Standard Institute	<b>OVS</b>	Open vSwitch
<b>ExpB</b>	Experiment Blueprint	<b>PCS</b>	Physical Coding Sublayer
<b>ExpD</b>	Experiment Descriptor	<b>PDU</b>	Protocol Data Unit
<b>FlexE</b>	Flexible Ethernet	<b>PHY</b>	Physical
<b>gNBs</b>	Next Generation NodeBs	<b>PM</b>	Performance Monitoring
<b>GE</b>	Gigabit Ethernet	<b>PNF</b>	Physical Network Function
<b>GUI</b>	Graphical User Interface	<b>PPR</b>	Preferred Path Routing
<b>HTML</b>	Hyper Text Markup Language	<b>RAU</b>	Radio Aggregation Unit
<b>HTTP</b>	Hyper Text Transfer Protocol	<b>RBAC</b>	Role Based Access Control
		<b>RAN</b>	Radio Access Network

<b><i>RCC</i></b>	Radio Cloud Center	<b><i>TSB</i></b>	Ticketing System Backend
<b><i>REST</i></b>	REpresentational State Transfer	<b><i>UDP</i></b>	User Datagram Protocol
<b><i>RFC</i></b>	Request for Comment	<b><i>URL</i></b>	Uniform Resource Locator
<b><i>RRU</i></b>	Remote Radio Unit	<b><i>URLLC</i></b>	Ultra-Reliable Low-Latency Communication
<b><i>RSS</i></b>	Resident Set Size	<b><i>VIM</i></b>	Virtual Infrastructure Manager
<b><i>SDN</i></b>	Software Defined Networking	<b><i>VM</i></b>	Virtual Machine
<b><i>SID</i></b>	Segment ID	<b><i>VNF</i></b>	Virtual Network Function
<b><i>SLA</i></b>	Service Level Agreement	<b><i>VNF</i></b>	Virtual Network Function Descriptor
<b><i>SNMP</i></b>	Simple Network Management Protocol	<b><i>VNF</i></b>	Virtual Network Function Descriptor
<b><i>SR</i></b>	Segment Routing	<b><i>VNF</i></b>	Virtual Network Function Forwarding Graph
<b><i>SRH</i></b>	Segment Routing Header	<b><i>VNF</i></b>	Virtual Network Function Forwarding Graph
<b><i>SRv6</i></b>	Segment Routing over IPv6	<b><i>VSB</i></b>	Vertical Service Blueprint
<b><i>STARE</i></b>	STate REplication	<b><i>VSD</i></b>	Vertical Service Descriptor
<b><i>SUT</i></b>	System Under Test	<b><i>WP</i></b>	Work Package
<b><i>TCB</i></b>	Test Case Blueprint	<b><i>YAML</i></b>	Yet Another Markup Language

## List of Figures

Figure 1: Updated architecture of the DCS .....	26
Figure 2: DCS: Dashboard creation workflow .....	27
Figure 3: DCS: Dashboard retrieval workflow.....	27
Figure 4: DCS: Dashboard deletion workflow .....	28
Figure 5: DCS Java logic data model .....	29
Figure 6: Interface to upload VNF packages to a selected number of sites.....	41
Figure 7: Interface to list all VNF packages uploaded. ....	42
Figure 8: Updated architecture of the DV .....	43
Figure 9: Screenshot of the Portal GUI containing several Kibana dashboards.....	43
Figure 10: Visualization of VNF Packages - list.....	44
Figure 11: Visualization of VNF Packages – descriptor in textual mode .....	44
Figure 12: Visualization of VNF Packages – VNFD graph .....	45
Figure 13: Visualization of Network Service Descriptors - list .....	45
Figure 14: Visualization of Network Service Descriptors – descriptor in textual mode .....	46
Figure 15: Visualization of Network Service Descriptors – NSD graph.....	46
Figure 16: 5G EVE PORTAL WELCOME PAGE.....	49
Figure 17: 5G EVE PORTAL EXPERIMENT DESIGN PAGE .....	49
Figure 18: 5G EVE PORTAL VERTICAL SERVICE BLUEPRINTS DESIGN PAGE .....	50
Figure 19: 5G EVE PORTAL VSB TRANSLATION RULES .....	51
Figure 20: 5G EVE VIEW UPLOADED BLUEPRINT .....	51
Figure 21: 5G EVE PORTAL UPLOADED VSB VIEW .....	52
Figure 22: 5G EVE EXECUTION CONTEXT BLUEPRINTS UPLOAD .....	52
Figure 23: 5G EVE Execution Context Blueprints view.....	53
Figure 24: 5G EVE ASTI Use-case delay context blueprint view .....	53
Figure 25: 5G EVE ASTI Use-case delay context blueprint view .....	54
Figure 26: 5G EVE ASTI Use-case Background traffic context blueprint view.....	54
Figure 27: 5G EVE Portal Test case blueprint onboarding .....	55
Figure 28: Create Test Case Blueprint .....	56
Figure 29: 5G EVE Portal simple_apache_UseCase_TCB .....	57
Figure 30: 5G EVE SIMPLE_APACHE_UseCase_TCB VIEW .....	58
Figure 31: 5G EVE PORTAL EXPERIMENT BLUEPRINT DESIGN.....	58
Figure 32: 5GEVE Portal ASTI_USECASE Experiment Blueprint .....	59
Figure 33: 5G EVE PORTAL VSB Selection.....	59
Figure 34: 5G EVE PORTAL VSB Selection.....	60
Figure 35: 5G EVE ASTI_USeCase CtxB Selection .....	60
Figure 36: 5G EVE ASTI USE_CASE COMPOSITE NSD UPLOADING .....	60



Figure 37: 5G EVE Composite NSD Translation rules..... 61

Figure 38: 5G EVE ASTI Composite NSD Translation rules..... 61

Figure 39: 5G EVE Experiment blueprint metrics & KPIs section..... 62

Figure 40: 5G EVE ASTI Use-Case Metrics & KPIs ..... 64

Figure 41: 5G EVE ExpB Test Case Blueprint Selection ..... 65

Figure 42: 5G EVE EXPERIMENT BLUEPRINTS VIEW ..... 65

Figure 43: 5G EVE ASTI USE-CASE EXPERIMENT BLUEPRINT ..... 66

Figure 44: 5G EVE PORTAL HOME PAGE ..... 66

Figure 45: 5G EVE Portal Request Experiments page..... 67

Figure 46: 5G EVE Experiment Descriptor metadata page..... 67

Figure 47: 5G EVE ASTI USE-CASE Experiment Descriptors Metadata ..... 68

Figure 48: 5G EVE Vertical Service Descriptor page..... 68

Figure 49: 5G EVE ASTI Use-Case VSD parameters ..... 69

Figure 50: 5G EVE ASTI Use-Case ExpD KPI..... 69

Figure 51: 5G EVE ASTI ExpD Delay\_Context..... 70

Figure 52: 5G EVE ASTI ExpD Delay\_Context Parameter ..... 70

Figure 53: 5G EVE ExpD Test Cases Configuration ..... 70

Figure 54: 5G EVE ASTI ExpD Delay Test Case Parameters ..... 71

Figure 55: 5G EVE EXPERIMENT DESCRIPTORS VIEW..... 71

Figure 56: 5G EVE ASTI Use-Case Experiment Descriptor Details ..... 71

Figure 57: 5G EVE Request Experiment using The IBN Tool ..... 72

Figure 58: 5G EVE IBN RBAC page..... 72

Figure 59: 5G EVE Intent Expression..... 73

Figure 60: 5G EVE Intent based tool-Experiment blueprint and service parameters selection ..... 74

Figure 61: 5G EVE Intent Guided selection-select site..... 74

Figure 62: 5G EVE IBN tool: Site- Available Services ..... 75

Figure 63: 5G EVE IBN tool: Site- select desired experiment..... 75

Figure 64: 5G EVE IBN tool: Site- Select vsb and ctb parameters..... 75

Figure 65: 5G EVE IBN tool: Site- select test case parameters ..... 76

Figure 66: 5G EVE IBN tool: Site-schedule experiment using the intent based tool ..... 76

Figure 67: 5G EVE Request Experiments page ..... 77

Figure 68: 5G EVE Schedule a New Experiment ..... 77

Figure 69: 5G EVE ASTI Use-Case Scheduled Experiment..... 78

Figure 70: 5G EVE Manage Experiment Section ..... 78

Figure 71: 5G EVE EXPERIMENT LIST ..... 79

Figure 72: 5G EVE Experiment Accepted ..... 79

Figure 73: 5G EVE Experiment Ready ..... 79

Figure 74: 5G EVE Execute an experiment ..... 80

Figure 75: 5G EVE Experiment Instantiating ..... 80

Figure 76: 5G EVE Instantiated experiment ..... 80

Figure 77: 5G EVE Execute experiment tests ..... 81

Figure 78: 5G EVE Experiment execution running ..... 81

Figure 79: 5G EVE simple APACHE use case KPI and metrics graphs..... 81

Figure 80: 5G EVE View experiment tests report..... 82

Figure 81: 5G EVE Experiment tests summary ..... 82

Figure 82: 5G EVE Validation report..... 83

Figure 83: 5G EVE UC 5G KPI VALIDATION\_I..... 83

Figure 84: Smart City UC in the 5G EVE project based on transparent WiFi scanners connected through eNB. .... 84

Figure 85: Stateful data plane for state replication in the Smart City use case ..... 84

Figure 86: STARE architecture for state replication ..... 86

Figure 87: Initialization phase according to the STARE protocol ..... 86

Figure 88: State replication phase according to the STARE protocol..... 87

Figure 89: Integration of STARE architecture for the Smart City: Safety and Environment - Smart Turin UC ..... 87

Figure 90: STARE middleware in a simplified scenario with 4 NFs, 2 servers and 1 P4 switch ..... 88

Figure 91: The software architecture for STARE Middleware ..... 88

Figure 92: Example of interaction for the local controller-based P4 implementation..... 90

Figure 93: Emulated topology for the experimental evaluation. .... 91

Figure 94: NFs, middleware and REPLICCA controller running in the STARE testbed ..... 92

Figure 95: Pure OpenFlow scenario ..... 92

Figure 96: RSS measurement for the register-based solution ..... 93

Figure 97: Increase in occupied memory for the embedded controller-based solution..... 93

Figure 98: Remote Controller-based RSS measurement..... 93

Figure 99: IPv4 destination address format for the state replication ..... 94

Figure 100: Information Request phase to get the NF identifier Global-ID during the Initializing phase..... 95

Figure 101: Information Request phase to obtain the variable-id in the Initializing phase..... 95

Figure 102: Publish-Header at application layer ..... 96

Figure 103: Protocol for the state replication through the P4 switch ..... 97

Figure 104: Protocol between the NFs and the middleware..... 98

Figure 105: The internal architecture for the NFs. .... 100

Figure 106: Interaction of the Middleware with the network. .... 101

Figure 107: Protocol for Remote Controller-based scenario with OpenFlow switch..... 103

Figure 108: The simplified threading architecture of the srsLTE software (left) and a possible serverless design of the software stack (right)..... 106

Figure 109: The liquid scalability (top) and an empirical evaluation (bottom)..... 107

Figure 110: FlexE Shim layer as standardized by OIF..... 108

Figure 111: Flexible Ethernet test setup..... 109

Figure 112: PPR test topology..... 110

Figure 113: PPR configured paths..... 110

Figure 114: ISIS LSP..... 115

Figure 115: PPR TLV structure..... 115

Figure 116: PPR TLVs ..... 116

Figure 117: Capture traffic R11. .... 118

Figure 118: Functional Decomposition of OAI RAN Components ..... 121

Figure 119: Current NFAPI (LTE) Protocol Split..... 121

Figure 120: Current openairinterface-k8s functional decomposition ..... 123

Figure 121: Evolution of openairinterface-k8s..... 124

Figure 122: Computing cluster ..... 124

Figure 123: Outdoor 3.5 GHz Radio units ..... 126

Figure 124: Indoor 2.6/3.5 GHz radio units ..... 126

Figure 125: URLLC Testing Scenario..... 127

Figure 126: Alternate coloring method (RFC 8321) ..... 129

Figure 127: Reference SRv6 network scenario for Performance Monitoring ..... 130

Figure 128: Packet processing in the Egress node ..... 133

Figure 129: Packet processing in the Ingress node..... 133

Figure 130: Iptables user space tool ..... 134

Figure 131: IPset userspace tool..... 135

Figure 132: Packet processing and statistics management in the Egress node using the PF-PLM eBPF implementation..... 137

Figure 133: Packet processing and statistics management in the Ingress node using the PF-PLM eBPF implementation..... 138

Figure 134: Testbed architecture ..... 139

Figure 135: SUT throughput (ingress node configuration) ..... 140

Figure 136: SUT throughput (egress node configuration)..... 140

Figure 137: SUT throughput (ingress node configuration) ..... 141

Figure 138: SUT throughput (egress node configuration) needed for parsing the packet headers and for extracting the flows. .... 142

## List of Tables

Table 1: Role-Based Access Control (RBAC) REST API .....	16
Table 2: REST API - POST /portal/rbac/register .....	17
Table 3: REST API – POST /portal/rbac/login .....	17
Table 4: REST API – POST /portal/rbac/refreshToken.....	17
Table 5: REST API – GET /portal/rbac/logout .....	18
Table 6: REST API – GET /portal/rbac/extra/use-cases .....	18
Table 7: REST API – POST /portal/rbac/extra/use-cases .....	18
Table 8: REST API – DELETE /portal/rbac/extra/use-cases .....	19
Table 9: REST API – GET /portal/rbac/extra/managed-sites.....	19
Table 10: REST API – POST/portal/rbac/extra/managed-sites.....	19
Table 11: REST API – DELETE /portal/rbac/extra/managed-sites .....	19
Table 12: Experiment Lifecycle Manager REST API.....	20
Table 13: REST API – POST /portal/elm/experiment .....	21
Table 14: REST API – GET /portal/elm/experiment<?[filter_parameters]> .....	21
Table 15: REST API – PUT /portal/elm/experiment/{expId}/status .....	21
Table 16: REST API – PUT /portal/elm/experiment/{expId}/timeslot.....	22
Table 17: REST API – POST /portal/elm/experiment/{expId}/action/deploy.....	22
Table 18: REST API – POST /portal/elm/experiment/{expId}/action/execute .....	22
Table 19: REST API – POST /portal/elm/experiment/{expId}/action/terminate .....	23
Table 20: REST API – DELETE /portal/elm/experiment/{expId} .....	23
Table 21: Data Collection and Storage REST API.....	24
Table 22: REST API – POST /portal/dcs/dashboard.....	24
Table 23: REST API – GET /portal/dcs/dashboard/{experimentId} .....	25
Table 24: REST API – DELETE /portal/dcs/dashboard .....	25
Table 25: REST API – POST /portal/dcs/start_signalling .....	25
Table 26: REST API – DELETE /portal/dcs/stop_signalling .....	25
Table 27: File Storage REST API .....	30
Table 28: REST API - GET /portal/fs .....	30
Table 29: REST API - POST /portal/fs/upload/{filename} .....	30
Table 30: REST API - POST /portal/fs/sites/{filename} .....	30
Table 31: REST API - POST /portal/fs/status/{filename} .....	31
Table 32: REST API - POST /portal/fs/download/{filename} .....	31
Table 33: REST API - POST /portal/fs/delete/{filename} .....	31
Table 34: 5G-EVE Portal Catalogue REST API .....	32
Table 35: REST API – GET /portal/catalogue/ctxblueprint<?[filter_parameters]>.....	33
Table 36: REST API – POST /portal/catalogue/ctxblueprint.....	33

Table 37: REST API – GET /portal/catalogue/ctxblueprint/<id> ..... 33

Table 38: REST API – DELETE /portal/catalogue/ctxblueprint/<id> ..... 34

Table 39: REST API – GET /portal/catalogue/vsblueprint<?[filter\_parameters]> ..... 34

Table 40: REST API – POST /portal/catalogue/vsblueprint ..... 34

Table 41: REST API – GET /portal/catalogue/vsblueprint/<id> ..... 35

Table 42: REST API – DELETE /portal/catalogue/vsblueprint/<id> ..... 35

Table 43: REST API – GET /portal/catalogue/expblueprint<?[filter\_parameters]> ..... 35

Table 44: REST API – POST /portal/catalogue/expblueprint ..... 36

Table 45: REST API – GET /portal/catalogue/expblueprint/<id> ..... 36

Table 46: REST API – DELETE /portal/catalogue/expblueprint/<id> ..... 36

Table 47: REST API – GET /portal/catalogue/expdescriptor<?[filter\_parameters]> ..... 36

Table 48: REST API – POST /portal/catalogue/expdescriptor ..... 37

Table 49: REST API – GET /portal/catalogue/expdescriptor/<id> ..... 37

Table 50: REST API – DELETE /portal/catalogue/expdescriptor/<id> ..... 37

Table 51: Ticketing System backend REST API ..... 38

Table 52: REST API – GET /portal/tsb/products ..... 38

Table 53: REST API - GET /portal/tsb/components ..... 38

Table 54: REST API - GET /portal/tsb/adminusers ..... 39

Table 55: REST API -POST /portal/tsb/tickets ..... 39

Table 56: REST API -GET /portal/tsb/tickets ..... 39

Table 57: REST API -GET /portal/tsb/tickets/trusted ..... 39

Table 58: REST API -GET /portal/tsb/tickets/{ticket\_id} ..... 40

Table 59: REST API -GET /portal/tsb/tickets/{ticket\_id}/comments ..... 40

Table 60: REST API -POST /portal/tsb/tickets/{ticket\_id}/comments ..... 40

Table 61: REST API -POST /portal/tsb/tickets/{ticket\_id}/comments/trusted ..... 40

Table 62: 5G EVE Metrics & KPIs parameters ..... 62

Table 63: 5G EVE VSD Slice Parameters Description ..... 68

Table 64: 5G EVE intent based selection parameters ..... 73

## Executive Summary

This deliverable presents two of the main milestones of the project: on the one hand, we present the second version of the 5G EVE Portal, including all services planned at the beginning of the project and, on the other hand, the results and benchmarking of innovative data planes and radio access features.

With respect to the second version of the 5G EVE Portal, this deliverable includes an updated version of the REST API offered by the Portal backend to both the GUI elements developed by the project, and to other authorized projects by means of the public methods defined by the 5G EVE project. This second version of the Portal includes new services, based on the requirements imposed by other technical work packages. Thus, we have included some implementation details of these new services. Finally, we have also included a new update of the service handbook, presenting a complete 5G EVE Portal user guide, which for sure will be helpful for all actors using our portal.

Furthermore, the deliverable explores many research directions to exploit innovative data plane techniques: (i) the use of P4 abstraction model to share non-local states between the VNFs, (ii) the comparison of different approaches to support VNF, (iii) the use of two novel networking technologies, namely Flexible Ethernet and of Preferred Path Routing, (iv) a fully cloud-native implementation of OpenAirInterface RAN and Core, (v) the implementation of an efficient and accurate solution for loss monitoring in SRv6 (Segment Routing over IPv6).

# 1 Introduction

The deliverable is divided in two parts, each of them corresponding to two of the main milestones of the project.

The first part of this document presents in Section 2 the second version of the 5G EVE Portal, with a fully operational ecosystem designed and implemented to facilitate all the different stages of an end-to-end 5G experiment. The 5G EVE Portal, publicly available at <https://portal.5g-eve.eu/>, presents different views depending on the role of the user logged in. VNF uploaders may upload VNF packages to be onboarded in one or several sites of the 5G EVE platform. Site managers, in turn, receive a ticket notifying the availability of a new VNF package, so they can download and process it. Experiment developers may onboard VSBs, CtxB, TCB and ExpB. Experimenters have two tools to create an experiment descriptor: intent-based tool and a guided mechanism. After that, experimenters request the scheduling of the experiment, which has to be accepted by the site manager. After an experiment is ready, an experimenter can deploy the experiment, execute experiments, monitor the results and validate the KPIs, among others. Furthermore, apart from the Portal GUI, the 5G EVE project offers access to the Portal backend by means of its public REST API, where authorized users can consume some services. The complete list of methods provided by the REST API, both the public and private ones, are listed in this document. This deliverable also reports the main updates to the implementation of both the backend and GUI components. Finally, we have also included a complete 5G EVE Portal user guide, which extends the service handbook presented in D4.2.

The second part of the deliverable, Sections 3-7, is divided into five distinct contributions. These contributions are the main outcomes of 5G EVE Task 4.5, titled “Implementation and benchmarking of innovative data planes and radio access features”, and can be seen as complementary approaches.

The first contribution, in Section 3, refers to the adoption of innovative data planes for the sharing of states between the VNFs. This allows to coordinate the actions of many concurrent instances of the same VNF, improving both the scalability and the reactivity of the implemented network policies. Finally, the section shows the feasibility of adopting P4 as programming abstraction for the data plane.

The second contribution, in Section 4, discusses different approaches to support VNF with a high scalability in terms of both resource and time granularity and to elastically adapt to the instantaneous demand. The provided results investigate the “cost” of softwarizing in terms of required resources to sustain a given traffic demand and show that the CPU resources are impaired by high traffic flows, limiting the scalability of softwarized solutions.

The third contribution, in Section 5, evaluates experimentally Flexible Ethernet, chosen as innovative data plane, to enable uRLLC services, which are crucial for 5G applications. Furthermore, it investigates a new source routing protocol encapsulation method, namely Preferred Path Routing (PPR), which follows the Segment Routing (SR) paradigm.

The fourth contribution, in Section 6, describes a fully cloud-native implementation of OpenAirInterface RAN and Core, in order to meet the requirements of uRLLC applications. In addition, it discusses openairinterface-k8s, which is a cloud-native flavor of OpenAirInterface, comprising both hardware and software real-time components. This allows to leverage the cloud-based architecture of 5G EVE for the RAN.

The fifth contribution, in Section 7, describes the design and implementation of an efficient and accurate solution for loss monitoring in SRv6 (Segment Routing over IPv6) and the evaluation of its performance. This approach is aimed at exploring an innovative data plane paradigm. SRv6 enables a fully programmable data plane, in which packets carry the “instructions” to process the packets across the network, differently from the standard SDN in which the controller interacts directly with the SDN switches.

## 2 Updates to the experimental 5G EVE Portal

This section covers all updates to the experiment 5G EVE Portal already presented in 5G EVE deliverables D4.1 [1] and D4.2 [2]. The Portal architecture has not changed since the first version already reported in D4.2 [2], but some extensions have been introduced in the functionality provided by some components, due to new requirements imposed by both WP4 and other technical 5G EVE work packages. Similarly to D4.2 [2], this section is split in two parts: Portal backend components and Portal GUI.

### 2.1 Portal backend components

Portal backend components are the core elements of the 5G EVE Portal, offering services to other components of the backend and to other elements. These other elements could be the Portal GUI, which will be explained in Section 2.2, but it could also provide services to authorized elements outside of the 5G EVE environment, mainly focused to H2020 ICT-19 projects using our platform. To better distinguish private and public services, we have included a new column in all tables that summarizes the REST API offered by each component.

As already described in deliverable D4.2 [2], the main components of the 5G EVE Portal backend are:

- The Role-Based Access Control, which provides a unified user management and authentication/authorization control.
- The Experiment Lifecycle Manager is in charge of the lifecycle of all experiments.
- The Data Collection and Storage component collects all metrics published by both the applications and the infrastructure.
- The File Storage component is an element used to store large files like virtual machine images.
- The Catalogue Service stores all the blueprints and descriptors associated to the definition of a 5G EVE experiment.
- The Ticketing System component is in charge of receiving and sending tickets generated by end-users or by other components of the 5G EVE Portal

In the next sections we explain the updates done to all these components since deliverable D4.2 in terms of new services, which implies new methods provided by the REST API, and modifications in terms of implementation details. In each section, we will first explain again the objectives of the component, next we introduce a summary table with all available methods and endpoints, including a column to show if each method is public or not, to then explain in detail each of these methods in individual tables.

#### 2.1.1 Role-Based Access Control

The Role-Based Access Control (RBAC) provides user management and authentication/authorization functionality. This component relies on Keycloak<sup>1</sup>, which is an Identity and Access Management service (IAM) to store users, generate and renew authentication artefacts that belong to each user. The RBAC component exposes a REST API in order to make all the functionality available to the front-end application and for other back-end components that might need it, thus avoiding HTTP redirections.

##### 2.1.1.1 Updated Role-Based Access Control Interface (Irbac)

**Table 1: Role-Based Access Control (RBAC) REST API.**

HTTP method	URI	Description	Public
POST	/portal/rbac/register	Endpoint to create a new user at the portal	Yes

<sup>1</sup> <https://www.keycloak.org/>



<b>POST</b>	/portal/rbac/login	Allows the creation of a new session for a specific user who is already registered at the portal.	Yes
<b>POST</b>	/portal/rbac/refreshtoken	Allows users to obtain a new access token once the one they have has expired	Yes
<b>GET</b>	/portal/rbac/logout	Closes a user session	Yes
<b>GET</b>	/portal/rbac/extra/use-cases	Enables the retrieval of use cases associated to a specific user	Yes
<b>POST</b>	/portal/rbac/extra/use-cases	Allows the addition of new use cases to a specific user	Yes
<b>DELETE</b>	/portal/rbac/extra/use-cases	Enables the removal of use cases associated to a specific user	Yes
<b>GET</b>	/portal/rbac/extra/managed-sites	Allows the retrieval of sites managed by a specific user	Yes
<b>POST</b>	/portal/rbac/extra/managed-sites	Enables the addition of sites managed by a specific user	Yes
<b>DELETE</b>	/portal/rbac/extra/managed-sites	Enables the removal of sites managed by a specific user	Yes

**Table 2: REST API - POST /portal/rbac/register**

<b>POST /portal/rbac/register</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request Body (JSON)	email	String	Email address of the new user
	username	String	Username of the new user
	firstName	String	First Name of the new user
	lastName	String	Last Name of the new user
	password	String	Password
Response body			
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

**Table 3: REST API – POST /portal/rbac/login**

<b>POST /portal/rbac/login</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request Body (JSON)	email	String	Email address of the new user
	password	String	Password
Response body	access_token	String	Access token to include in all the requests for resources
	Refresh_token	String	Refresh token that allows access token renewal
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

**Table 4: REST API – POST /portal/rbac/refreshtoken**

<b>POST /portal/rbac/refreshtoken</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>

Request Body (JSON)	refresh_token	String	Refresh token to generate a new access token
Response body	access_token	String	Access token to include in all the requests for resources
	refresh_token	String	Refresh token that allows access token renewal
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

**Table 5: REST API – GET /portal/rbac/logout**

<b>GET /portal/rbac/logout</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)			
Response body			
Successful response HTTP code: 204 NO_CONTENT			
Error response HTTP code: 401 UNAUTHORIZED, 500 INTERNAL ERROR			

**Table 6: REST API – GET /portal/rbac/extra/use-cases**

<b>GET /portal/rbac/extra/use-cases</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)			
Response body (JSON)	details	Array<String>	Use cases associated to the user
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 7: REST API – POST /portal/rbac/extra/use-cases**

<b>POST /portal/rbac/extra/use-cases</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)	use_cases	Array<String>	List of use cases to be added to the user
Response body (JSON)			
Successful response HTTP code: 204 NO_CONTENT			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 8: REST API – DELETE /portal/rbac/extra/use-cases**

<b>DELETE /portal/rbac/extra/use-cases</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)	use_cases	Array<String>	List of use cases to be removed
Response body			
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 9: REST API – GET /portal/rbac/extra/managed-sites**

<b>GET /portal/rbac/extra/managed-sites</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)			
Response body (JSON)	details	Array<String>	Sites managed by the specific user
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 10: REST API – POST /portal/rbac/extra/managed-sites**

<b>POST /portal/rbac/extra/managed-sites</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)	managed_sites	Array<String>	List of sites to be added at the managed sites list of the user
Response body (JSON)			
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 11: REST API – DELETE /portal/rbac/extra/managed-sites**

<b>DELETE /portal/rbac/extra/managed-sites</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)	managed_sites	Array<String>	List of sites to be removed from the managed sites list associated to the user
Response body			
Successful response HTTP code: 204 NO CONTENT			

Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST

### 2.1.1.2 Implementation updates

Together with small optimization improvements at the Authentication, we have included some functionality to have a better coordination between the different modules of our system.

First, we have added the ability of managing the different use cases in a centralized way, being then able to create, update, read and delete use cases associated to users. Then, we also added a functionality to manage the different sites that a site manager is taking care of. These updates enabled an efficient and standard way of managing this information between the different modules that conforms the back-end infrastructure.

## 2.1.2 Experiment Lifecycle Manager

The Experiment Lifecycle Manager (ELM) is the Portal backend component handling the requests to create new experiment instances or to perform actions related to existing experiments, offering a REST-based north-bound interface (NBI) that can be invoked by external REST clients to manage experiments. This REST API is typically used by the 5G EVE Portal GUI, which mediates between a 5G EVE user and the 5G EVE platform for all the procedures related to the creation, management and monitoring of experiments. However, the system allows also external clients (e.g., platforms developed in the context of ICT-19 projects) to directly access the ELM functionalities via the public methods included in the REST API. All the requests received at the ELM NBI are authenticated and authorized through the RBAC component (see Section 2.1.1), according to the actions permitted for the different 5G EVE roles, as defined in D4.1 [1].

### 2.1.2.1 Updated Experiment Lifecycle Manager Interface (Ielm)

**Table 12: Experiment Lifecycle Manager REST API**

HTTP method	URI	Description	Public
<b>POST</b>	/portal/elm/experiment	Creates a new experiment.	Yes
<b>GET</b>	/portal/elm/experiment<?[filter_parameters]>	Returns a list of experiments matching a given filter, i.e. based on the experiment identifier or the experiment descriptor identifier. The acceptable filter_parameters are the following: <Exp_ID; ExpD_ID>.	Yes
<b>PUT</b>	/portal/elm/experiment/{expId}/status	Changes the status of an experiment. The permitted changes are the following: - from scheduling to accepted - from scheduling to refused - from accepted to ready	No
<b>PUT</b>	/portal/elm/experiment/{expId}/timeslot	Changes the proposed timeslot for an experiment. The experiment must be in scheduling state.	No
<b>POST</b>	/portal/elm/experiment/{expId}/action/deploy	Deploys the virtual environment to run the experiment (i.e. instantiates the associated NFV network service). The experiment must be in ready state.	Yes
<b>POST</b>	/portal/elm/experiment/{expId}/action/execute	Executes of one or more tests for a given experiment. The experiment must be in instantiated state.	Yes
<b>POST</b>	/portal/elm/experiment/{expId}/action/terminate	Terminates the virtual environment where the experiment has been executed (i.e. terminates	Yes

		the associated NFV network service). The experiment must be in instantiated state.	
<b>DELETE</b>	/portal/elm/experiment/{expId}	Removes an experiment and its record from the system. The experiment must be in refused, terminated or failed state.	Yes

The OpenAPI of the Experiment Lifecycle Manager is available at the following link:

<https://github.com/nextworks-it/experiment-portal/blob/master/API/ExperimentLifecycleManager.json>

**Table 13: REST API – POST /portal/elm/experiment**

<b>POST /portal/elm/experiment</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	experimentName	String	Name of the experiment instance.
	experimentDescriptorId	String	ID of the experiment descriptor defining the characteristics of the experiment to be created.
	proposedTimeSlot	ExperimentExecutionTimeslot	Definition of the timeslot proposed for the execution of the experiment. Includes start <code>Time</code> and end <code>Time</code> .
	targetSites	List<EveSite>	List of sites where the experiment must be instantiated and executed.
	useCase	String	Use case associated to the experiment.
Response body	experimentId	String	ID of the created experiment.
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 14: REST API – GET /portal/elm/experiment<?[filter\_parameters]>**

<b>GET /portal/elm/experiment&lt;?[filter_parameters]&gt;</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
URI Query Parameters	expId	String	ID of the requested experiment.
	expDId	String	ID of the experiment descriptor associated to the requested experiments.
Response body	experiments	List<Experiment>	Information about the requested experiments, including their results if available.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 15: REST API – PUT /portal/elm/experiment/{expId}/status**

<b>PUT /portal/elm/experiment/{expId}/status</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	status	Enum	New status of the experiment.

URI Variables	expId	String	ID of the experiment to be updated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 16: REST API – PUT /portal/elm/experiment/{expId}/timeslot**

<b>PUT /portal/elm/experiment/{expId}/timeslot</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	timeslot	ExperimentExecution-Timeslot	Definition of the new timeslot proposed for the execution of the experiment. Includes startTime and endTime.
URI Variables	expId	String	ID of the experiment to be updated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 17: REST API – POST /portal/elm/experiment/{expId}/action/deploy**

<b>POST /portal/elm/experiment/{expId}/action/deploy</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	--	--	--
URI Variables	expId	String	ID of the experiment to be deployed.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 18: REST API – POST /portal/elm/experiment/{expId}/action/execute**

<b>POST /portal/elm/experiment/{expId}/action/execute</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	executionName	String	Name of the requested execution.
	testCaseDescriptor-Configuration	Map<String, Map<String, String>>	Specification of the test cases to be executed and their user configuration. The key of the outer map is the testCaseDescriptorId. The value is an inner map with the user parameters for that test cases. The test cases and the user parameters specified in the request override the ones defined in the experiment descriptor. If this field is empty, all the test cases defined in

			the experiment descriptor are executed by default.
URI Variables	expId	String	ID of the experiment to be executed.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

Table 19: REST API – POST /portal/elm/experiment/{expId}/action/terminate

POST /portal/elm/experiment/{expId}/action/terminate			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	expId	String	ID of the experiment to be terminated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

Table 20: REST API – DELETE /portal/elm/experiment/{expId}

DELETE /portal/elm/experiment/{expId}			
	Parameter name	Parameter type	Description
URI Variables	expId	String	ID of the experiment to be deleted.
Response body	--	--	--
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

### 2.1.2.2 Implementation updates

Implementation details can be found in 5G EVE deliverables D4.1 [1] and D4.2 [2].

### 2.1.3 Data Collection and Storage

The Data Collection and Storage component (DCS) placed in the Portal backend is one of the components that belongs to the Experiment monitoring and Maintenance & Results Collection toolchain, already presented in D4.1 [1]. Two different tools can be distinguished:

- The data collection, aggregation and pre-processing tool, which collects the experiment results, metrics (both infrastructure and application metrics) and KPIs generated for a given experiment provided by the Data Collection Manager component from the I/W Framework.
- The data indexing and storage tool, which enables the capability of searching and filtering among all the data gathered by the data collection tool for obtaining the useful information that will be displayed afterwards in the Portal GUI, also providing data persistence.

### 2.1.3.1 Updated Data Collection and Storage Interface (Ids)

The Data Collection and Storage interface presents several changes with regards to the version provided in deliverable D4.2 [2], in which a REST API based on the operations exposed by the Elastic Stack<sup>2</sup> was presented. In this new release of the 5G EVE Portal, it has been decided that all these operations will not be provided to the users, remaining only for administration’s purposes.

Then, the operations that can be handled by other components in the Portal architecture will be related only to the following functionalities offered by the DCS:

- The management of Kibana dashboards.
- The management of the signalling topics.

The first functionality will be enabled by a specific Java logic embedded in the DCS, and the second one, as it is simpler, will be provided with a simple Python logic implementing Flask. This architecture will be further explained in Section 2.1.3.2.

Then, the updated REST API exposed by these two components in the DCS is the following:

**Table 21: Data Collection and Storage REST API**

HTTP method	URI	Description	Public
POST	/portal/dcs/dashboard	Create the Kibana dashboards for the metrics and/or KPIs provided in the request.	No
GET	/portal/dcs/dashboard/{experimentId}	Retrieve the Kibana dashboards’ URLs already created for the metrics and/or KPIs related to the {experimentId} experiment.	Yes
DELETE	/portal/dcs/dashboard	Delete the Kibana dashboards for the metrics and/or KPIs provided in the request.	No
POST	/portal/dcs/start_signalling	Enable the signalling topics. Internal operation.	No
DELETE	/portal/dcs/stop_signalling	Disable the signalling topics. Internal operation.	No

The specification of the REST APIs for interacting with both functionalities can be checked in their Github repositories: <https://github.com/5GEVE/5geve-wp4-dcs-kibana-dashboards-handler> for the Kibana dashboards’ handler, and <https://github.com/5GEVE/5geve-wp4-dcs-signalling-topic-handler> for the signalling topics’ handler.

**Table 22: REST API – POST /portal/dcs/dashboard**

POST /portal/dcs/dashboard			
	Parameter name	Parameter type	Description
Request body	dcsRequestWrapper	DcsRequestWrapper	It provides a list of RecordWrapper, where each RecordWrapper contains a single ValueWrapper that represents the data related to the metric or KPI whose dashboard will be created afterwards. This

<sup>2</sup> <https://www.elastic.co/elastic-stack>



			format is the same as the message sent from the ELM to the DCM <sup>3</sup> .
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED.			
Error response HTTP code: 400 BAD REQUEST.			

**Table 23: REST API – GET /portal/dcs/dashboard/{experimentId}**

<b>GET /portal/dcs/dashboard/{experimentId}</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	--	--	--
URI Variables	experimentId	String	Value of the experimentId from which the metrics' and KPIs' dashboards will be obtained.
Response body	dashboardWrapper	DashboardWrapper	It contains a parameter called urls, a List<UrlWrapper> where the UrlWrapper contains a String with the url of a metric/KPI.
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST.			

**Table 24: REST API – DELETE /portal/dcs/dashboard**

<b>DELETE /portal/dcs/dashboard</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	dcsRequestWrapper	DcsRequestWrapper	The same as in the POST request.
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED.			
Error response HTTP code: 400 BAD REQUEST.			

**Table 25: REST API – POST /portal/dcs/start\_signalling**

<b>POST /portal/dcs/start_signalling</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	--	--	--
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 201 CREATED.			
Error response HTTP code: 400 BAD REQUEST.			

**Table 26: REST API – DELETE /portal/dcs/stop\_signalling**

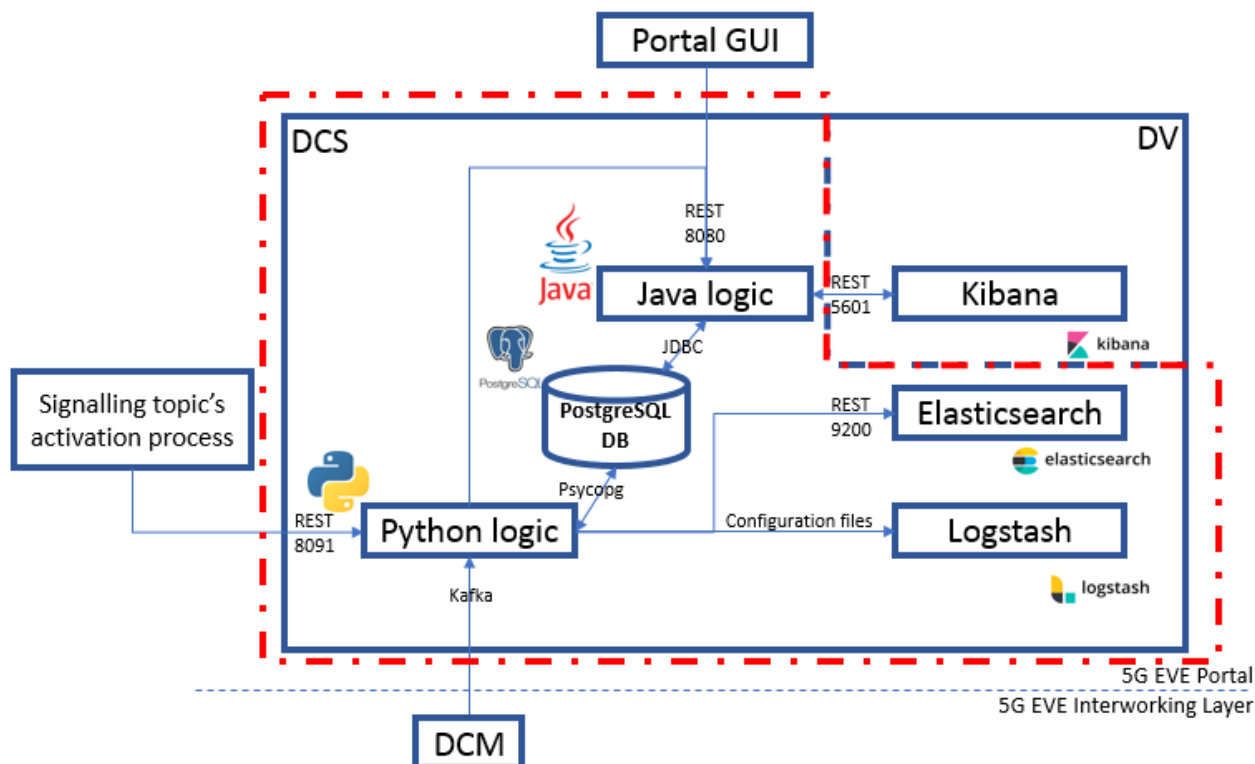
<b>DELETE /portal/dcs/stop_signalling</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body	--	--	--
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 201 CREATED.			

<sup>3</sup> Example of these messages were described in D4.3 [3], Annex J. However, as there have been changes in the content of these messages, an update is provided in the Annex A of this deliverable D4.4.

Error response HTTP code: 400 BAD REQUEST.

### 2.1.3.2 Implementation updates

There are several important updates regarding the architecture in which the DCS has been implemented. D4.2 [2] only described the components from the Elastic Stack that compose the DCS (i.e. Logstash and Elasticsearch). However, in order to introduce the DCS in the automated E2E workflow, some other functionalities were needed, resulting in the architecture presented in Figure 1.

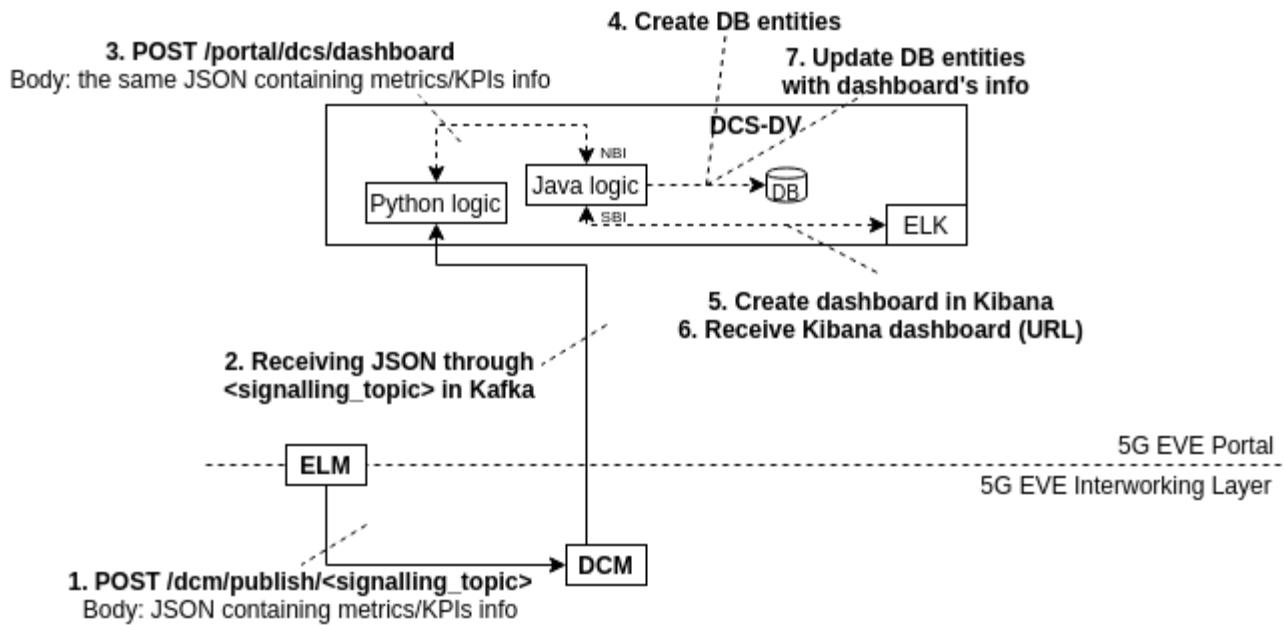


**Figure 1: Updated architecture of the DCS**

Now, the DCS also contains two small modules integrated for specific functionalities to be offered for allowing the automation of the whole E2E workflow. These are the following:

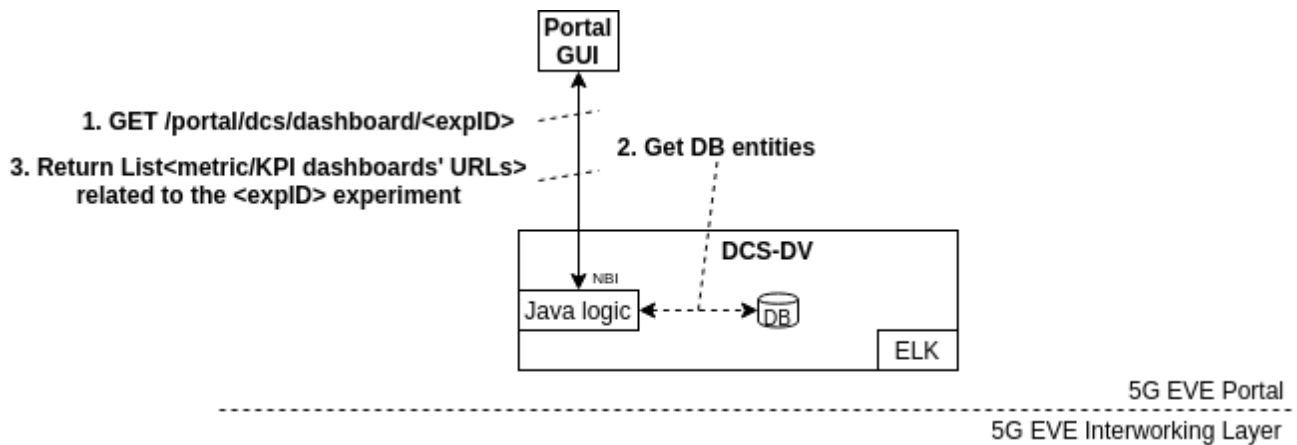
- The management of Kibana dashboards through a Java logic<sup>4</sup> that serves a REST API, handling the creation, retrieval and removal of dashboards in Kibana (Data Visualization component from the GUI). Basically, this application takes the information related to each metric and KPI and generates the corresponding Kibana dashboard by interacting with the Kibana REST API, generating an URL that can be embedded directly in the Portal GUI. Then, each operation is triggered at a specific time:
  - The dashboard creation of a given metric or KPI is performed when the first message containing a value of that metric or KPI arrives to the DCS through the corresponding Kafka topic. The workflow followed is completely described in the Figure 2.

<sup>4</sup> <https://github.com/5GEVE/5geve-wp4-dcs-kibana-dashboards-handler>



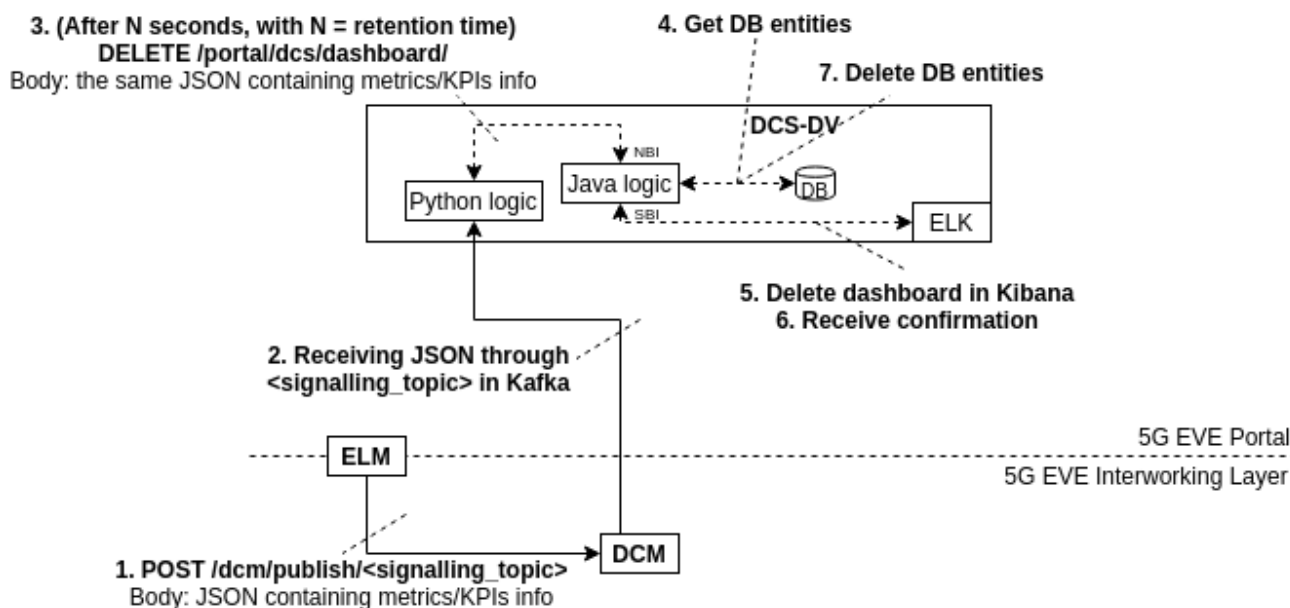
**Figure 2: DCS: Dashboard creation workflow**

- The retrieval of Kibana dashboards is performed by the Portal GUI when the experimenter enters in the Dashboards' visualization page. The workflow followed in this operation can be checked in Figure 3.



**Figure 3: DCS: Dashboard retrieval workflow**

- The removal of dashboards is performed automatically after a specific amount of time once the experiment has terminated, so that the experimenter can check the dashboards and the related data during that retention time after the experiment is finished. The workflow for this operation is similar to the dashboard creation and can be seen in Figure 4:



**Figure 4: DCS: Dashboard deletion workflow**

- The management of the signalling topics through a Python logic<sup>5</sup> implementing a REST API with Flask, enabling and disabling these signalling topics that allows the DCS to receive the topics and related information to properly configure the Monitoring platform. Depending on the operation<sup>6</sup>, there are different interactions between this Python logic and other components of the DCS:
  - When the signalling topics are started and the ELM publishes the topics to be used during the experiments in the DCM, this Python logic receives that data. Then, the Logstash pipelines (i.e. the Kafka subscribers for each topic in the experiment) are configured by modifying specific configuration files, and also the Elasticsearch indexes (that are used by Kibana to display the data collected, afterwards) are created by interacting with the Elasticsearch REST API. Finally, a Kafka consumer is created for each topic, waiting for the first message in that topic to trigger the creation of the Kibana dashboards, as explained before. When the experiment finishes, it is also notified through the signalling topics, then this Python logic removes the Logstash pipelines and triggers a countdown to remove both the Elasticsearch indexes and the Kibana dashboards after passing the retention time specified (which will be between 7 and 14 days).
  - The signalling topics are stopped only when no more experiments will be executed. Then, the signalling topics installed in Kafka are removed in the DCM and the Monitoring system stops listening messages on these signalling topics.

Additionally, a PostgreSQL database is also deployed in the DCS to server both modules aforementioned. The Python logic only uses the database for saving the IDs of the Logstash pipelines, just in case of failure in the system. Regarding the Java logic, the database contains the data model used for managing the Kibana dashboards, which is presented in Figure 5.

<sup>5</sup> <https://github.com/5GEVE/5geve-wp4-dcs-signalling-topic-handler>

<sup>6</sup> The operations presented are related to the Monitoring and Data Collection process, already presented in previous WP3 and WP4 deliverables. However, in D3.4 [4], the updated workflows for this process have been included, just to centralize the description of the Monitoring and Data Collection Platform in the Interworking Layer.

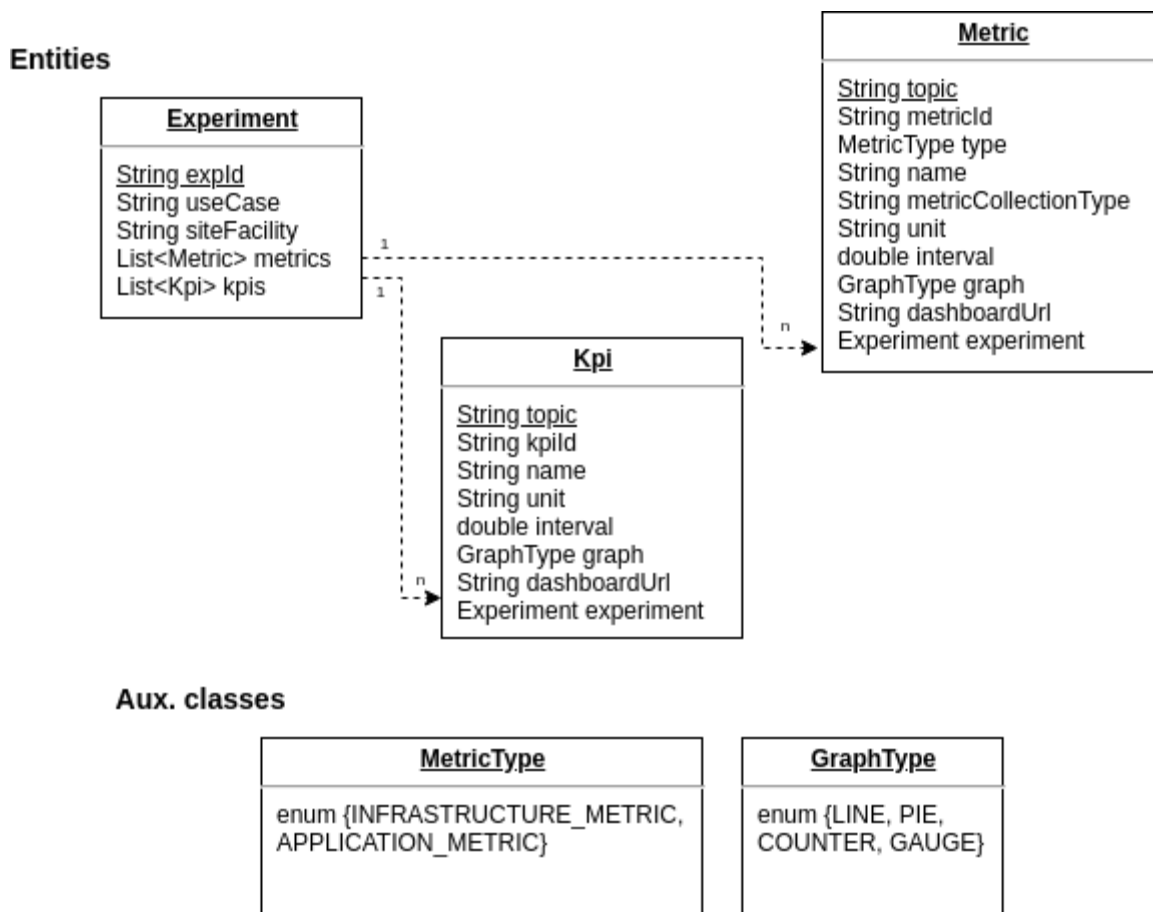


Figure 5: DCS Java logic data model

The deployment of the DCS, together with the DV, has been done with a specific Ansible playbook that configures an Ubuntu Server 16.04 LTS from scratch, provisioning all the packages, files and tools needed to integrate the Monitoring platform in 5G EVE. This Ansible playbook can be found here: <https://github.com/5GEVE/5geve-wp4-dcs-dv-deployment>.

Finally, regarding the interaction with Keycloak and the RBAC, some advances have been achieved by integrating a keycloak-kibana plugin<sup>7</sup> in the DCS-DV. However, this is still under testing and it is not fully integrated in the E2E workflow, although this issue does not affect it. This integration is intended to be reported in the next WP4 deliverable.

### 2.1.4 File Storage

The File Storage module is intended to be used by other modules to upload large files, similar to a cloud storage system. The main user of this module is the VNF Storage component at the Portal GUI layer, which is used by VNF providers to upload VNF packages, so the site managers selected in this process will receive a ticket requesting the onboarding of the selected packages in their sites. Thus, the File Storage module after a successful uploading of the file, needs to create as many tickets as the sites selected by the VNF provider. These tickets should include the link to the uploaded package, so that site managers can download such file.

<sup>7</sup> <https://github.com/novomatic-tech/keycloak-kibana>

## 2.1.4.1 Updated File Storage Interface (Ifs)

**Table 27: File Storage REST API**

HTTP method	URI	Description	Public
GET	/portal/fs/	Retrieves files associated to a user. If user has SiteManager role, he/she will retrieve all available files to be uploaded to their corresponding sites. It will return all files uploaded by the logged in user otherwise	Yes
POST	/portal/fs/upload/{filename}	Used to upload a single zip file and a list of one or more sites that have to be notified.	Yes
POST	/portal/fs/sites/{filename}	Allows the assignation of deployment sites to an already uploaded file	Yes
POST	/portal/fs/sites/{filename}	Used to update the status of an uploaded file at the site, allowing the site manager to notify the user as soon as the file is deployed	Yes
GET	/portal/fs/download/{filename}	Requests to download the file identified with a name.	Yes
POST	/portal/fs/delete/{filename}	Removes the upload notification created at during the uploading step.	Yes

**Table 28: REST API - GET /portal/fs**

GET /portal/fs/			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 29: REST API - POST /portal/fs/upload/{filename}**

POST /portal/fs/upload/{filename}			
	Parameter name	Parameter type	Description
Request body (form-data)	file	File	File to be uploaded
URI Variables	Filename	String	Name of the file to be uploaded
Response body (JSON)	details	String	Details of the operation
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 404 NOT FOUND, 401 UNAUTHORIZED, 500 INTERNAL ERROR			

**Table 30: REST API - POST /portal/fs/sites/{filename}**

POST /portal/fs/sites/{filename}			
	Parameter name	Parameter type	Description
Request body (JSON)	sites	Array<String>	List of sites where the uploaded file should be deployed

URI Variables	filename	String	Name of the file that will be deployed on the different sites
Response body (JSON)	details	Array<String>	List of sites where deployment operation was correctly requested
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND			

**Table 31: REST API - POST /portal/fs/status/{filename}**

<b>POST /portal/fs/status/{filename}</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	site	String	Site where the status of the file has changed
	status	String	Status at which the file should be updated at the specific site
URI Variables	filename	String	Name of the file
Response body (JSON)	details	String	Details of the operation
Successful response HTTP code: 200 OK			

**Table 32: REST API - POST /portal/fs/download/{filename}**

<b>GET /portal/fs/download/{filename}</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)			
URI Variables	filename	String	Name of the file
Response body (JSON)	details	String	Details of the operation
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND			

**Table 33: REST API - POST /portal/fs/delete/{filename}**

<b>POST /portal/fs/delete/{filename}</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	site	String	Site where we want to remove the file
URI Variables	filename	String	Name of the file
Response body (JSON)	details	String	Details of the operation
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND			

### 2.1.4.2 Implementation updates

We have implemented all the functionality required for completely manage the lifecycle of the uploaded file. Starting from the uploading process, the uploaded files are securely stored in a cloud environment, being the user able to request the deployment on the different site facilities. Then, the site managers will be notified, and they will be able to download the uploaded files in order to include them at their infrastructures. Finally, the site manager will notify the user that the uploaded file is ready to be used.

All this functionality has been implemented for large files management, including mechanisms to properly upload and download files with those characteristics.

## 2.1.5 Catalogue Service

The Portal Catalogue provides services to store all the blueprints and descriptors associated to the definition of a 5G EVE experiment. In particular, it manages the information elements for Vertical Services, experiment execution contexts, test cases and experiments. Its functionalities have been fully documented in deliverable D4.1 [1], Section 5.

### 2.1.5.1 Updated Catalogue Service Interface (Ics)

**Table 34: 5G-EVE Portal Catalogue REST API**

HTTP method	URI	Description	Public
GET	/portal/catalogue/ctxblueprint<?[filter_parameters]>	Get the list of the IDs of all the context blueprints matching the filter criteria provided in the URI query parameters. If the filter is not specified, the IDs of all the existing context blueprints are returned.	Yes
POST	/portal/catalogue/ctxblueprint	Create and onboard a new context blueprint. The context blueprint is onboarded together with the associated NSD (if present in the request) and translation rules.	No
GET	/portal/catalogue/ctxblueprint/<id>	Get the details of the context blueprint with ID <id>.	Yes
DELETE	/portal/catalogue/ctxblueprint/<id>	Remove the context blueprint with ID <id>.	No
GET	/portal/catalogue/vsblueprint<?[filter_parameters]>	Get the list of the IDs of all the vertical service blueprints matching the filter criteria provided in the URI query parameters. If the filter is not specified, the IDs of all the existing vertical service blueprints are returned.	Yes
POST	/portal/catalogue/vsblueprint	Create and onboard a new vertical service blueprint. The vertical service blueprint is onboarded together with the associated NSD (if present in the request) and translation rules.	No
GET	/portal/catalogue/vsblueprint/<id>	Get the details of the vertical service blueprint with ID <id>.	Yes
DELETE	/portal/catalogue/vsblueprint/<id>	Remove the vertical service blueprint with ID <id>.	No
GET	/portal/catalogue/expblueprint<?[filter_parameters]>	Get the list of the IDs of all the experiment blueprints matching the filter criteria provided in the URI query parameters. If the filter is not specified, the IDs of all the existing experiment blueprints are returned.	Yes
POST	/portal/catalogue/expblueprint	Create and onboard a new experiment blueprint. The experiment blueprint is onboarded together with the associated NSD (if present in the request) and translation rules.	No
GET	/portal/catalogue/expblueprint/<id>	Get the details of the experiment blueprint with ID <id>.	Yes
DELETE	/portal/catalogue/expblueprint/<id>	Remove the experiment blueprint with ID <id>.	No
POST	/portal/catalogue/expdescriptor	Create and onboard a new experiment descriptor. All the associated VSD, CtxD and TCD are automatically created and onboarded.	Yes



GET	/portal/catalogue/expdescriptor	Get the details of all the experiment descriptors.	Yes
GET	/portal/catalogue/expdescriptor/<id>	Get the details of the experiment descriptor with ID <id>.	Yes
DELETE	/portal/catalogue/expdescriptor/<id>	Remove the experiment descriptor with ID <id>.	Yes

The OpenAPI of the 5G EVE Portal Catalogue is available at the following link:

<https://github.com/nextworks-it/slicer-catalogue/blob/5geve-release/API/EveBlueprintCatalogue.json>

**Table 35: REST API – GET /portal/catalogue/ctxblueprint{?[filter\_parameters]}**

GET /portal/catalogue/ctxblueprint{?[filter_parameters]}			
URI Query Parameters	ContextType	String	Type of context execution
	Site	String	5G EVE site where the context execution if available
	VsbId	String	ID of the VSB which the queried context blueprints are compatible to.
Response body	ContextID	List<String>	List of IDs of the contexts matching the filter provided in the URI Query parameters.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 500 INTERNAL ERROR			

**Table 36: REST API – POST /portal/catalogue/ctxblueprint**

POST /portal/catalogue/ctxblueprint			
Request body	ctxBlueprint	ContextBlueprint	Content of the context blueprint to be onboarded.
	nsds	List<Nsd>	NSD of the Network Services required to deploy the context execution elements.
	translationRules	List<TranslationRule>	Rules regulating the translation between a context descriptor based on the given blueprint and the associated network service.
Response body	n/a	n/a	n/a
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 37: REST API – GET /portal/catalogue/ctxblueprint/<id>**

GET /portal/catalogue/ctxblueprint/<id>			
URI Query Parameters	id	String	Unique ID of the queried context blueprint.
Response body	blueprintId	String	Unique ID of the context blueprint.
	version	String	Version of the context blueprint.
	name	String	Name of the context blueprint.
	ctxBlueprint	ContextBlueprint	Content of the context blueprint.

	onBoardedNsdInfoId	List<String>	IDs of the onboarded NSD associated to the given context blueprint.
	activeCtxdId	List<String>	IDs of the context descriptors associated to the given blueprint.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 38: REST API – DELETE /portal/catalogue/ctxblueprint/<id>**

<b>DELETE /portal/catalogue/ctxblueprint/&lt;id&gt;</b>			
URI Query Parameters	id	String	Unique ID of the context blueprint to be deleted.
Response body	n/a	n/a	n/a
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 39: REST API – GET /portal/catalogue/vsblueprint<?[filter\_parameters]>**

<b>GET /portal/catalogue/vsblueprint&lt;?[filter_parameters]&gt;</b>			
URI Query Parameters	Site	String	5G EVE site where the vertical service can be deployed
Response body	VsbID	List<String>	List of IDs of the vertical service blueprints matching the filter provided in the URI Query parameters.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 500 INTERNAL ERROR			

**Table 40: REST API – POST /portal/catalogue/vsblueprint**

<b>POST /portal/catalogue/vsblueprint</b>			
Request body	vsBlueprint	VerticalServiceBlueprint	Content of the vertical service blueprint to be onboarded.
	nsds	List<Nsd>	NSD of the Network Services required to deploy the vertical service elements.
	translationRules	List<TranslationRule>	Rules regulating the translation between a vertical service descriptor based on the given blueprint and the associated network service.
Response body	n/a	n/a	n/a
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 41: REST API – GET /portal/catalogue/vsblueprint/<id>**

<b>GET /portal/catalogue/vsblueprint/&lt;id&gt;</b>			
URI Query Parameters	id	String	Unique ID of the queried vertical service blueprint.
Response body	blueprintId	String	Unique ID of the vertical service blueprint.
	version	String	Version of the vertical service blueprint.
	name	String	Name of the vertical service blueprint.
	vsBlueprint	VerticalServiceBlueprint	Content of the vertical service blueprint.
	onBoardedNsdInfoId	List<String>	IDs of the onboarded NSDs associated to the given vertical service blueprint.
	activeVsdId	List<String>	IDs of the vertical service descriptors associated to the given blueprint.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 42: REST API – DELETE /portal/catalogue/vsblueprint/<id>**

<b>DELETE /portal/catalogue/vsblueprint/&lt;id&gt;</b>			
URI Query Parameters	id	String	Unique ID of the vertical service blueprint to be deleted.
Response body	n/a	n/a	n/a
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 43: REST API – GET /portal/catalogue/expblueprint<?[filter\_parameters]>**

<b>GET /portal/catalogue/expblueprint&lt;?[filter_parameters]&gt;</b>			
URI Query Parameters	Site	String	5G EVE site where the experiment can be deployed
	VsbId	String	ID of the VSB which the queried experiments blueprints are associated to.
Response body	ExpbID	List<String>	List of IDs of the experiment blueprints matching the filter provided in the URI Query parameters.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 500 INTERNAL ERROR			

**Table 44: REST API – POST /portal/catalogue/expblueprint**

POST /portal/catalogue/expblueprint			
Request body	experimentBlueprint	ExperimentBlueprint	Content of the experiment blueprint to be onboarded.
	nsds	List<Nsd>	NSD of the Network Services required to deploy the experiment elements.
	translationRules	List<TranslationRule>	Rules regulating the translation between an experiment descriptor based on the given blueprint and the associated network service.
Response body	n/a	n/a	n/a
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 45: REST API – GET /portal/catalogue/expblueprint/<id>**

GET /portal/catalogue/expblueprint/<id>			
URI Query Parameters	id	String	Unique ID of the queried experiment blueprint.
Response body	blueprintId	String	Unique ID of the experiment blueprint.
	version	String	Version of the experiment blueprint.
	name	String	Name of the experiment blueprint.
	experimentBlueprint	ExperimentBlueprint	Content of the experiment blueprint.
	activeExpdId	List<String>	IDs of the experiment descriptors associated to the given blueprint.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 46: REST API – DELETE /portal/catalogue/expblueprint/<id>**

DELETE /portal/catalogue/expblueprint/<id>			
URI Query Parameters	id	String	Unique ID of the experiment blueprint to be deleted.
Response body	n/a	n/a	n/a
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 47: REST API – GET /portal/catalogue/expdescriptor<?[filter\_parameters]>**

GET /portal/catalogue/expdescriptor			
URI Query Parameters	--	--	--
Response body	expdID	List<String>	List of IDs of the available experiment descriptors.

Successful response HTTP code: 200 OK
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 500 INTERNAL ERROR

**Table 48: REST API – POST /portal/catalogue/expdescriptor**

<b>POST /portal/catalogue/expdescriptor</b>			
Request body	name	String	Name of the experiment descriptor
	version	String	Version of the experiment descriptor
	experimentBlueprintId	String	ID of the associated experiment blueprint
	tenantId	String	ID of the tenant who is creating the experiment descriptor
	isPublic	Boolean	True if the experiment descriptor is public
	vsDescriptor	VsDescriptor	Descriptor of the associated vertical service
	contextDetails	List<BlueprintUserInformation>	Information provided by the experimenter about the configuration of the contexts associated to the experiment
	testCaseConfiguration	List<BlueprintUserInformation>	Information provided by the experimenter about the configuration of the test cases associated to the experiment
	kpiThresholds	Map<String,String>	Thresholds to evaluate the KPIs.
Response body	n/a	n/a	n/a
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 409 CONFLICT, 500 INTERNAL ERROR			

**Table 49: REST API – GET /portal/catalogue/expdescriptor/<id>**

<b>GET /portal/catalogue/expdescriptor/&lt;id&gt;</b>			
URI Query Parameters	id	String	Unique ID of the queried experiment blueprint.
Response body	expDescriptor	ExperimentDescriptor	Content of the experiment descriptor.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

**Table 50: REST API – DELETE /portal/catalogue/expdescriptor/<id>**

<b>DELETE /portal/catalogue/expdescriptor/&lt;id&gt;</b>			
URI Query Parameters	id	String	Unique ID of the experiment descriptor to be deleted.
Response body	n/a	n/a	n/a
Successful response HTTP code: 204 NO CONTENT			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

## 2.1.5.2 Implementation updates

Implementation details can be found in 5G EVE deliverables D4.1 [1] and D4.2 [2].

## 2.1.6 Ticketing System Backend

The Ticketing System Backend (TSB) component offers a service to other modules at the backend and to the Ticketing GUI module through the *Itsb* interface to create, comment and delete tickets. The main goal of this module is to notify events like errors, experiment requests, VNF onboarding requests, etc. to the corresponding actors who have to manage such events. The design of this module has been already presented in D4.1 [1], section 4.2.

### 2.1.6.1 Updated Ticketing System Backend Interface (Itsb)

**Table 51: Ticketing System backend REST API**

HTTP method	URI	Description	Public
GET	/portal/tsb/products	Find available products that are susceptible for ticket association	Yes
GET	/portal/tsb/components	Find available components that are susceptible for ticket association	Yes
GET	/portal/tsb/adminusers	Find admin users to whom tickets can be assigned	Yes
POST	/portal/tsb/tickets	Add a new ticket	Yes
GET	/portal/tsb/tickets	Find tickets susceptible to be checked by the user	Yes
POST	/portal/tsb/tickets/trusted	Add a new ticket from a trusted module without authorization mechanisms	No
GET	/portal/tsb/tickets/{ticket_id}	Retrieve specific information of a ticket	Yes
GET	/portal/tsb/tickets/{ticket_id}/comments	Retrieve comments associated to a ticket	Yes
POST	/portal/tsb/tickets/{ticket_id}/comments	Create a new comment at a specific ticket	Yes
POST	/portal/tsb/tickets/{ticket_id}/comments/trusted	Create a new comment at a specific ticket from a trusted module without authorization mechanisms	No

**Table 52: REST API – GET /portal/tsb/products**

GET /portal/tsb/products			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	--	--	--
Response body (JSON)	details	Array<String>	List of available product names
Successful response HTTP code: 200 OK			

**Table 53: REST API - GET /portal/tsb/components**

GET /portal/tsb/components			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	--	--	--
Response body (JSON)	details	Array<String>	List of available components

Successful response HTTP code: 200 OK
Error response HTTP code: 401 UNAUTHORIZED

**Table 54: REST API - GET /portal/tsb/adminusers**

GET /portal/tsb/adminusers			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	--	--	--
Response body (JSON)	Details	Array<String>	List of admin users susceptible of tickets association
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED			

**Table 55: REST API -POST /portal/tsb/tickets**

POST /portal/tsb/tickets			
	Parameter name	Parameter type	Description
Request body (JSON)	product	String	Product name at which the ticket will be created
	component	String	Component name at which the ticket will be created
	summary	String	Summary of the ticket to be created
	description	String	Description of the ticket
	assigned_to (optional)	email	Email address of the user in charge of managing the ticket
URI Variables	--	--	--
Response body	Details	String	ID of the newly created ticket
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 56: REST API -GET /portal/tsb/tickets**

GET /portal/tsb/tickets			
	Parameter name	Parameter type	Description
Request body (JSON)	--	--	--
URI Variables	--	--	--
Response body (JSON)	Details	Array<tickets>	List of tickets
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED			

**Table 57: REST API -GET /portal/tsb/tickets/trusted**

POST /portal/tsb/tickets/trusted			
	Parameter name	Parameter type	Description
Request body (JSON)	product	String	Product name at which the ticket will be created
	component	String	Component name at which the ticket will be created
	summary	String	Summary of the ticket to be created
	description	String	Description of the ticket

	assigned_to (optional)	email	Email address of the user in charge of managing the ticket
URI Variables	--	--	--
Response body	Details	String	ID of the newly created ticket
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 58: REST API -GET /portal/tsb/tickets/{ticket\_id}**

<b>GET /portal/tsb/tickets/{ticket_id}</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	--	--	--
URI Variables	ticket_id	String	Id of the ticket to be retrieved
Response body (JSON)	Details	ticket	Requested ticket
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED			

**Table 59: REST API -GET /portal/tsb/tickets/{ticket\_id}/comments**

<b>GET /portal/tsb/tickets/{ticket_id}/comments</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	--	--	--
URI Variables	ticket_id	String	Id of the ticket whose comments we want to retrieve
Response body (JSON)	Details	Array<comments>	List of comments associated to the specific ticket
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED			

**Table 60: REST API -POST /portal/tsb/tickets/{ticket\_id}/comments**

<b>POST /portal/tsb/tickets/{ticket_id}/comments</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	comment	String	Comment to be created at the ticket
URI Variables	ticket_id	String	Id of the ticket whose comments we want to retrieve
Response body (JSON)	Details	String	Id of the newly created comment
Successful response HTTP code: 200 OK			
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST			

**Table 61: REST API -POST /portal/tsb/tickets/{ticket\_id}/comments/trusted**

<b>POST /portal/tsb/tickets/{ticket_id}/comments/trusted</b>			
	<b>Parameter name</b>	<b>Parameter type</b>	<b>Description</b>
Request body (JSON)	comment	String	Comment to be created at the ticket
URI Variables	ticket_id	String	Id of the ticket whose comments we want to retrieve
Response body (JSON)	Details	String	Id of the newly created comment



Successful response HTTP code: 200 OK
Error response HTTP code: 401 UNAUTHORIZED, 400 BAD REQUEST

### 2.1.6.2 Implementation updates

We have included additional endpoints for communication with other backend modules. This functionality enables a correct workflow of notifications through the ticketing system when working with experiments.

Moreover, we have improved how tickets are managed to provide a more complete and flexible set of tools to other components such as the portal GUI.

## 2.2 Portal GUI

In the following sections we present the components present in the Portal graphical user interface sublayer, which are the ones implemented to provide a graphical tool to all actors to perform all phases involved in designing, defining, executing and analysing all results.

### 2.2.1 VNF Storage

A VNF Storage section has been included at the GUI side. First, we have the main view where all the uploaded files (or files to be deployed at the site facilities if the user is a site manager) are shown on a table. This table contains all the information about the file, such as name, site where the file should be deployed, and status. Moreover, each row representing a file contains its own control buttons to trigger different actions (delete, change status, etc.)

Furthermore, we have also included a form allowing users to upload their files, shown in Figure 6. It contains a basic form to select the file to be uploaded, the name of the file (that will be used to store it) and the sites where the file should be deployed. Once the upload button is pressed, a progress bar appears indicating how fast our file is being uploaded. There is also a service to list all files uploaded, shown in Figure 7.

Finally, we have included the download functionality, where the download progress is shown similarly via a small progress bar.

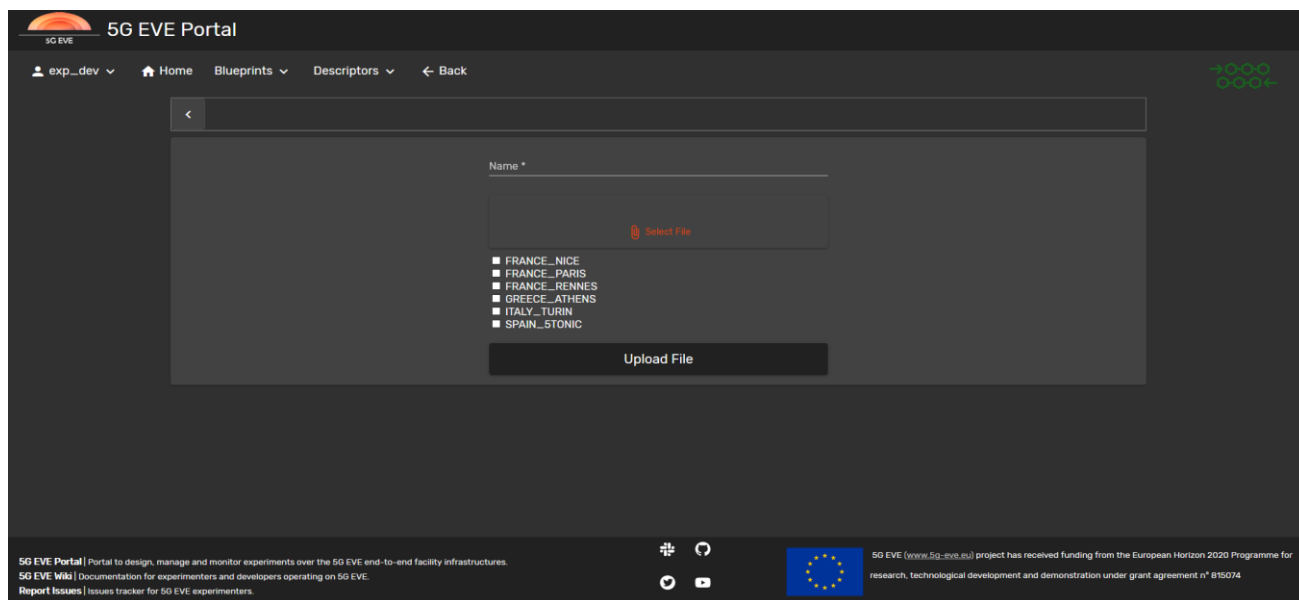


Figure 6: Interface to upload VNF packages to a selected number of sites.

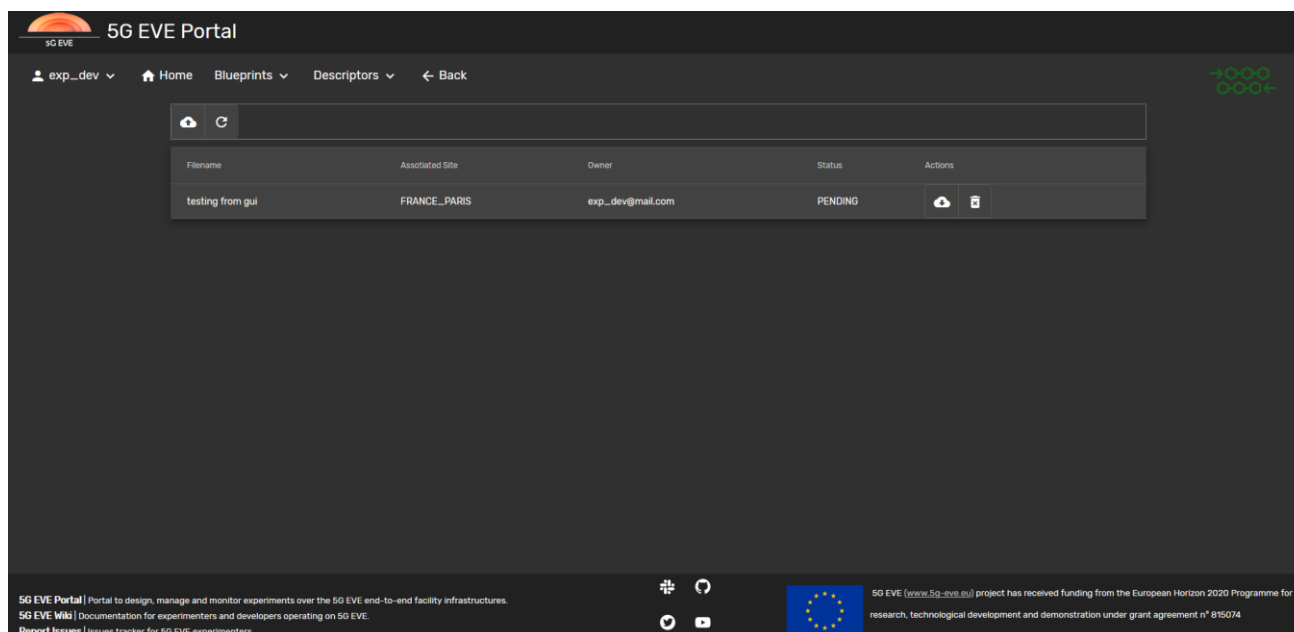


Figure 7: Interface to list all VNF packages uploaded.

### 2.2.2 Sign-up/Login

This component has not been updated, so all implementations detail can be found in 5G EVE deliverable D4.2 [2].

### 2.2.3 Experiment Blueprint Builder

The Experiment Blueprint Builder has been updated according to the latest version of the information models of the blueprints and descriptors. Moreover, it has been improved with features that facilitate the on-boarding procedures, as follows:

- integrated mechanisms for the early validation of the items uploaded by the user, e.g. to validate the format of a Vertical Service Blueprint or the content of the associated NSD, providing a feedback on the potential mistakes;
- procedure to allow the user to easily skip the definition of optional parameters, like context blueprints, NSD or translation rules. In particular, the platform has been updated with the capability to handle experiments that are not associated to any network service to be deployed in the 5G EVE facility;
- simplified declaration of translation rules, with immediate identification of the service parameters that the user needs to fill.

### 2.2.4 Data Visualization

Although the DCS-DV architecture has been already presented and commented in Section 2.1.3.2, it will be presented again in Figure 8 highlighting the components that belong to this module in the 5G EVE Portal, and which is, uniquely, Kibana.

The main feature provided in this new release of the 5G EVE Portal is the integration of the Kibana dashboards in the Portal GUI thanks to the Java module included in the DCS (already commented in Section 2.1.3.2), providing the experimenter with a single point to follow the status of the experiment.

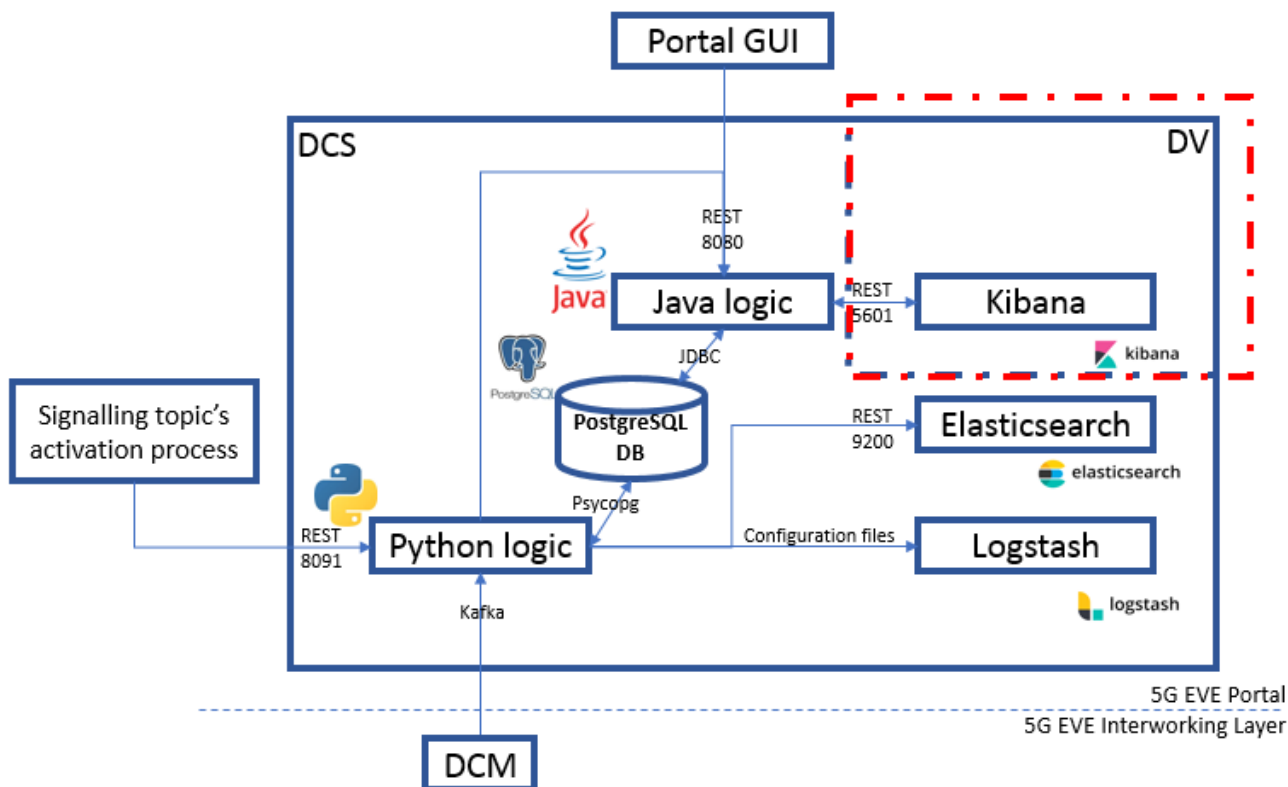


Figure 8: Updated architecture of the DV

In Figure 9, a screenshot of the Experiment Metrics Dashboards for a test performed during the testing of an experiment execution is presented, showing two dashboards: on the left, a pie graph related to a monitored KPI, and on the right, a line graph related to a monitored application metric.

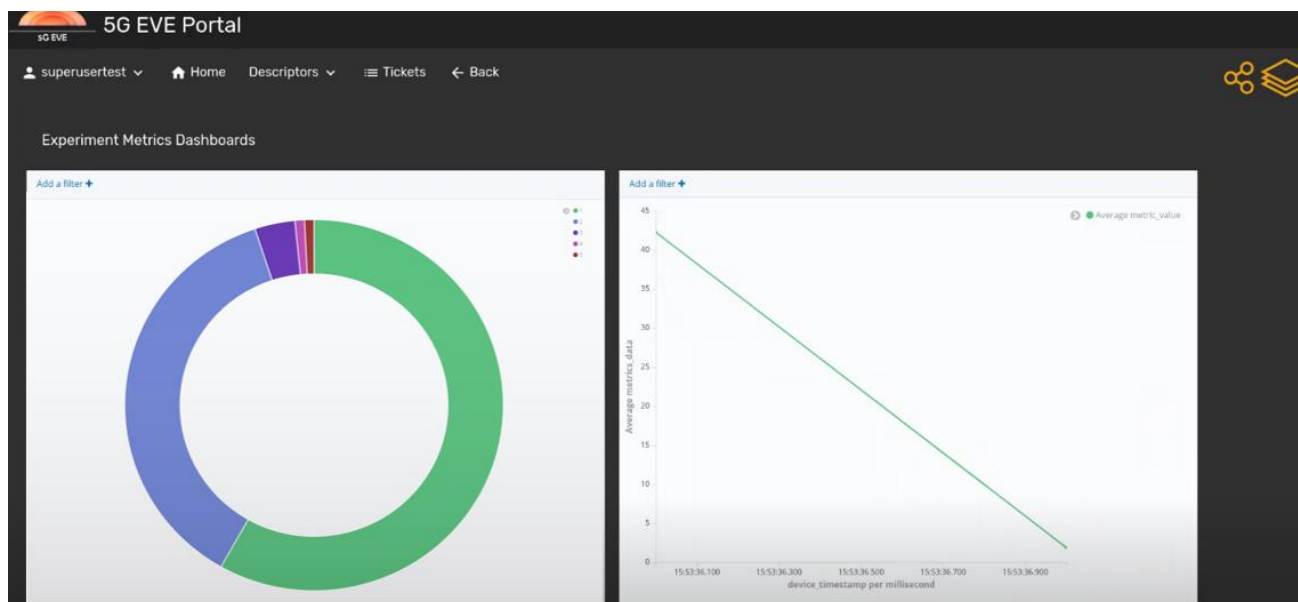


Figure 9: Screenshot of the Portal GUI containing several Kibana dashboards

In order to homogenise the format of messages received in the Elastic Stack to be displayed afterwards in Kibana, an Information model to publish the metrics’ and KPIs’ values in the DCM has been proposed in D3.4 [4]. The fields from the message that are currently used by the Kibana dashboards are both the *metric\_value* or the *kpi\_value* (depending on if the value is related to a metric or a KPI) and the *timestamp* generated by the device.

Note that, from the experimenter’s side, Kibana resources can be only accessed through the Experiment Metrics Dashboard page in the Portal GUI, so the access to the Kibana GUI is only reserved for management purposes and directly handled by the Portal administrators.

Finally, in the same way as with the DCS, the deployment of the DV, together with the DCS, has been done with a specific Ansible playbook that configures an Ubuntu Server 16.04 LTS from scratch, provisioning all the packages, files and tools needed to integrate the Monitoring platform in 5G EVE. This Ansible playbook can be found here: <https://github.com/5GEVE/5geve-wp4-dcs-dv-deployment>.

### 2.2.5 Browse and look-up

The browse and look-up tool on the GUI has been modified to match the latest updates in the information models of the blueprints and descriptors. Moreover, new features for the visualization of VNFD and NSD available in the IWL catalogue have been added in the portal catalogue page. The user can visualize the list of available VNF packages and NSDs (Figure 10 and Figure 13) and, for each element, can visualize the textual definition of the descriptor in YAML format (Figure 11 and Figure 14) and the graphical representation of its elements and their interconnections (Figure 12 and Figure 15).

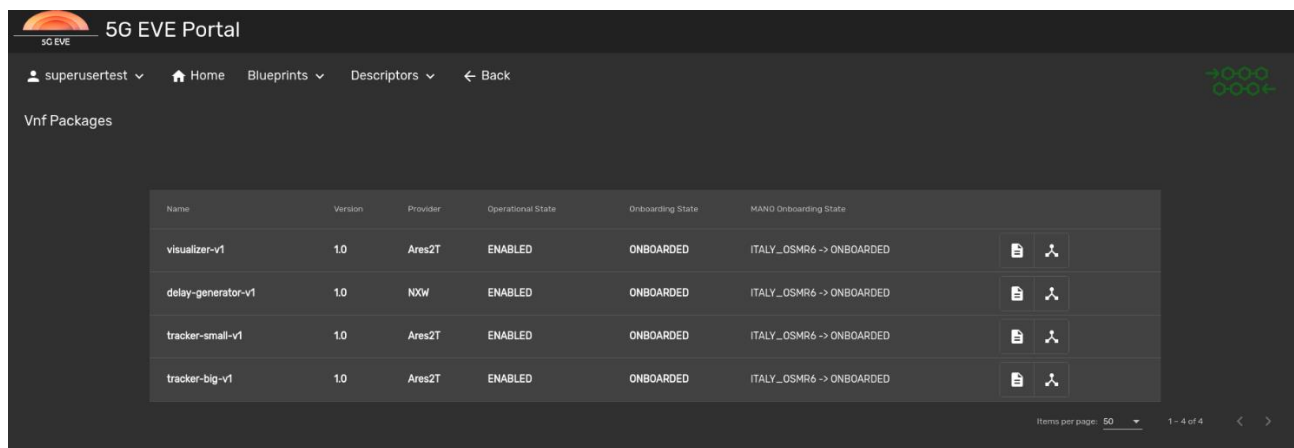


Figure 10: Visualization of VNF Packages - list

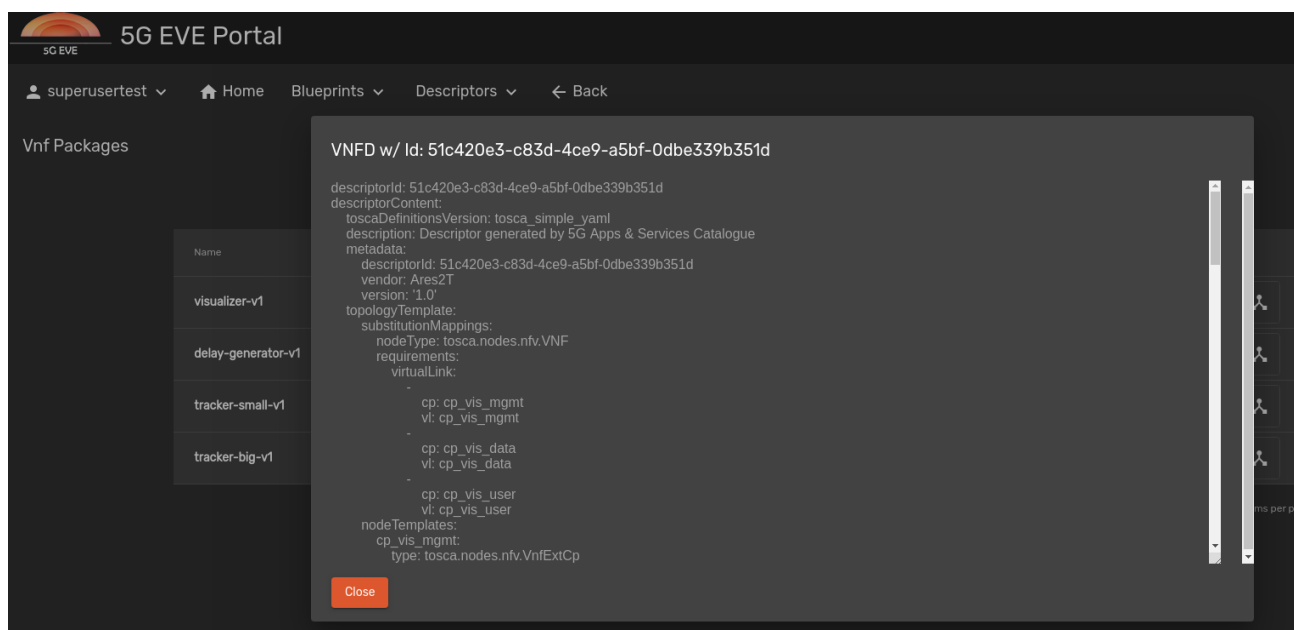


Figure 11: Visualization of VNF Packages – descriptor in textual mode

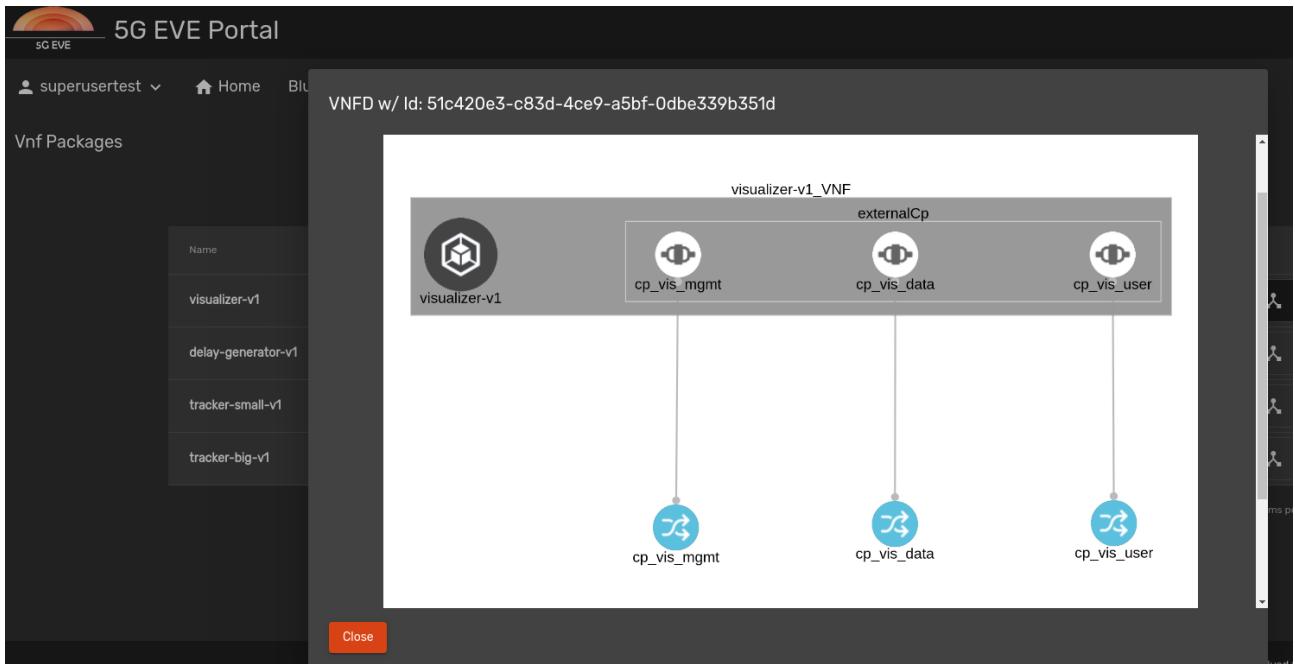


Figure 12: Visualization of VNF Packages – VNFD graph

Name	Version	Provider	Operational State	Onboarding State	MANO Onboarding State
ns_ares2t_tracker_delay_exp_ns_Ares2T_Tracking_Exp_df_ns_ares2t_tracker_exp_il_big	1.0	NXW	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED
ns_ares2t_tracker_delay_exp_ns_Ares2T_Tracking_Exp_df_ns_ares2t_tracker_exp_il_small	1.0	NXW	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED
vsb_polito_smartcity_nsd_vsb_polito_smartcity_df_vsb_polito_smartcity_il_default	1.0	NSD generator	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED
Netem_Delay_Generator_ns_Netem_Delay_Generator_df_ns_Netem_Delay_Generator_il	1.0	NXW	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED
ns_ares2t_tracker_ns_Ares2T_Tracking_df_ns_ares2t_tracker_il_big	1.0	NXW	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED
ns_ares2t_tracker_ns_Ares2T_Tracking_df_ns_ares2t_tracker_il_small	1.0	NXW	ENABLED	ONBOARDED	ITALY_OSMR6 -> ONBOARDED

Figure 13: Visualization of Network Service Descriptors - list

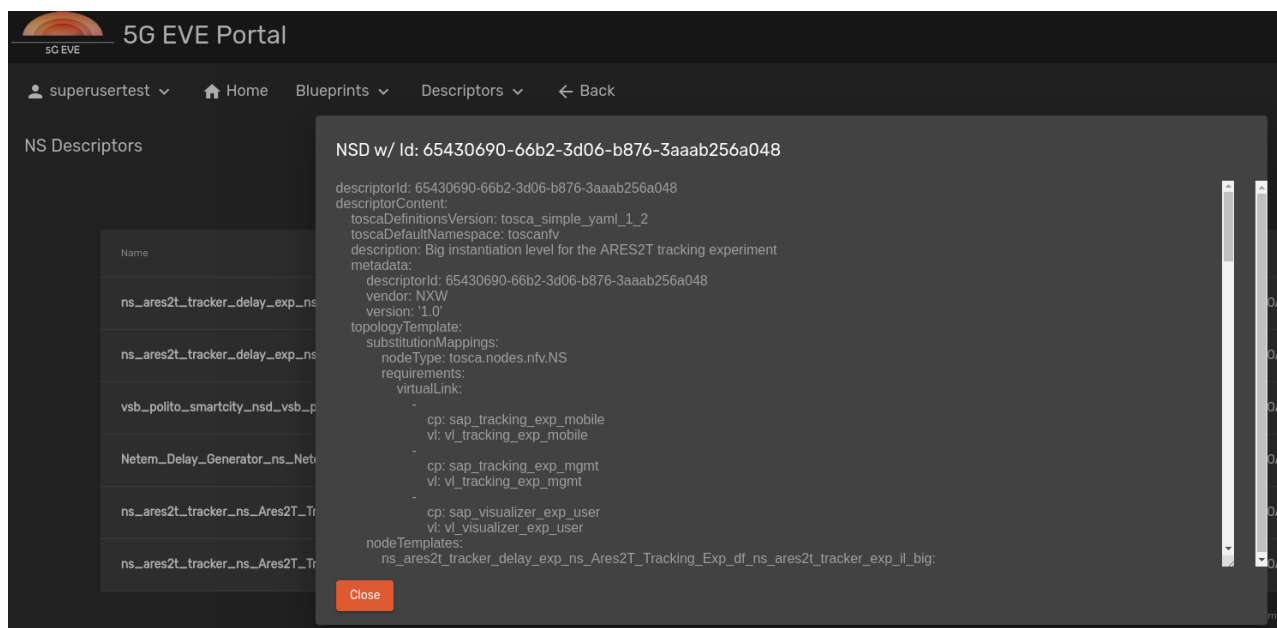


Figure 14: Visualization of Network Service Descriptors – descriptor in textual mode

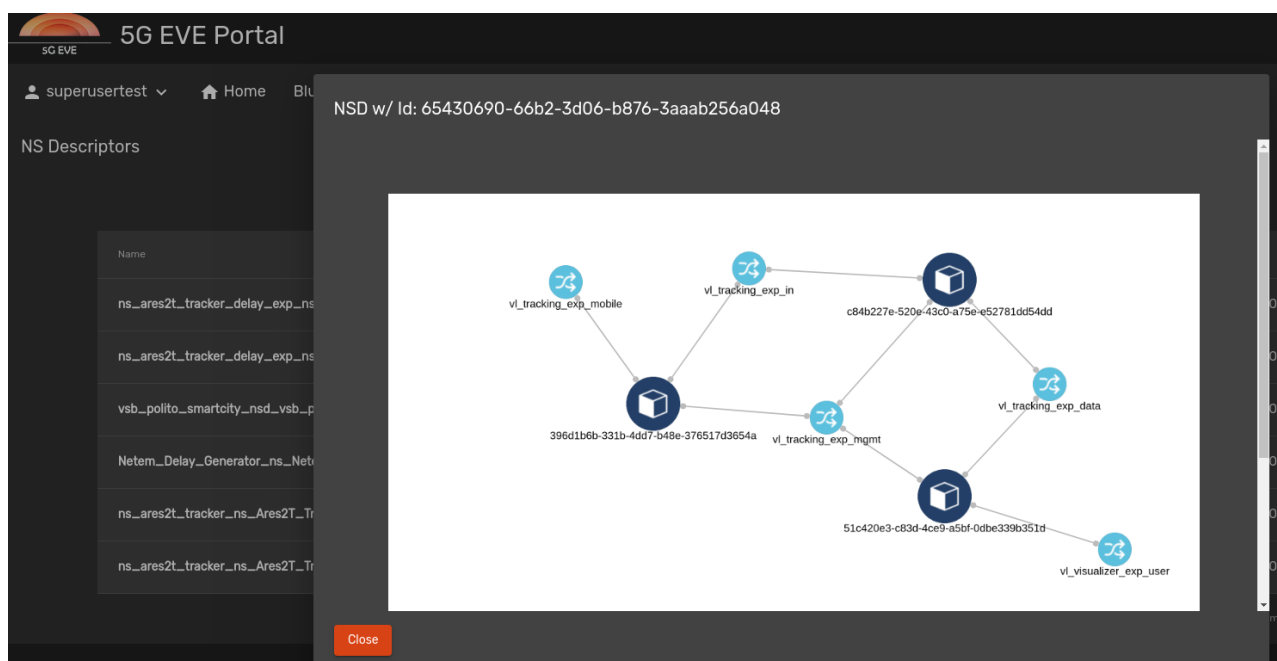


Figure 15: Visualization of Network Service Descriptors – NSD graph

## 2.2.6 Intent-based networking

As the project progresses, the integration of the IBN tool with the rest of the 5G-EVE platform and portal will be further enhanced to provide a smooth experimentation experience, and potentially can be further improved based on early experimenters' feedback.

## 2.3 5G EVE Portal User Guide

In order to be able to use the different 5G EVE portal functionalities, the following steps can be taken:

### 2.3.1 Blueprints Descriptor Preparation

- First, the experiment developer needs to prepare two types of blueprints, i.e., Vertical Service Blueprints (VSB) and Context Blueprints (CtxBs). The number of context blueprints depend on the number of contexts the experiment developer wants to consider in the experiment. For example, in the ASTI use-case we are considering two different types of contexts, i.e., delay and background traffic, and as a result we have two types of CtxBs corresponding to each context.
- For all blueprints, the format used in this manual is YAML, although JSON can be also used.
- For an example on a valid Vertical Service Blueprint, please refer to the following links:
  - Simple vsb: [https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb\\_apache\\_simple/vsb\\_apache\\_simple.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb_apache_simple/vsb_apache_simple.yaml)
  - Advanced vsb: [https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb\\_asti\\_agv/asti\\_agv\\_vsb\\_v2.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb_asti_agv/asti_agv_vsb_v2.yaml)
- Compatible context blueprints corresponding to the above VSB can be found at the following links:
  - Simple Delay\_CtxB [https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_delay/ctx\\_delay\\_apache\\_simple.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_delay/ctx_delay_apache_simple.yaml)
  - Advanced Delay\_CtxB [https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_delay/asti\\_ctx\\_delay.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_delay/asti_ctx_delay.yaml)
  - Advanced Background traffic CtxB [https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_bg\\_traffic/asti\\_ctx\\_bg\\_traffic.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_bg_traffic/asti_ctx_bg_traffic.yaml)
- While preparing the Context Blueprints, the kind of tests to perform within each context needs to be taken into account. These tests will be captured later on in the portal under the “*Test Case Blueprint*” section. We will discuss this further in Section 2.3.4.4.
- After preparing all the blueprint descriptors, the blueprint format needs to be validated by checking the blueprint type (vsb, and ctx) with the 5G EVE blueprint-validator tool. Detailed instructions on how to install and use the blueprint validator can be found at the following link:
  - <https://github.com/5GEVE/blueprint-validator>
- Once the blueprint formats have been validated and they are all OK, then we move on to next step.

### 2.3.2 PREPARE NETWORK SERVICE DESCRIPTORS (NSD)

For each of the blueprints in Step 1 above, the experiment developer has to create the corresponding Network Service Descriptors in NFV-IFA014 format. In 5G EVE, all the NSD descriptors are following the NFV-IFA014 standard. More details about the NFV-IFA014 standard can be found in this document:

[https://www.etsi.org/deliver/etsi\\_gs/NFV-IFA/001\\_099/014/02.01.01\\_60/gs\\_NFV-IFA014v020101p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/014/02.01.01_60/gs_NFV-IFA014v020101p.pdf)

- For example, the NSDs in IFA014 corresponding to the above VSBs from step 1 can be found in the following links:
  - Simple NSD [https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb\\_apache\\_simple/vsb\\_apache\\_simple\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb_apache_simple/vsb_apache_simple_nsd.yaml)
  - Advanced NSD [https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb\\_asti\\_agv/asti\\_agv\\_vsb\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb_asti_agv/asti_agv_vsb_nsd.yaml)
- Whereas, the NSD corresponding to the delay context blueprint from step 1 can be found here:

- Simple [https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_delay/ctx\\_delay\\_apache\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_delay/ctx_delay_apache_nsd.yaml)
- Advanced [https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_delay/asti\\_ctx\\_delay\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_delay/asti_ctx_delay_nsd.yaml)
- Similarly, the NSD associated with the background traffic context blueprint can be found in the following link:  
[https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx\\_bg\\_traffic/asti\\_ctx\\_bg\\_traffic\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/ctx/ctx_bg_traffic/asti_ctx_bg_traffic_nsd.yaml)
- After preparing all the NSDs corresponding to each of the blueprints, one more NSD needs to be prepared: the “composite NSD”. The composite NSD is the final network service which encompasses both the vertical service and context NSDs. It is the final NSD that will actually be executed by the 5G EVE portal<sup>8</sup>.
- An example of a composite NSD that encompasses the above VS and Ctx NSDs can be accessed at the following link:
- Simple (without context) [https://github.com/5GEVE/blueprint-yaml/blob/master/expb/expb\\_apache\\_simple/expb\\_apache\\_simple\\_nsd\\_all.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/expb/expb_apache_simple/expb_apache_simple_nsd_all.yaml)
- Advanced [https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb\\_asti\\_agv/asti\\_agv\\_all\\_nsd.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/vsb/vsb_asti_agv/asti_agv_all_nsd.yaml)

### 2.3.3 CONVERT ALL YAML DESCRIPTORS TO JSON (OPTIONAL)

This optional step is for those who used the YAML format for both the blueprints and network service descriptors in Sections 2.3.1 and 2.3.2. If JSON format was used in Sections 2.3.1 and 2.3.2, this step can be skipped.

- At this point, an experiment developer should have all blueprints and corresponding NSDs in YAML format.
- Next step is to convert each of those descriptors, i.e. blueprints and NSDs, into JSON format.
- The following online tool can be used to convert the YAML files to JSON <https://www.json2yaml.com/convert-yaml-to-json>
- Once the YAML file has been converted to JSON using the above link, experiment developers have to copy the JSON file from the browser by clicking anywhere in the JSON text, press ctrl+A to select all, followed by ctrl+c to copy and save the file on his or her device with the same name as before but this time with the .json extension.
- At this point, for each of our blueprints and NSD there should be two files; one in YAML and the another one in JSON format.
- Once this step is done, then the experiment developer is ready to start designing the experiment within the 5G EVE portal.
- The 5G EVE portal only accepts the JSON formats of the descriptors so the JSON files from this step will be the ones uploaded to the portal.

### 2.3.4 Experiment design using the 5G EVE Portal

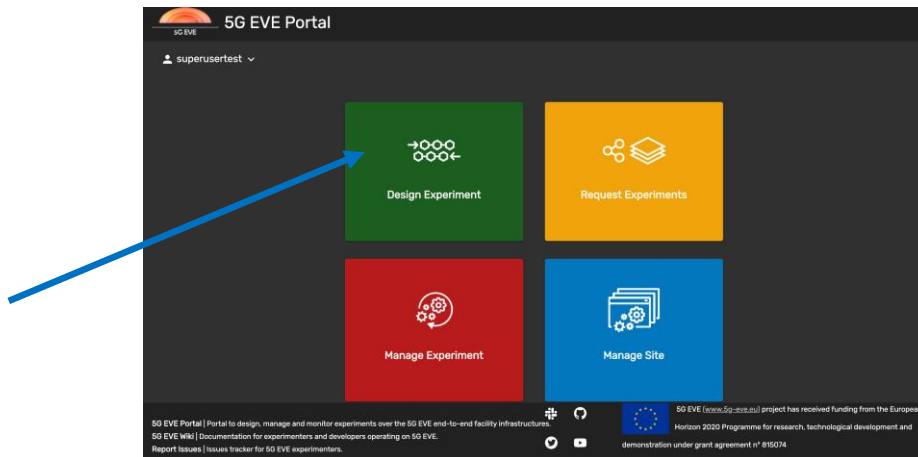
Before being able to design the experiment in the 5G EVE portal, any user needs to obtain the right login credentials.

- After logging into the portal, the user will have access to the screen shown in **Figure 16**.

---

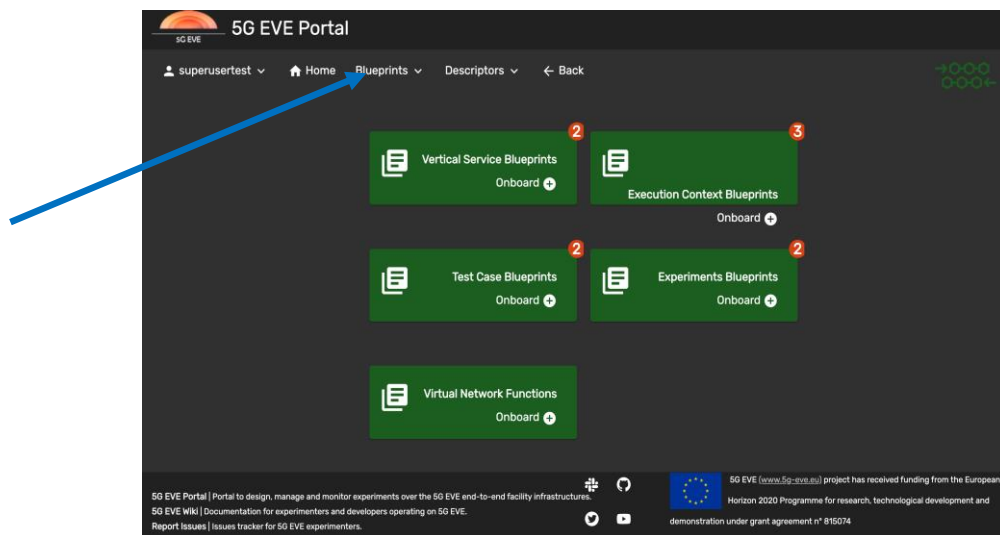
<sup>8</sup> This step will not be necessary in future versions of the 5G EVE Portal.





**Figure 16: 5G EVE PORTAL WELCOME PAGE**

- After clicking on the Design Experiment icon, the user will be led to the design experiment page which looks similar to **Figure 17**.



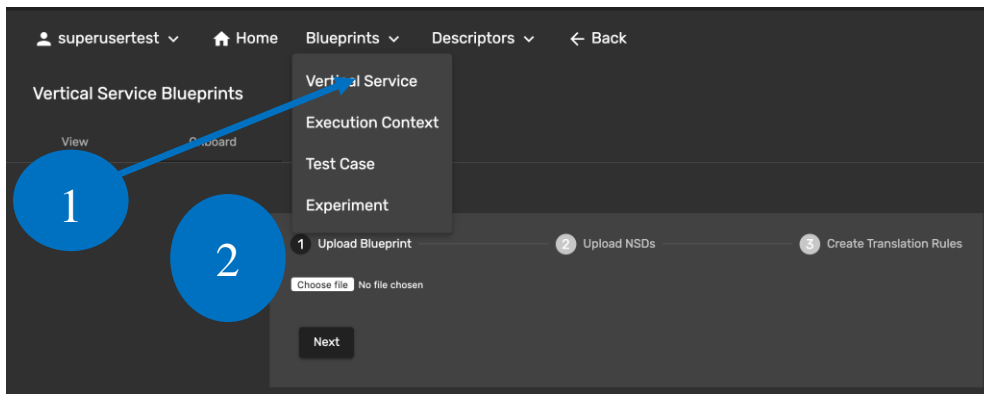
**Figure 17: 5G EVE PORTAL EXPERIMENT DESIGN PAGE**

- Once the user is on the design experiment page, the next step is to click on the blueprints section to display the different options representing each of the blueprints we discussed in previous section i.e. Vertical Service, Execution Context, Test Case and Experiment blueprints.

### 2.3.4.1 Uploading the Vertical Service Blueprint

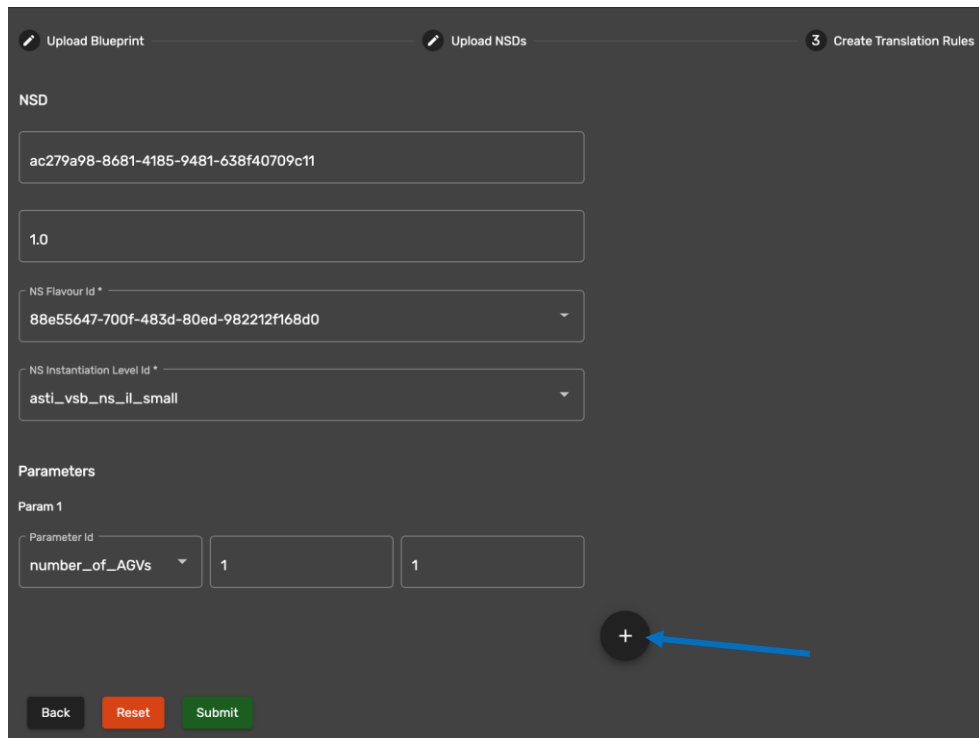
The next step is to begin with the vertical service blueprints, and similar steps for all the other blueprints.

- Experiment developers should click on the Vertical Service under the Blueprints section, to then be re-directed to a page similar to **Figure 18**.



**Figure 18: 5G EVE PORTAL VERTICAL SERVICE BLUEPRINTS DESIGN PAGE**

- There are three tasks to perform here: Upload Blueprint, Upload NSDs and Create Translation Rules.
- Beginning with task 1, a user has to upload the vertical service blueprint (VSB) from Section 2.3.3 above and for those whose descriptors were done in JSON from the start, then the user can upload the VSB descriptor.
- Once the user is done uploading the VSB, then he or she can click on the next button.
- Next, the experiment developer has to move on to task 2 to upload the NSDs. As stated in Section 2.3.2 above, every VSB has a corresponding NSD so at this point in time the user has to upload the NSD in JSON format (from Section 2.3.3) corresponding to your VSB.
- Next, experiments developers will be directed to the create translation rules task; by this stage the portal would have already read the identifier and version of the uploaded NSD and they will be auto filled by the portal. Please refer to **Figure 19** for more information.
- So, the only task of the user is to select the Network Service (NS) Deployment Flavor id and Instantiation Level id required in this particular experiment.
- All the available NS Deployment Flavors and Instantiation Levels inside the NSD will be presented to the user so to use the arrow button to select the right combination of Deployment Flavor and Instantiation Level ids that needed.
- The next field is the parameters section, for this field the 5G EVE portal will have already read the “parameters” section of the VSB. Thereafter, the portal will present the user with the VSB parameters so that the user selects the range of each VSB parameter required to be used in the experiment.
- If the VSB has more than one parameter, then the user can click on the + icon inside to add more parameters.

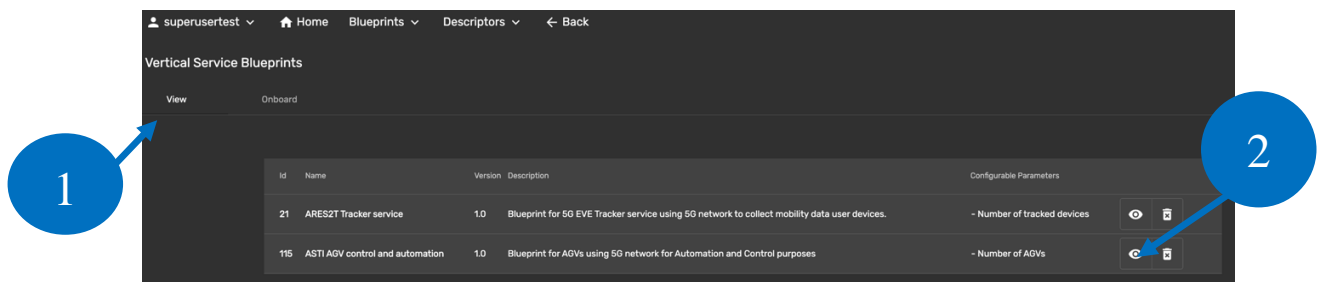


**Figure 19: 5G EVE PORTAL VSB TRANSLATION RULES**

- Once the user is done filling in/selecting all the required parameters, then the next step is to click the submit button.

### 2.3.4.2 Verification of the VSB submission

After clicking the submit button, if the submission was successful then the user will be presented with a message with the “**SUCCESS**” keyword in the body response. Otherwise, if the submission failed then the user will be presented with an error response message with the “**FAILED**” keyword in the body response. Once the submission is successful, the uploaded blueprint can be viewed by clicking on the “**view**” tab as shown in **Figure 20**.



**Figure 20: 5G EVE VIEW UPLOADED BLUEPRINT**

- As we can see in **Figure 20**, the submission was successful hence the ASTI AGV VSB is also listed.
- In order to view the uploaded blueprint, one can click on the eye icon as shown in **Figure 20**.
- Clicking on the eye icon, the blueprint looks as shown in **Figure 21**.

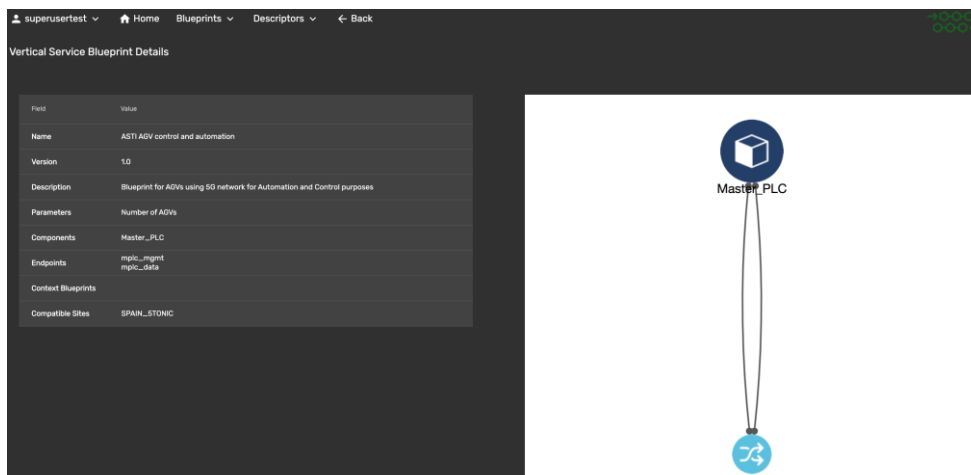


Figure 21: 5G EVE PORTAL UPLOADED VSB VIEW

**NOTE:**

- Sometimes, it may happen that the upload was successful although the blueprint is not shown when clicking on the eye icon.
- The VSB upload is only considered successful if one can view the VSB using the eye icon in the “view” tab.

### 2.3.4.3 Uploading of the Context Blueprints

At this step, the experiment developer has to go back to the blueprints section and this time, selecting the Execution context.

- The procedure to upload the execution context blueprints is exactly the same as the procedure to upload the vertical service blueprints (Section 2.3.4.1) as shown in **Figure 22**.
- The only difference being that now the user will be uploading the context blueprints from Section 2.3.1 or 2.3.3 with the corresponding NSDs instead of VSBs.

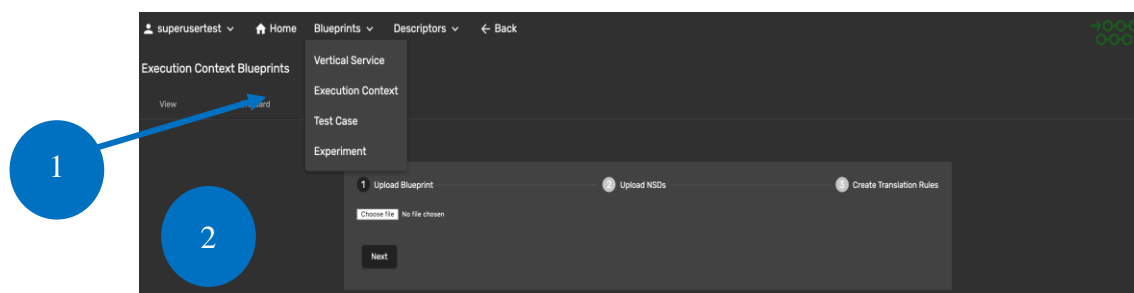


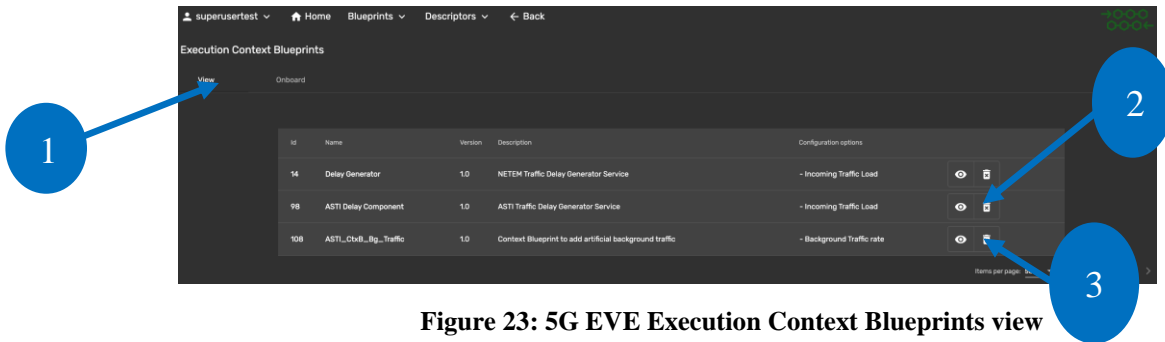
Figure 22: 5G EVE EXECUTION CONTEXT BLUEPRINTS UPLOAD

- Experiment developers have to upload all the context blueprints from Section 2.3.1 or 2.3.3 and the corresponding NSDs that they expect to use in their experiment.
- As stated in Section 2.3.1, in to elaborate this manual we were considering two contexts (i.e. delay and background traffic) and hence we had to upload two CtxBs and their corresponding NSD and translation rules. For help with the translation rules, please refer to Section 2.3.4.1 above and apply the same principles to the context NSDs in your experiment.
- Once done, the user has to click the submit button.

#### 2.3.4.3.1 Verification of the Execution Context Blueprints submission

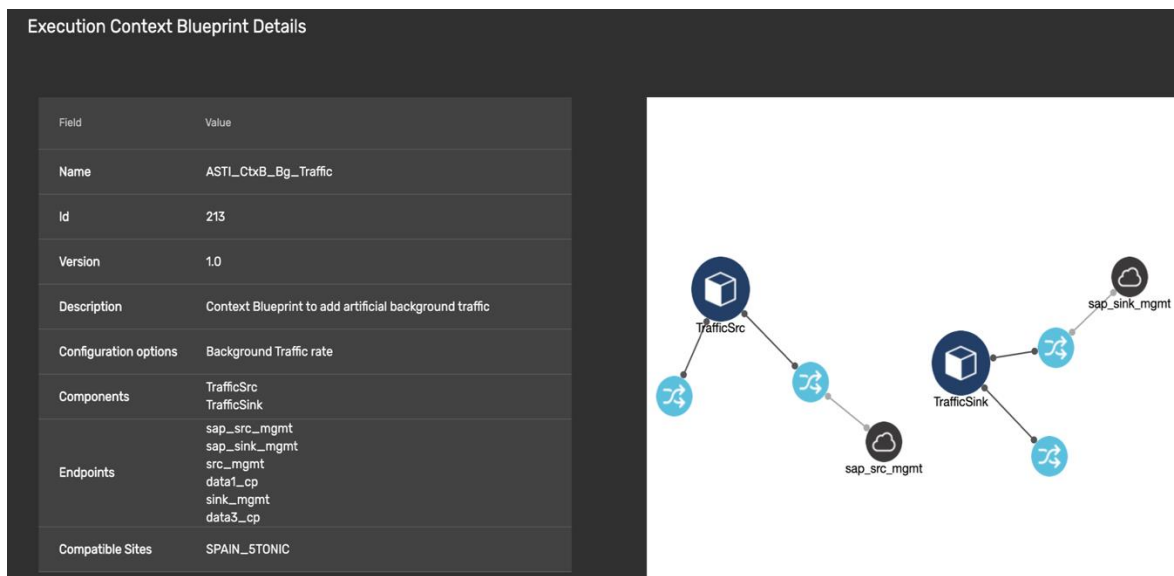
- After clicking the submit button, if the submission was successful then the user will be presented with a message with the “**SUCCESS**” keyword in the body response.

- Otherwise, if the submission failed then the user will be presented with an error response message with the “**FAILED**” keyword in the body response.
- In order to check that all the relevant context blueprints for the experiment have been successfully uploaded, the user has to click on the “**view**” tab of the execution context blueprints. As we can see from the “view” tab, both of the context blueprints were uploaded as shown in **Figure 23**.



**Figure 23: 5G EVE Execution Context Blueprints view**

- In order to verify that the context blueprints were successfully onboarded, the user needs to click on the eye icon for each of the context blueprints as shown in **Figure 23**.
- If the upload was successful, then the user will be able to view each of your uploaded context blueprints. In our case, the two contexts view looked as shown in **Figure 24 - Figure 26**



**Figure 24. 5G EVE ASTI Use-case delay context blueprint view**



Figure 25. 5G EVE ASTI Use-case delay context blueprint view

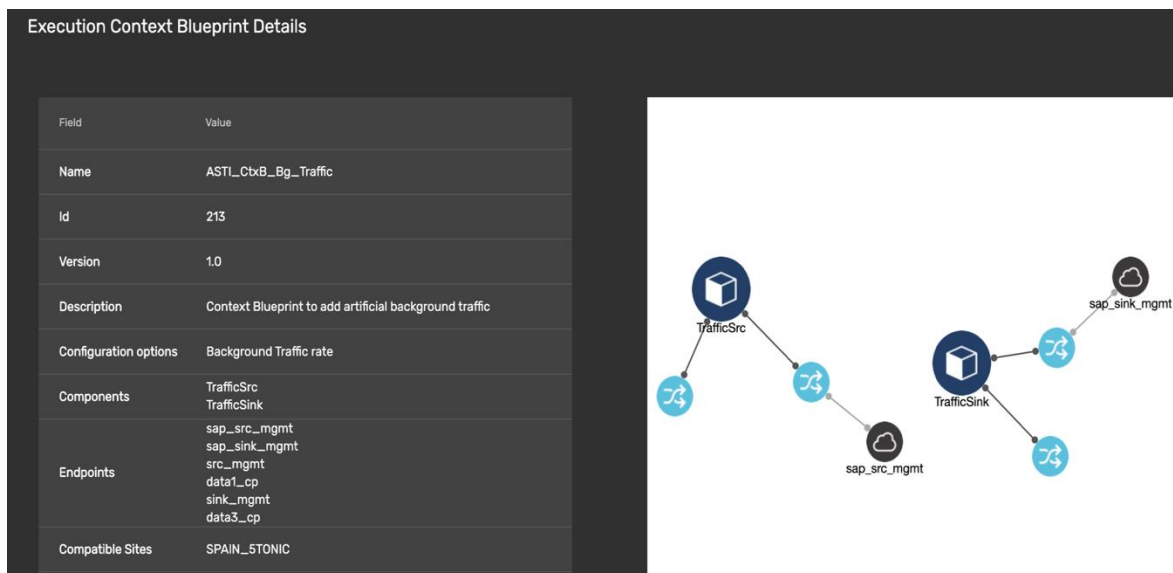
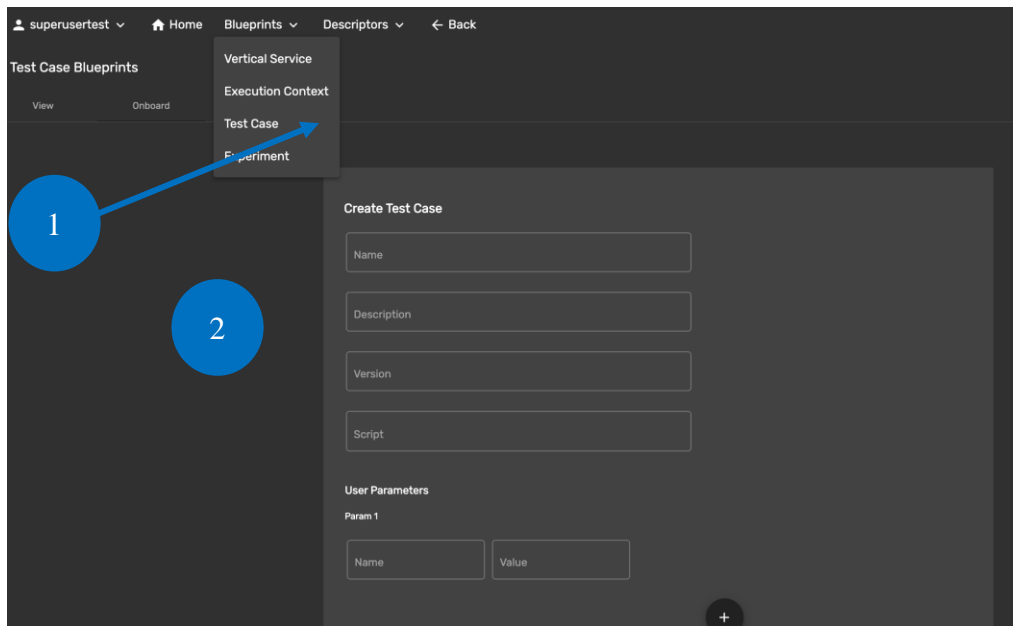


Figure 26: 5G EVE ASTI Use-case Background traffic context blueprint view

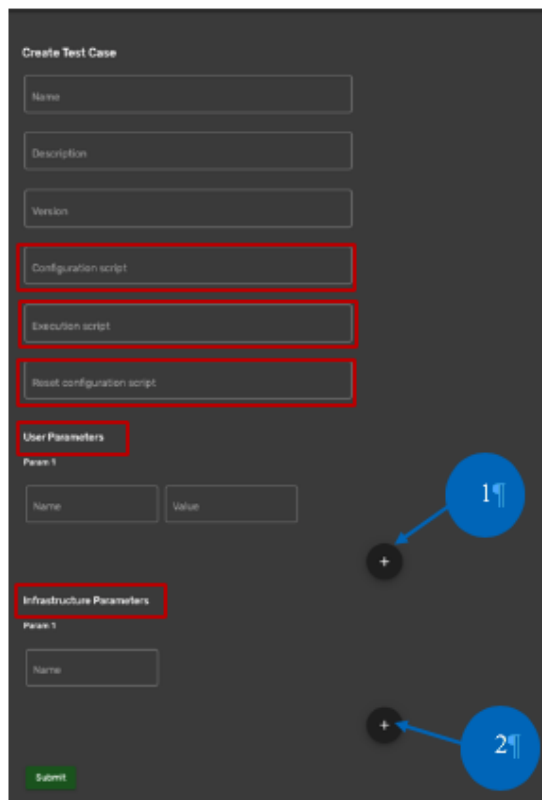
### 2.3.4.4 Uploading of the Test Case Blueprints

The experiment developer has to go back to the blueprints section and this time, select “**Test Case**” blueprints as shown in **Figure 27**.



**Figure 27: 5G EVE Portal Test case blueprint onboarding**

- Currently, the test case blueprints are created on the fly on the 5G EVE portal, so users do not need to upload any descriptors.
- All that is required of the user, is to fill in the test parameters related the different metrics (application and/or infrastructure) that users are considering in their experiment in the test-case blueprint form available in portal.
- In Figure 28 we show a screenshot and a text box at its right side, to illustrate what each of the test-case blueprint form parameters represent:

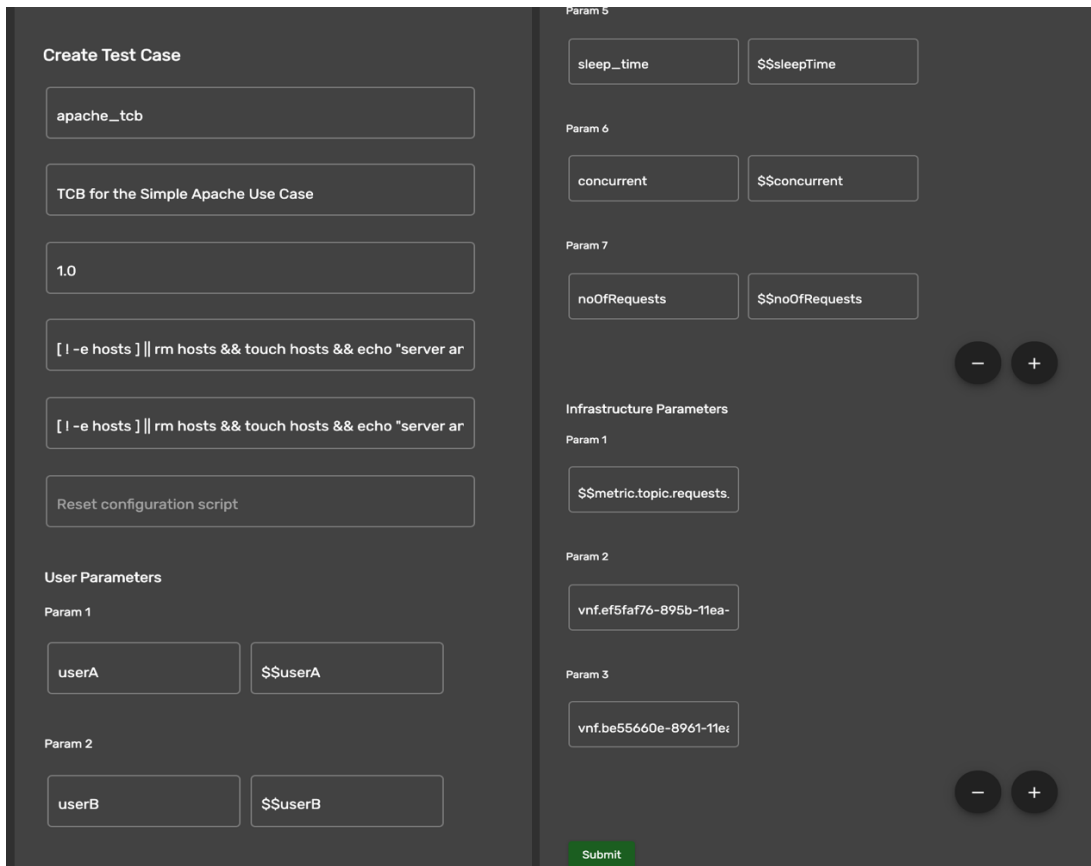


- → The name, description and version parameters are compulsory. So, go ahead and fill them in. ¶
- → Just remember you can only have one combination of name and version for each test-case blueprint. ¶
- → So, let us focus on the highlighted parameters: ¶
  - Label 1 – Configuration Script** ¶
  - → This parameter captures all the commands and/or scripts required to configure your vertical service to be ready to produce the application and/or infrastructure metrics. ¶
  - Label 2 – Execution Script** ¶
  - → This parameter captures all the commands and/or scripts required to generate the application and/or infrastructure metrics for your vertical service test case. ¶
  - Label 3 – Reset configuration Script** ¶
  - → This optional parameter captures all the commands and/or scripts required to clean/clear your vertical service environment after it has finished executing the test case. ¶
  - Label 4 – User Parameters** ¶
  - → Here, we provide all the parameters needed to access our vertical service (e.g. authentication) and run the test case (e.g. script arguments). ¶
  - → This includes the user parameters required by your **Cx/B(s)** in case there is any. ¶
  - → Use the + icon to add more user parameters. ¶
  - Label 5 – Infrastructure Parameters** ¶
  - → These are the parameters that should be provided by the 5G EVE platform to your test case for successful execution. ¶
  - → Examples include; VNF **ip** addresses, application metric topic id to publish the data too so that it's viewable in the 5G EVE monitoring and reporting tool. ¶

**Figure 28: Create Test Case Blueprint**

- Once all these fields are filled in the test case blueprint (TCB) parameters, then users can submit their TCB by clicking on the submit button.
- An example of a filled-in test case blueprint is given in **Figure 29**.





**Figure 29: 5G EVE Portal simple\_apache\_UseCase\_TCB**

- A detailed TCB for the simple apache use case can be found in the following link, which can be modified to suit anything a use case test needs, and thereafter copy and paste into the 5G EVE portal TCB section to create a custom TCB.

[https://github.com/5GEVE/blueprint-yaml/blob/master/tcb/ApacheTCB\\_v2.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/tcb/ApacheTCB_v2.yaml)

- It is important to notice that the last version of the Portal allows users to upload a JSON file with TCBs, simplifying this task.

**Verification of the Test Case Blueprints submission**

- After clicking the submit button, if the submission was successful then the user will be presented with a message with the “**SUCCESS**” keyword in the body response.
- Otherwise, if the submission failed then the user will be presented with an error response message with the “**FAILED**” keyword in the body response.
- To verify that a submission was successful, an experiment developer can click of the “view” tab of the test case blueprints to see the submitted TCB in the list. For our example, this is illustrated in **Figure 30**.

ID	Name	Version	Description	User Parameters	Infrastructure Parameters
11	Apache Simple TCB	1.0	Apache Simple TCB	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	-url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint
41	TCB BESTTUP test case 1	1.0	Test case to run the BESTTUP use case	executionTime: \$@executionTime	
54	Apache_TCB_v2	2.0	TCB Apache simple v2	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	\$@metric: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint
71	Apache_TCB_v3	3.0	Apache_TCB_v3	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	\$@metric: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint
80	test_sash	1.0	test_sash		
100	TCB BESTTUP test case 2	1.0	Test case to run the BESTTUP use case 2	executionTime: \$@executionTime	
118	Apache_TCB_v4	2.0	Apache_TCB_v4	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	\$@metric: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint
128	apache_sash_v4_v2	2.0	apache_sash_v4_v2	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	\$@metric: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint
144	TCB BESTTUP test case 31	3.1	Test case to run the BESTTUP use case 31	executionTime: \$@executionTime	
162	apache_sash	1.0	TCB for the Simple Apache Use Case	parameters: \$@component \$@component \$@component user: \$@user user: \$@user	\$@metric: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint -url: http://10.10.10.10:8080/endpoint

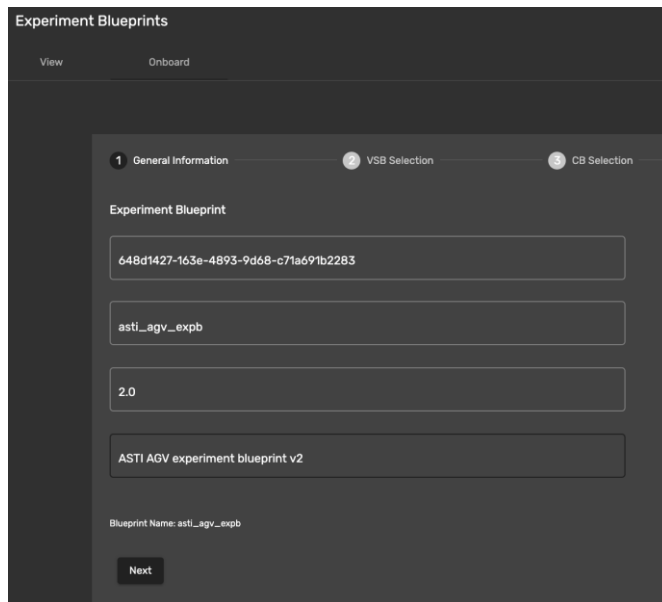
Figure 30: 5G EVE SIMPLE\_APACHE\_UseCase\_TCB VIEW

### 2.3.4.5 Uploading of the Experiment Blueprints

In order to design the Experiment Blueprint, experiment developers have to go back to the “descriptors” section and select Experiment as shown in **Figure 31**.

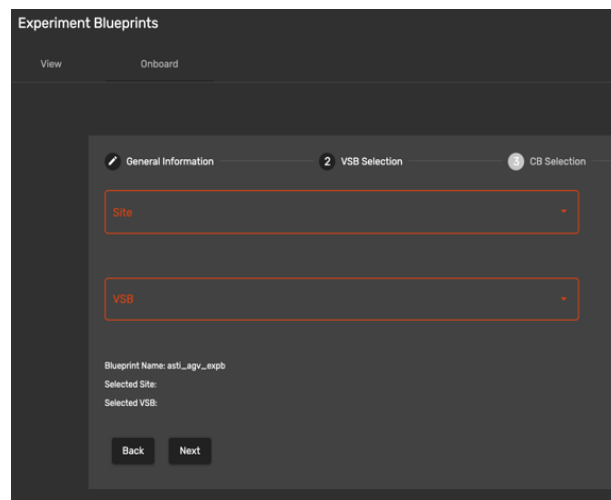
Figure 31: 5G EVE PORTAL EXPERIMENT BLUEPRINT DESIGN

- The fields highlighted in red are compulsory.
- First, the user has to fill in the id, name, version and optionally the description to use for the experiment blueprint. Only one unique combination of the name and version is allowed.
- An example of a filled-in experiment blueprint form is given in **Figure 32**.
  - Currently, the Portal provides an additional field “Deployment Type” with two possible options i.e., ON\_DEMAND or STATIC. If an experiment includes a Network Service Descriptor (NSD), then the experiment developer has to select the “ON\_DEMAND” option, otherwise if the experiment does not include any NSDs (i.e., no NSDs), then the user has to select the “STATIC” option.

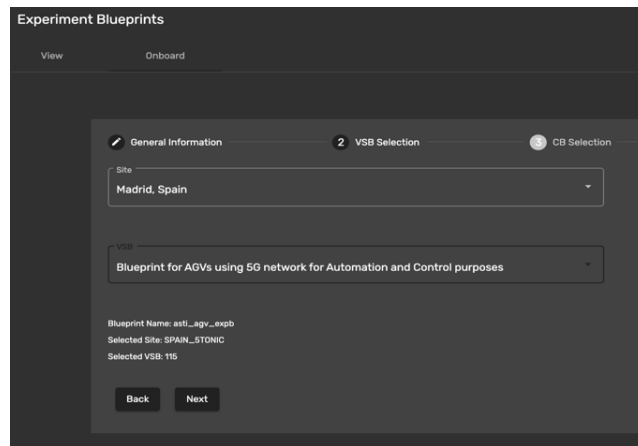


**Figure 32: 5GEVE Portal ASTI\_USECASE Experiment Blueprint**

- The next step is to move on the VSB selection stage. In the VSB selection stage, using the arrow icon, users can select the site where they want to conduct the experiment and thereafter, selecting the VSB that they want to use in that site. All previously uploaded VSBs will be available to experiment developers and they can use the arrow icon to select the VSB that they want to use for their experiment. This process is illustrated further both in **Figure 33** and in **Figure 34**.

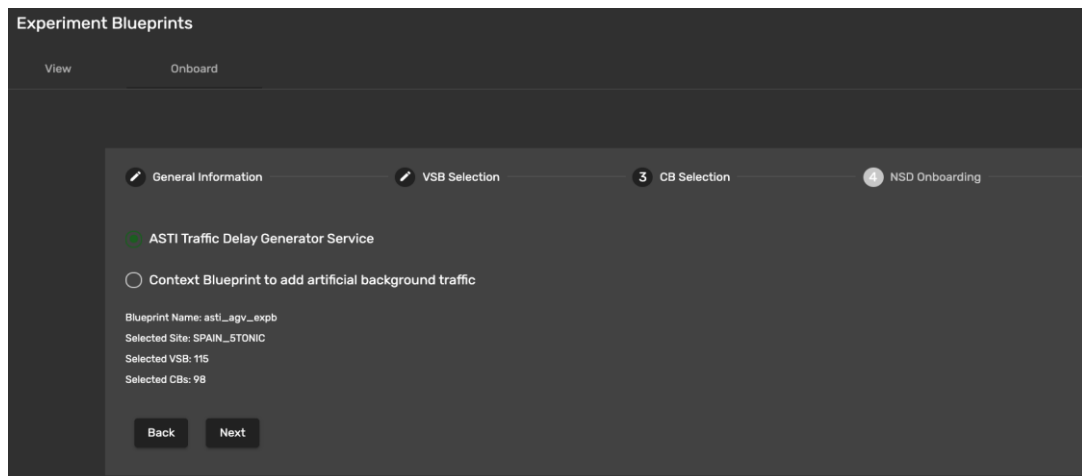


**Figure 33: 5G EVE PORTAL VSB Selection**



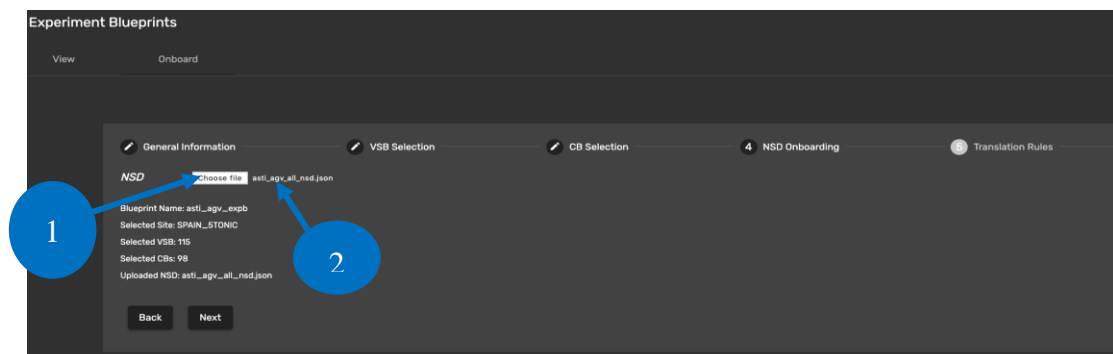
**Figure 34: 5G EVE PORTAL VSB Selection**

- An example of a selected site and VSB for the ASTI Use-Case is given in **Figure 35**.
- The next step is to move on to the Context Blueprint selection process.
- At this point, all the compatible context blueprints related to a VSB are available to the experiment developer to select the most adequate context blueprint for a particular experiment. In this case, we selected the ASTI delay context blueprint.



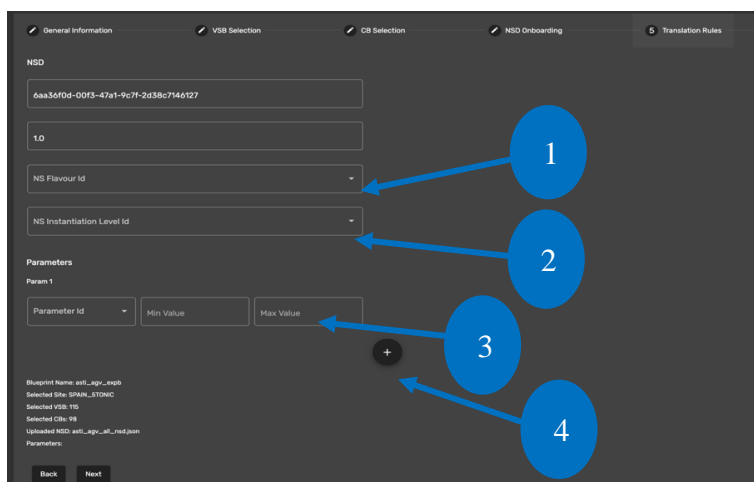
**Figure 35: 5G EVE ASTI\_USeCase CtxB Selection**

- The user can click next to move on to the NSD Onboarding stage. At this point, the experiment developer has to upload the **composite NSD** prepared in Section 2.3.2 and 2.3.3.
- The user has to click on choose file, and select the proper “composite NSD” file that encompasses both the VSB and CtxB NSDs. In reference to **Figure 36**, at point 2 users will be able to view their onboarded composite NSD.



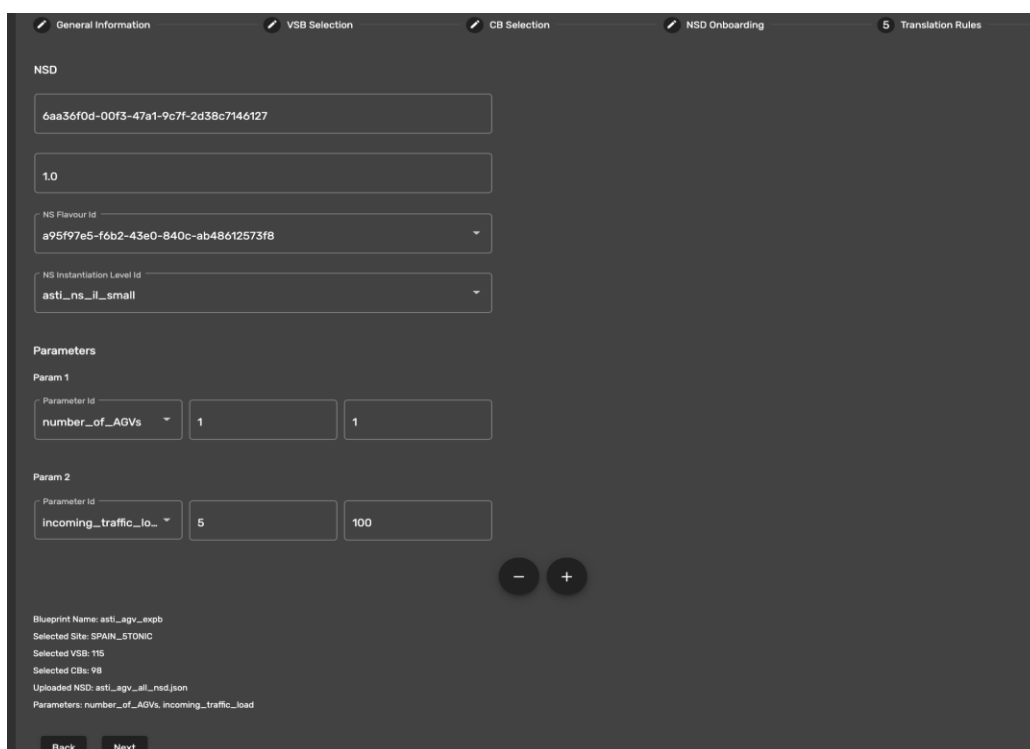
**Figure 36: 5G EVE ASTI USE\_CASE COMPOSITE NSD UPLOADING**

- After clicking Next to move on to the translation rules step, users will be presented with a page similar to **Figure 37**.
- In the translation rules section, the id and version of the uploaded composite NSD will have already been read by the portal.
- Next using the arrow icon, the user selects the Network Service (NS) Deployment flavor and NS Instantiation Level id to be used in the experiment.
- Next, experiment developers have to fill in the range of the parameters listed in both VSB and context blueprints. It is possible to click on the + icon to add more parameters.



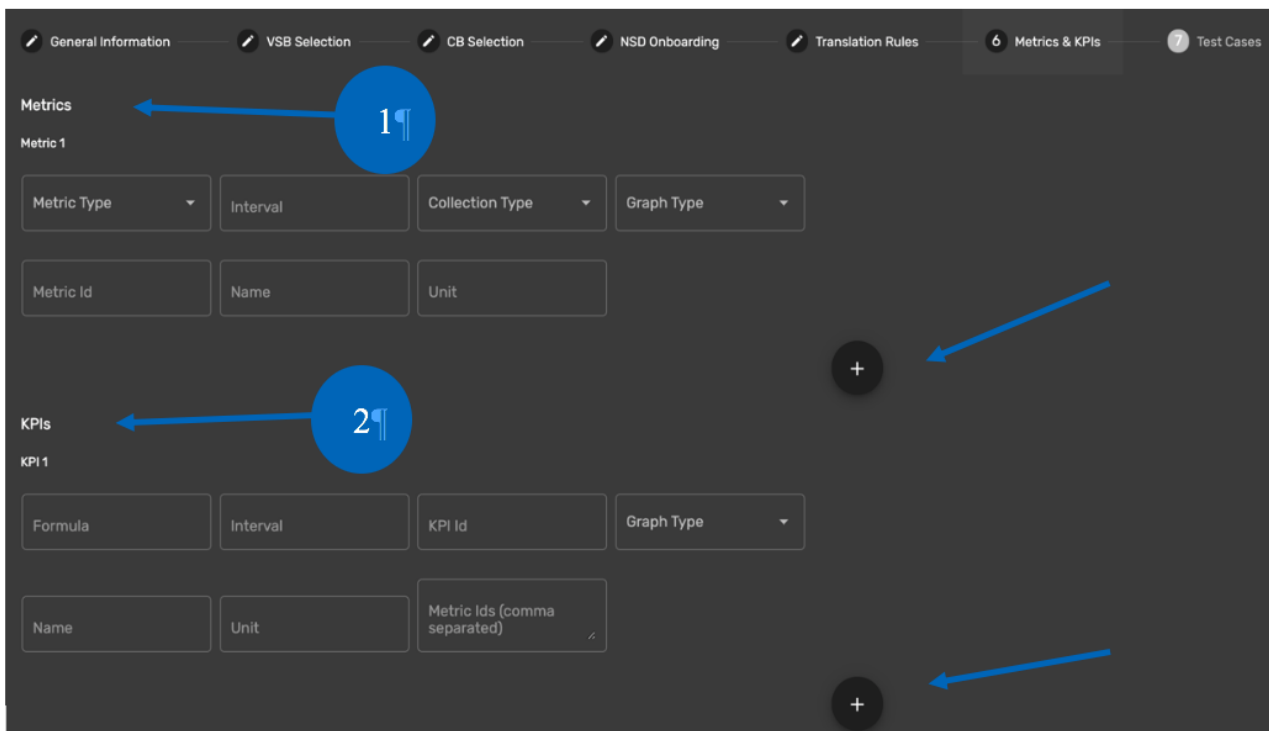
**Figure 37: 5G EVE Composite NSD Translation rules**

- An example of the composite NSD translation rules for the ASTI use-case is shown in **Figure 38**. The “*number\_of\_AGVs*” parameter is listed inside the parameters section of the ASTI VSB whereas the “*incoming\_traffic\_load*” parameter is listed inside the parameters section of the ASTI delay context blueprint.



**Figure 38: 5G EVE ASTI Composite NSD Translation rules.**

- Once done filling in all the required values, the user has to click next to move on to the Metrics & KPIs section.
- In the Metrics & KPIs section, there are two parts i.e. Metrics and KPIs as illustrated in **Figure 39**. To understand what each of the parameters in the Metrics and KPIs parts means, please refer to Table 62.
- The Metrics requested at this stage are the same metrics listed in the context blueprints under the “*applicationMetrics*” section.



**Figure 39: 5G EVE Experiment blueprint metrics & KPIs section**

**Table 62: 5G EVE Metrics & KPIs parameters**

Part 1- Metrics		
Parameter Type	Description	Comments
Metric Type	Type of the metric you want to measure	Current possible values [LOST_PKT, RECEIVED_PKT, SENT_PKT, LATENCY, BANDWIDTH]
Interval	The time interval by which you want to measure your metric	
Collection Type	Metric collection method	Current possible values [CUMULATIVE, DELTA, GAUGE]
Metric Id	Id of your metric	
Name	Name of the metric	
Unit	Standard SI Unit in which the metric is to be measured	For time-related metrics, this could be “s” for seconds and “ms” for milliseconds
<i>To add more metrics, please click on the + icon</i>		
Part 2- KPIs		
Parameter Type	Description	Comments

Formula	Formula/Equation to compute your KPIs	In case your KPI is a result of applying some sort of operation(s) to the collected metrics. Otherwise, if your KPI is the same as the collected metric, then in that case just put the metric name as the formula.
Interval	Interval at which this KPI should be computed	Depends on the KPIs you are looking at
KPI Id	Id of your KPI	uuid format
Name	Name of the KPI	
Unit	Standard SI Unit of the KPI	For time related KPIs, this could be “s” for seconds and “ms” for milliseconds
Metric Ids	Ids of the metrics you want to consider for this KPI	Add all the metric ids of the metrics used for this KPI. These metrics should be related to the formula given above
<i>To add more metrics, please click on the + icon</i>		

An example of a filled-in Metrics & KPIs section for the ASTI Use-Case is given in **Figure 40**.

**Metric 1**

Metric Type LATENCY	1	Collection Type GAUGE	Graph Type LINE
asti_agv_latency	asti_agv_latency	ms	

**Metric 2**

Metric Type LOST_PKT	1	Collection Type GAUGE	Graph Type LINE
asti_agv_lost_pkts	asti_agv_lost_pkts	percentage	

-
+

**KPIs**

**KPI 1**

asti_agv_latency	1	asti_agv_latency_kpi	Graph Type LINE
asti_agv_latency_kpi	ms	asti_agv_latency	

**KPI 2**

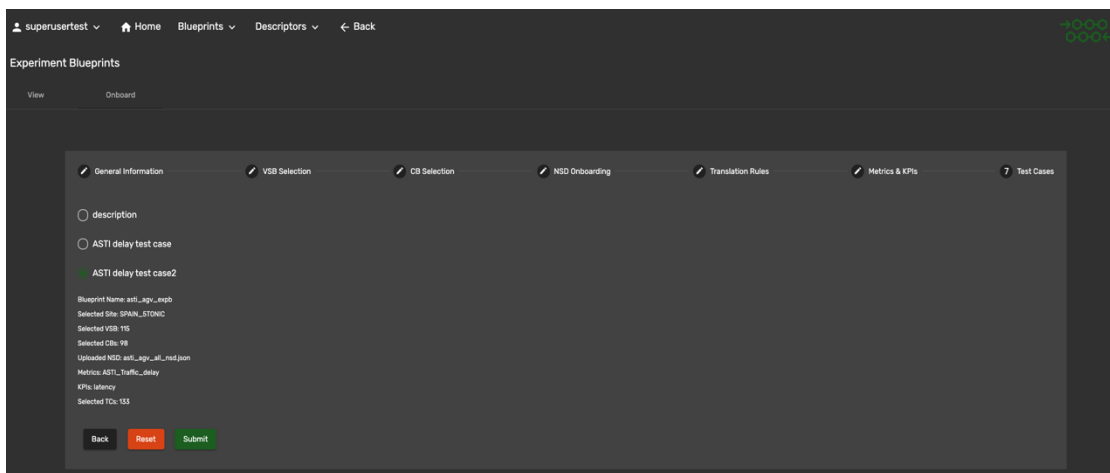
100 - (asti_agv_lost_pk	1	asti_agv_reliability_kpi	Graph Type LINE
asti_agv_reliability_kp	percentage	asti_agv_lost_pkts	

-
+

**Figure 40: 5G EVE ASTI Use-Case Metrics & KPIs**

- At this step, the user has to click Next to move on to the “Test Cases” section.
- In the “Test Cases” section, the user has to select one of the test-case blueprints created earlier (in Section 2.3.4.4) as shown in **Figure 41**.



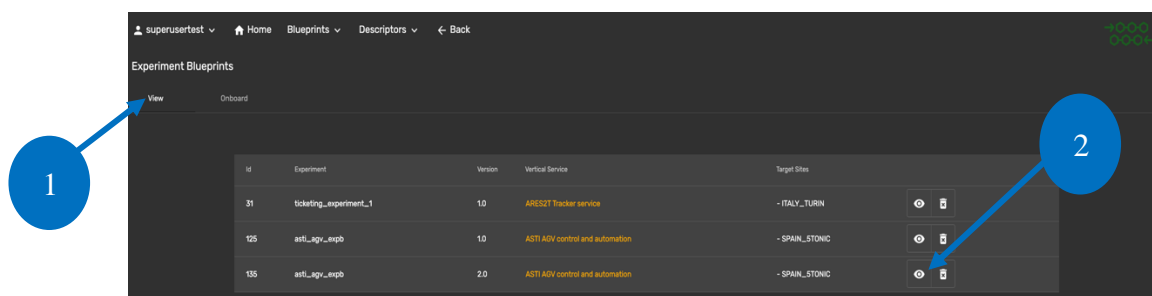


**Figure 41: 5G EVE ExpB Test Case Blueprint Selection**

- Once done, the user can click on the submit button to submit the completed experiment blueprint which contains the selected VSB, CtxB, TCB and composite NSD.

### 2.3.4.6 Verification of the Experiment Blueprints submission

After clicking the submit button, if the submission was successful then it will be presented with a message with the “**SUCCESS**” keyword in the body response. Otherwise, if the submission failed then the user will be presented with an error response message with the “**FAILED**” keyword in the body response. In order to verify that the Experiment Blueprint was uploaded successfully, the user has to click on the “**view**” tab of the Experiment Blueprints section. This is illustrated in **Figure 42**.



**Figure 42: 5G EVE EXPERIMENT BLUEPRINTS VIEW**

- As you can see from **Figure 42**, our experiment is listed. However, to confirm that the submission was successful we need to click on the eye icon.
- Clicking on the eye icon, we can see that we are able to view our experiment blueprint as shown in **Figure 43**.

Experiment Blueprint Details

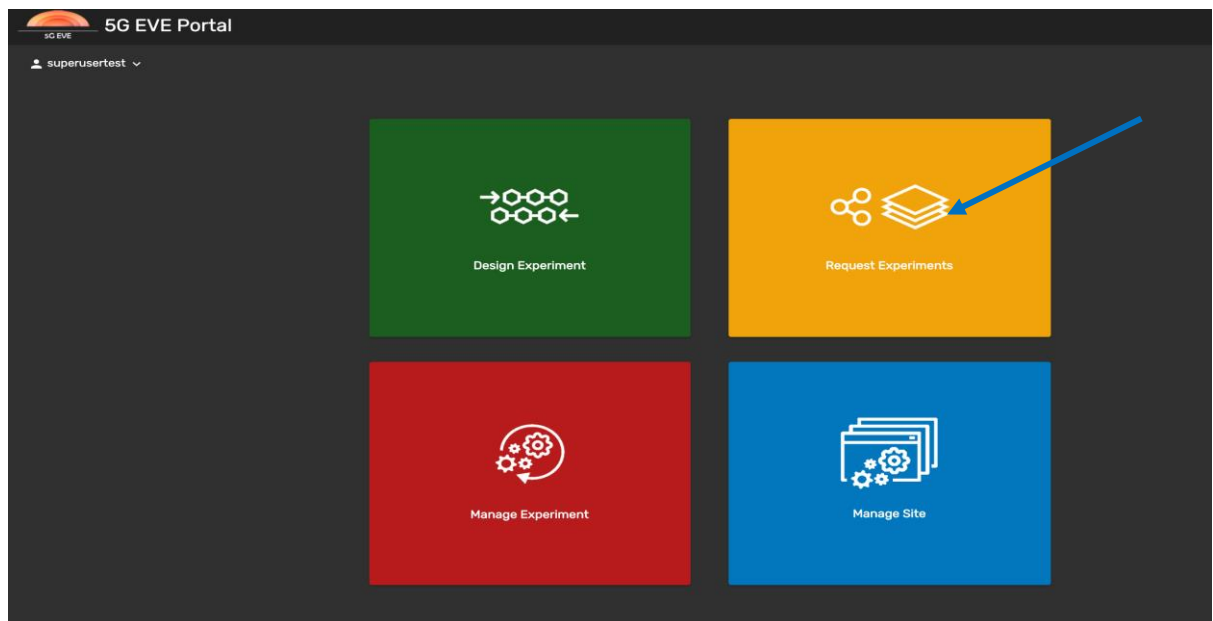
Field	Value
Name	asti_agv_expb_27_05
Id	229
Version	1.0
Description	ASTI AGV experiment blueprint 27_05
KPIs	asti_agv_latency_kpi asti_agv_reliability_kpi
Metrics	asti_agv_latency asti_agv_lost_pkts
Compatible Sites	SPAIN_5TONIC
Vertical Service	ASTI AGV control and automation
Execution Contexts	ASTI Delay Component ASTI_CtxB_Bg_Traffic
Test Cases	asti_testcase_v2
Onboarded NSDs	af07cf75-5770-46c0-9f75-373a30877556

**Figure 43: 5G EVE ASTI USE-CASE EXPERIMENT BLUEPRINT**

- At this point, the experiment developer is done with designing the experiment hence it is possible to move on the next section, i.e., requesting an experiment.

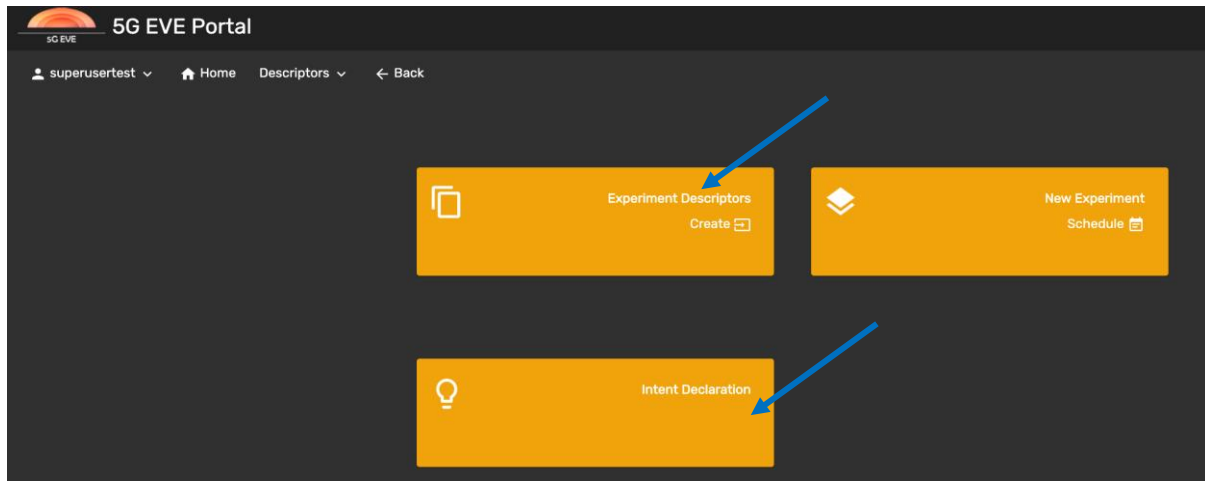
### 2.3.5 5G EVE Portal Request Experiment

In this section, we will cover the steps taken in order to request an experiment in the 5G EVE portal. This process is focused to experimenters. They can go to the 5G EVE portal home page and this time, click on the Request Experiments tool as shown in **Figure 44**.



**Figure 44: 5G EVE PORTAL HOME PAGE**

- After clicking on the “Request Experiments” section, experimenters will be presented with a screen similar to the one shown in **Figure 45**.
- Inside the 5G EVE portal, experimenters can request an experiment by either using the “**Experiment Descriptors create**” wizard or by using the “**Intent-based networking (IBN) experiment scheduling**” tool as illustrated in **Figure 45**.

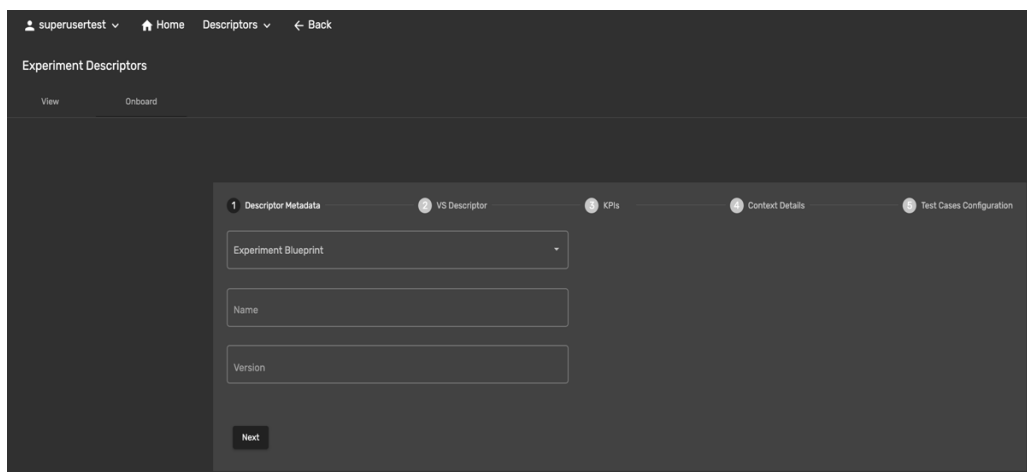


**Figure 45: 5G EVE Portal Request Experiments page**

### 2.3.5.1 5G EVE Portal Create Experiment Descriptor

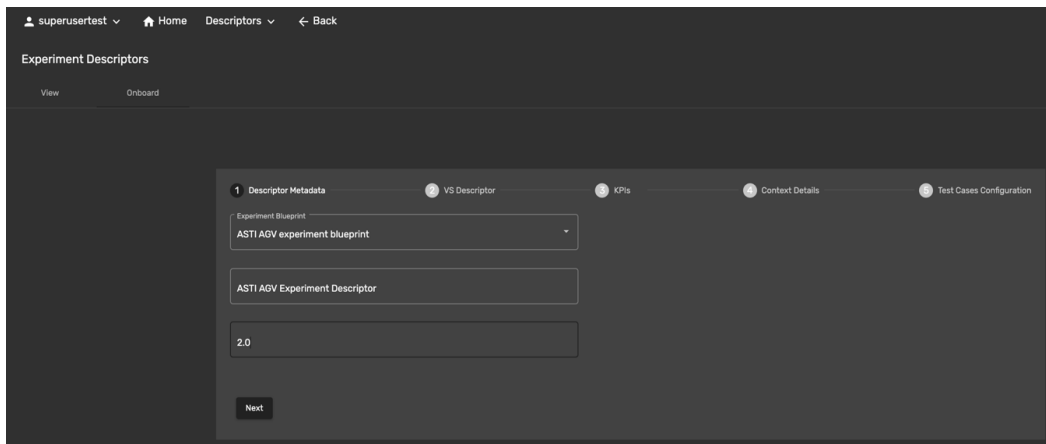
To request an experiment using the Experiment Descriptors create wizard, experimenters can click on the “*Experiment Descriptors Create*” icon. This wizard will guide them through the steps needed to create the various descriptors of their experiment, i.e., Vertical Service, Context, Test-Case and Experiment Descriptors.

- Once inside the “*Experiment Descriptors Create*”, experimenters will be presented with a form as shown on **Figure 46**.
- Using the arrow icon to select the experiment blueprint associated to this experiment descriptor followed by the name and version.



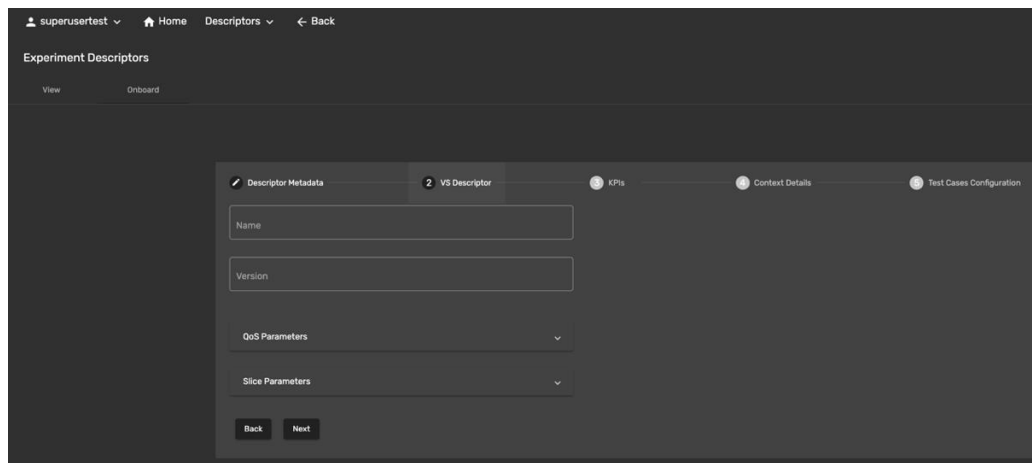
**Figure 46: 5G EVE Experiment Descriptor metadata page**

- An example of the Experiment descriptor metadata for the ASTI use case that we have been using in this document is given in **Figure 47**.



**Figure 47: 5G EVE ASTI USE-CASE Experiment Descriptors Metadata**

- The next step is to move on to the “VS Descriptors” section. The “VS Descriptors” page is displayed in **Figure 48**.



**Figure 48: 5G EVE Vertical Service Descriptor page**

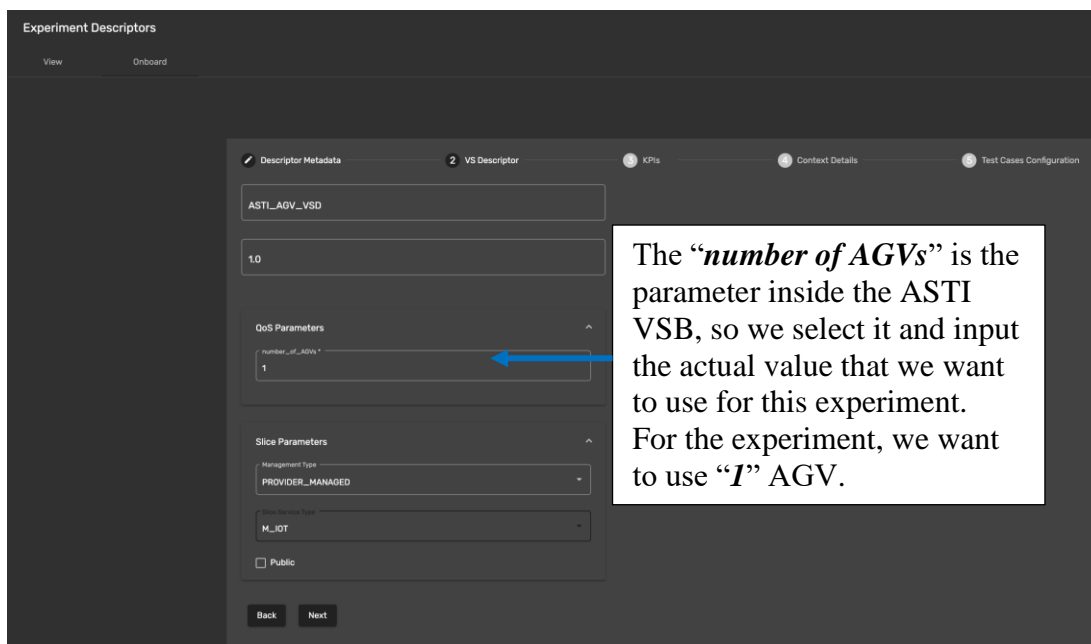
- Inside the “*VS Descriptors*” section, experimenters can input the name and version that they want to use for the Vertical Service Descriptor.
- The main difference between the Vertical Service Descriptor (VSD) and Vertical Service Blueprint (VSB) is that the VSD is the parameterized version of the VSB i.e. Inside the VSD, the actual values of the VSB parameters to be used in the experiment are provided.
- Inside the VSD, the VSB parameters are captured by the QoS parameters section. Experimenters can use the arrow icon to select the VSB parameter to associate with their VSD.
- Next, experimenters can move on to the Slice Parameters selection. After clicking on the arrow button, the parameters shown in Table 63 will be presented.

**Table 63: 5G EVE VSD Slice Parameters Description**

Slice Parameters		
Parameter	Possible Values	Description
Management_Type	Provider_Managed	This vertical service is to be managed by the 5G EVE system
	Tenant_Managed	This vertical service will be managed by the Vertical Service Provider
	EMBB	Mobile Broadband use-cases

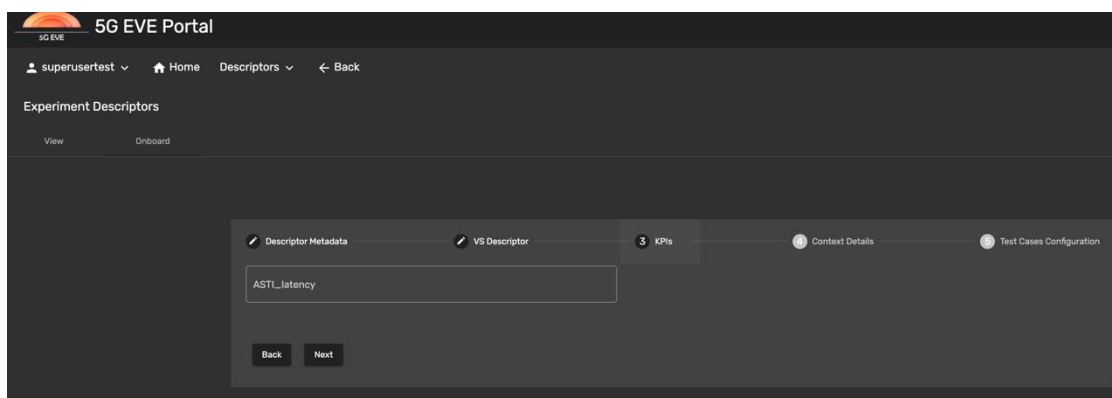
Slice_Service_Type	URLLC	Ultra-low latency applications
	MMTC	Massive Machine Type Communications

- Experimenters have to select the Management\_Type and Slice\_Service\_Type that corresponds to their use-case.
- Finally, experimenters can check *public* if their experiment is publicly available to every user of the 5G EVE portal. Please refer to **Figure 49** for the VSD of the ASTI use-case that we have been using in this document.



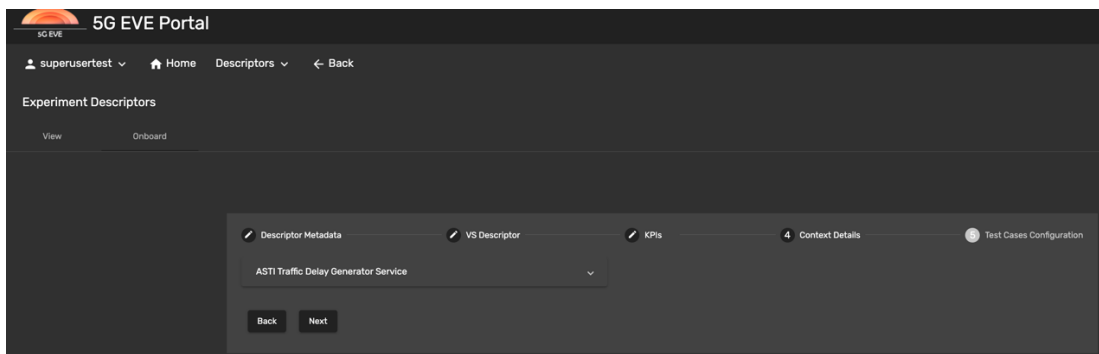
**Figure 49: 5G EVE ASTI Use-Case VSD parameters**

- The next step is to move on to the “KPIs” section. As you can see in **Figure 50**, the KPI that we added in the ASTI Experiment blueprint has already been selected for us. In this example, this KPI is ASTI\_latency.



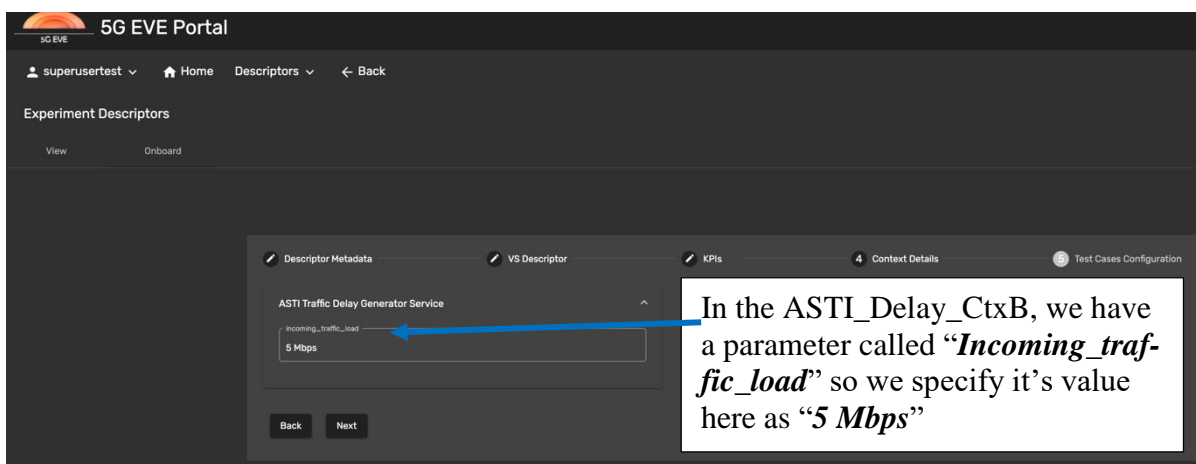
**Figure 50: 5G EVE ASTI Use-Case ExpD KPI**

- The next step is to move on to the “Context Details” section. As shown in **Figure 51**, the context associated with our Experiment blueprint has also already been selected for us.



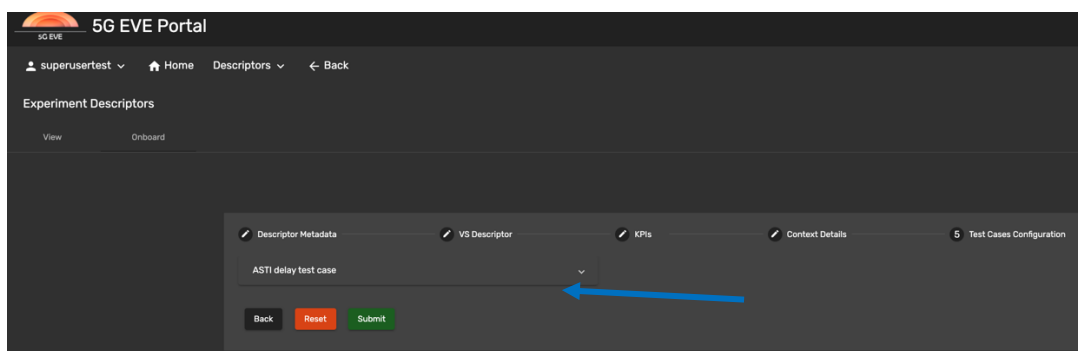
**Figure 51: 5G EVE ASTI ExpD Delay\_Context**

- By Clicking on the “ASTI traffic Delay Generator Service”, the parameter(s) associated with the delay context blueprint will be presented and experimenters have to select the actual value(s) of the parameter(s) that they want to use in their experiment. This is illustrated in **Figure 52**.



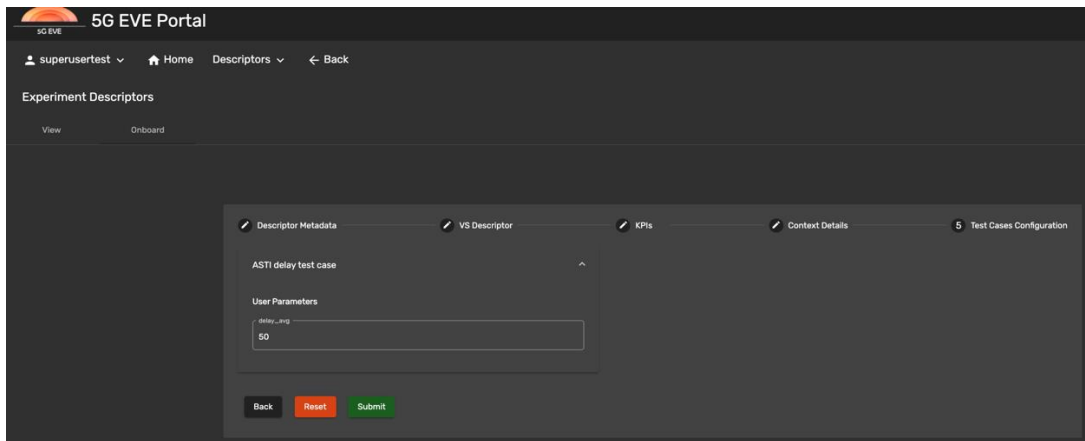
**Figure 52: 5G EVE ASTI ExpD Delay\_Context Parameter**

- The next step is to go to the “Test Cases Configuration” section. As shown in **Figure 53**, our “ASTI delay test case” associated with this experiment blueprint is already available to us.



**Figure 53: 5G EVE ExpD Test Cases Configuration**

- For the ASTI Use-Case, the test cases configuration parameter(s) are detailed in **Figure 54**.

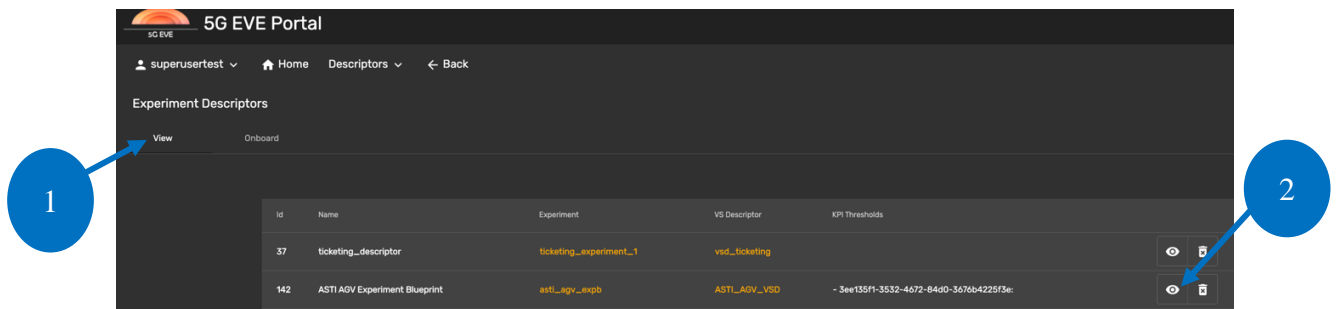


**Figure 54: 5G EVE ASTI ExpD Delay Test Case Parameters**

- Once done, experimenters can click on the “*submit*” button in order to submit their Experiment Descriptor.

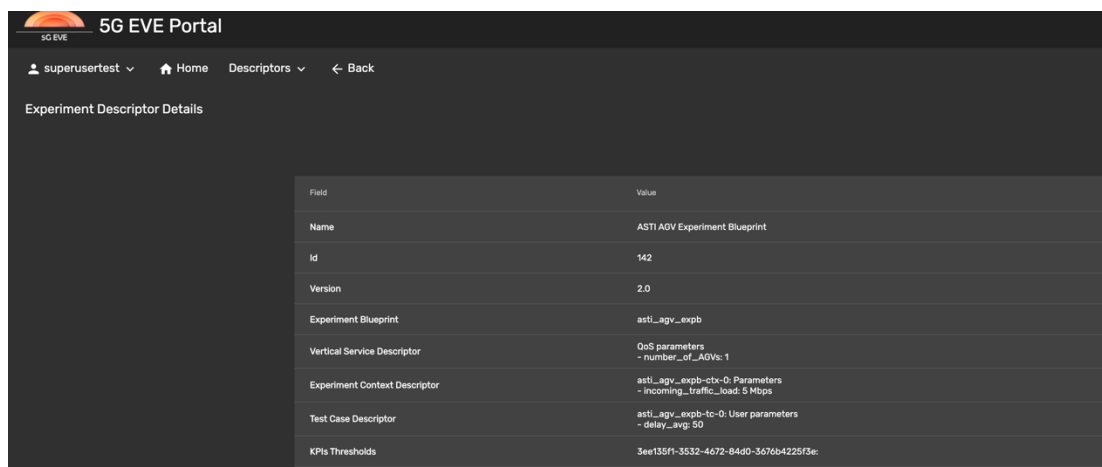
### 2.3.5.2 Verification of the Experiment Descriptor submission

In order to verify that their Experiment Descriptor was uploaded successfully, experimenters have to click on the “**view**” tab of the Experiment Descriptors section. This is illustrated in **Figure 55**.



**Figure 55: 5G EVE EXPERIMENT DESCRIPTORS VIEW**

- As it can be seen from **Figure 55**, our experiment is listed. However, to confirm that the submission was successful, we need to click on the eye icon.
- Clicking on the eye icon, we can see that we are able to view our descriptors as shown in **Figure 56**.

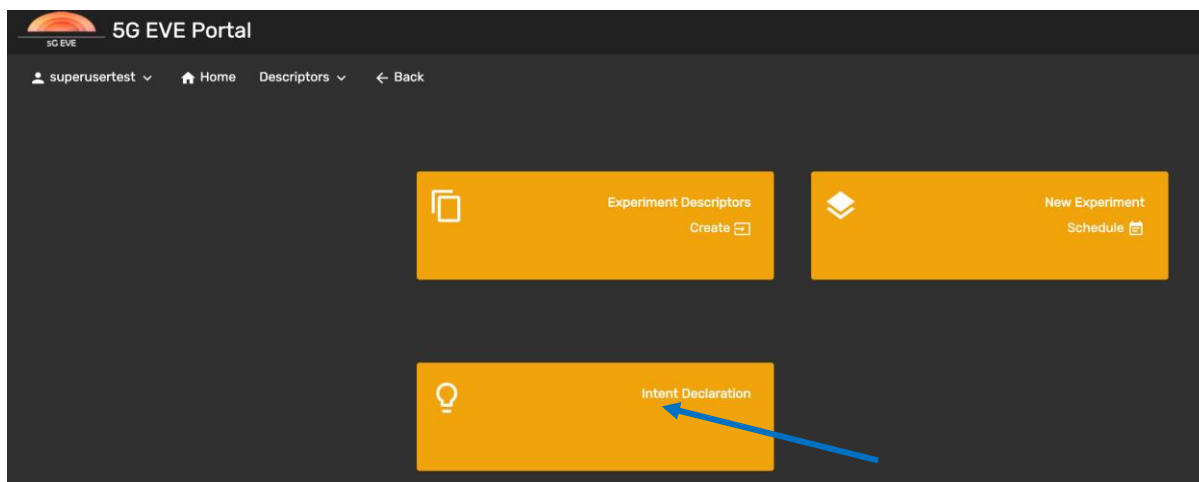


**Figure 56: 5G EVE ASTI Use-Case Experiment Descriptor Details**

- The next step is to schedule the experiment with the 5G EVE portal.

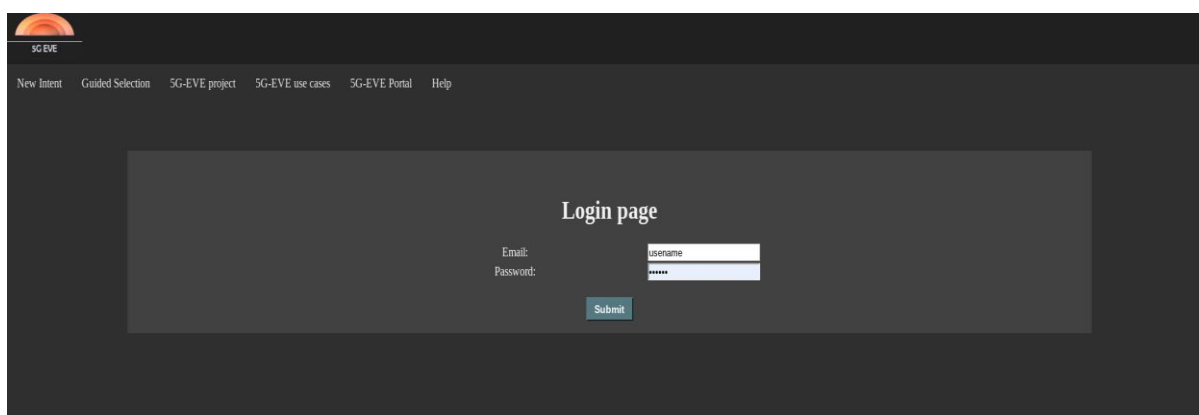
### 2.3.5.3 5G EVE Intent-Based Networking (IBN) Experiment Scheduling Tool

To request an experiment using the IBN experiment scheduling tool, experimenters have to click on the “Intent Declaration” icon as shown in **Figure 57**.



**Figure 57: 5G EVE Request Experiment using The IBN Tool**

- The IBN tool offers two modes of operation: the intent based selection and the guided selection.
- The intent based selection enables the user to write his intent using natural language, defining the desired service, sites and relevant parameters to customize an experiment
- The guided selection consists of a step-by-step wizard to set up and schedule the desired experiment.
- After clicking on the “Intent Declaration” icon, a login page as shown in **Figure 58** appears as it is mandatory for the user to enter his 5G EVE credentials to authenticate using the Role-Based Access Control Interface (RBAC).



**Figure 58: 5G EVE IBN RBAC page**

- A logout button appears at the upper right corner so that the user can easily logout and terminate the active session.
- After RBAC authorizes the user, the intent-based and guided selection tools become available.

### 2.3.5.4 Intent based selection

In this mode, the user is asked to fill in the blank fields with the intent. An intent can contain the following data:



- Information from the desired experiment's description, so that the most fitting experiment blueprint is found (i.e. experiment with ares2t)
- Information about the desired site following the keyword "in" (i.e. in Italy, in Turin, in Greece etc)
- Information about the experiment date that the user desires in the dd/mm/yyyy format (i.e. 26/06/2020)
- Information about the experiment start time in the hh:mm 24h format (i.e 16:45)
- Values for the service and context parameters using the "#" sign followed by the value
- Information about the experiment duration in minutes using the "#" sign followed by the duration and the "duration" keyword (i.e. #30 duration)

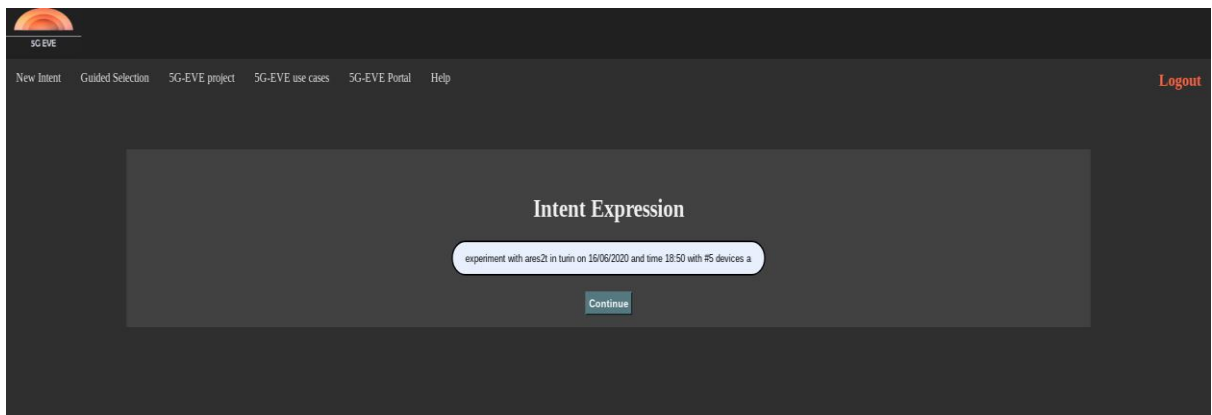
**Table 64: 5G EVE intent based selection parameters**

5G EVE Intent Parameters	Possible values
Experiment use-case	ares2t, asti, orange_v360, polito_smartcity and wings_utilities. More to be included in the future.
5G EVE site	Turin (Italy), 5TONIC(Spain), Athens (Greece), Paris (France), Nice (France), Rennes (France)
Date(dd/mm/yyyy)	16/06/2020
Time (hh:mm 24h format)	18:50
vsb and context parameters (begin with '#')	#5 devices (vsb parameter) and #10 load (ctxb parameter)
Experiment duration in minutes (begin with '#')	#40 duration

Therefore, a full intent that captures all these parameters can be as follows:

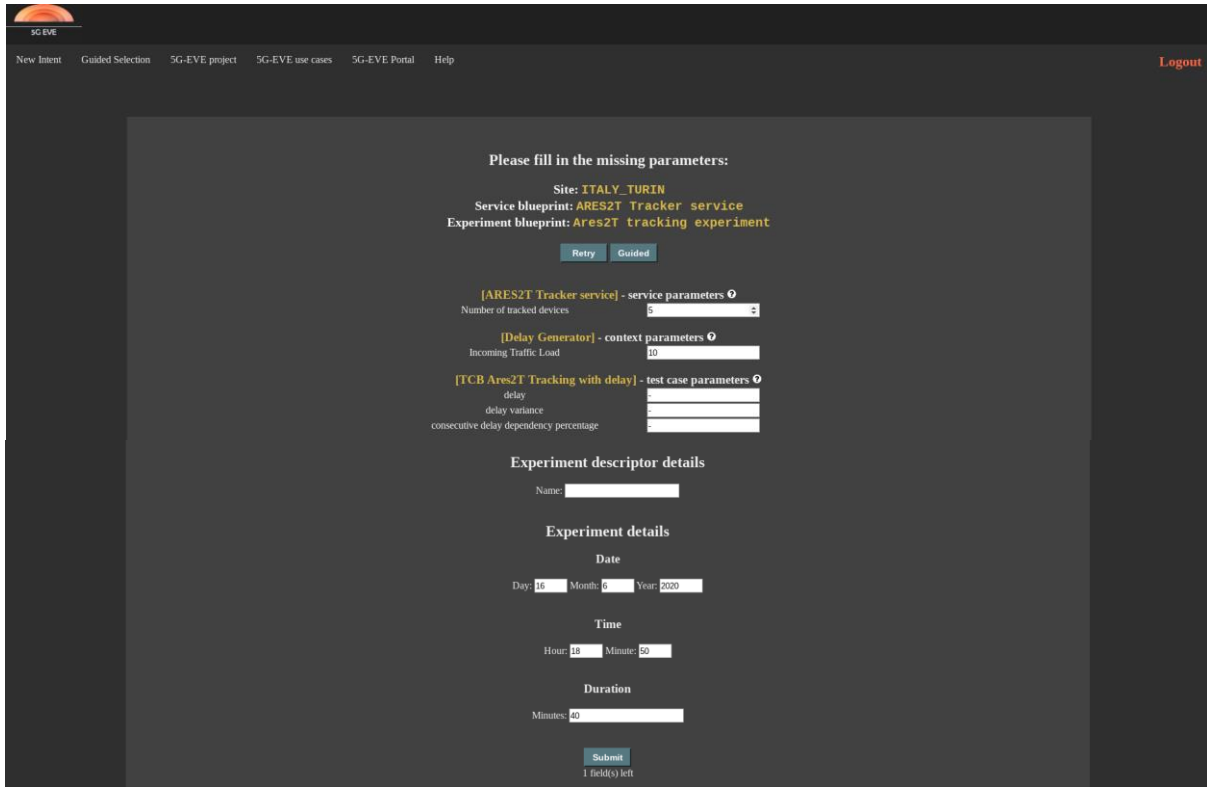
*“experiment with ares2t in turin on 16/06/2020 and time 18:50 with #5 devices and #10 load and #20 delay and #40 duration”*

This is illustrated further in **Figure 59**



**Figure 59: 5G EVE Intent Expression**

The input is analysed, and the optimal service and experiment blueprint pair is shown to the user along with the extracted site similar to **Figure 60**.



The screenshot shows a web interface with a dark theme. At the top, there is a navigation bar with links: "New Intent", "Guided Selection", "5G-EVE project", "5G-EVE use cases", "5G-EVE Portal", and "Help". A "Logout" link is in the top right corner. The main content area is titled "Please fill in the missing parameters:" and displays the following information:

- Site: ITALY\_TURIN
- Service blueprint: ARES2T Tracker service
- Experiment blueprint: Ares2T tracking experiment

Below this information are two buttons: "Retry" and "Guided". The form is organized into sections:

- [ARES2T Tracker service] - service parameters**: A dropdown menu for "Number of tracked devices" with the value "5" selected.
- [Delay Generator] - context parameters**: A text input field for "Incoming Traffic Load" with the value "10".
- [TCB Ares2T Tracking with delay] - test case parameters**: Three text input fields for "delay", "delay variance", and "consecutive delay dependency percentage", all with empty values.

Below the parameters are three sections for experiment details:

- Experiment descriptor details**: A text input field for "Name".
- Experiment details**:
  - Date**: Three input fields for "Day" (16), "Month" (6), and "Year" (2020).
  - Time**: Two input fields for "Hour" (18) and "Minute" (50).
  - Duration**: One input field for "Minutes" (40).

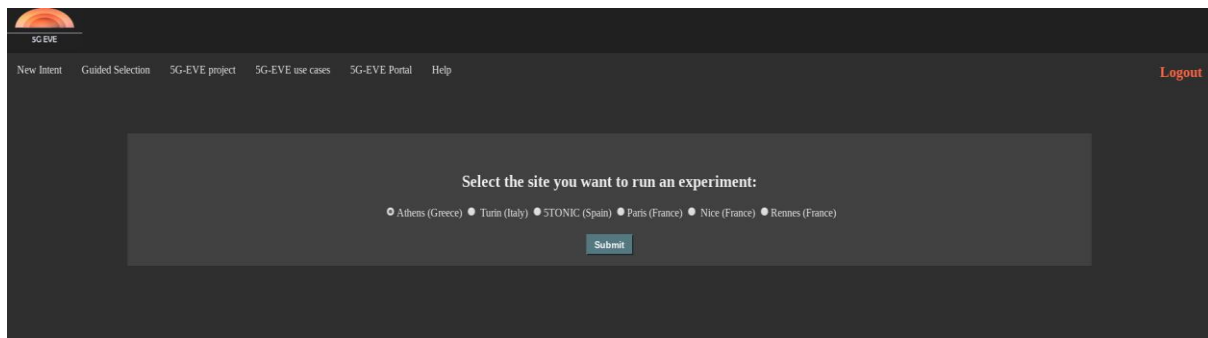
At the bottom of the form is a "Submit" button and a small text indicator "1 field(s) left".

**Figure 60: 5G EVE Intent based tool-Experiment blueprint and service parameters selection**

- All the parameters and experiment date/time/duration values that were recovered from the user's input are displayed in case the user wants to edit any value.
- It is important to note that the description field of each experiment blueprint that is onboarded at the Catalogue is evaluated and the one that is more correlated with the user intent is chosen.
- If every field has the desired value, the user submits the form.
- An experiment descriptor is created with all the information and is onboarded at the Portal Catalogue.
- Finally, the Experiment Lifecycle Manager (ELM) is notified about the desired experiment details and the scheduling takes place.

**Guided Selection**

- In this section, the user follows a step-by-step approach accompanied with useful messages in order to configure an experiment.
- Initially, the user selects the desired site(s) to run an experiment as shown in Figure 61



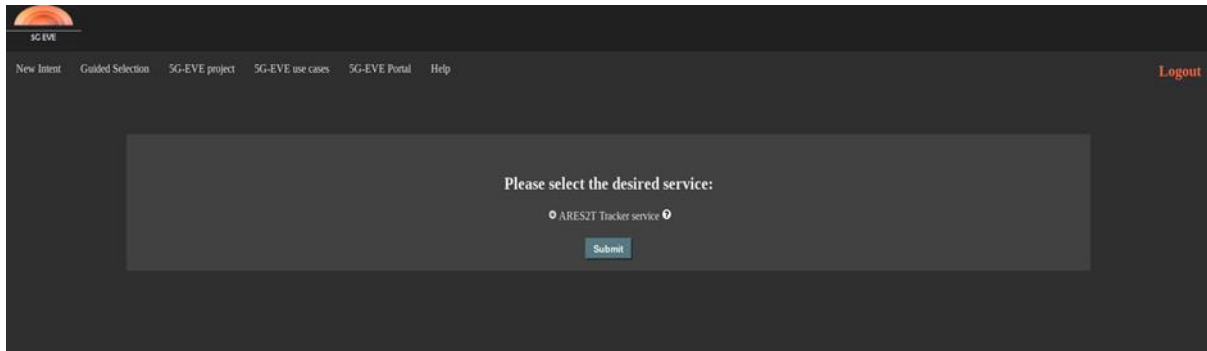
The screenshot shows a web interface with a dark theme. At the top, there is a navigation bar with links: "New Intent", "Guided Selection", "5G-EVE project", "5G-EVE use cases", "5G-EVE Portal", and "Help". A "Logout" link is in the top right corner. The main content area is titled "Select the site you want to run an experiment:" and displays a list of radio buttons for site selection:

- Athens (Greece)
- Turin (Italy)
- STONIC (Spain)
- Paris (France)
- Nice (France)
- Rennes (France)

Below the list is a "Submit" button.

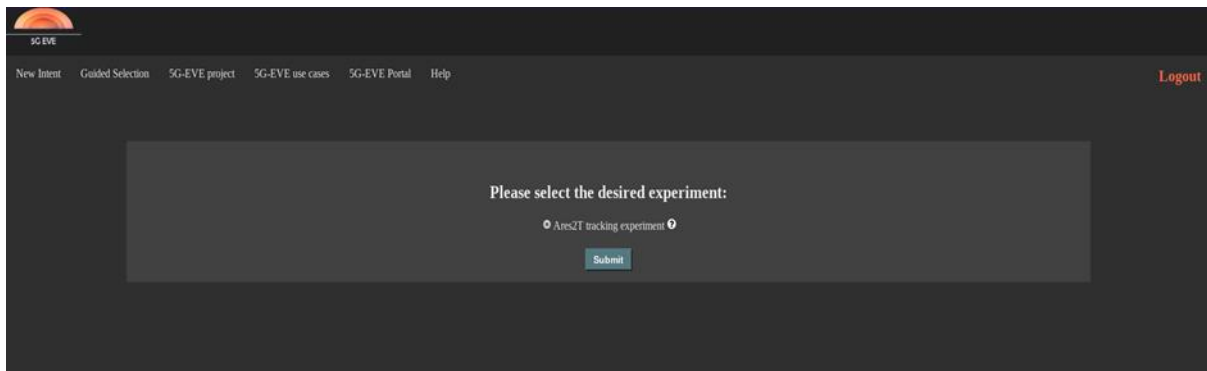
**Figure 61: 5G EVE Intent Guided selection-select site**

- The services that are supported in the selected site are shown in and the user chooses the suitable one.



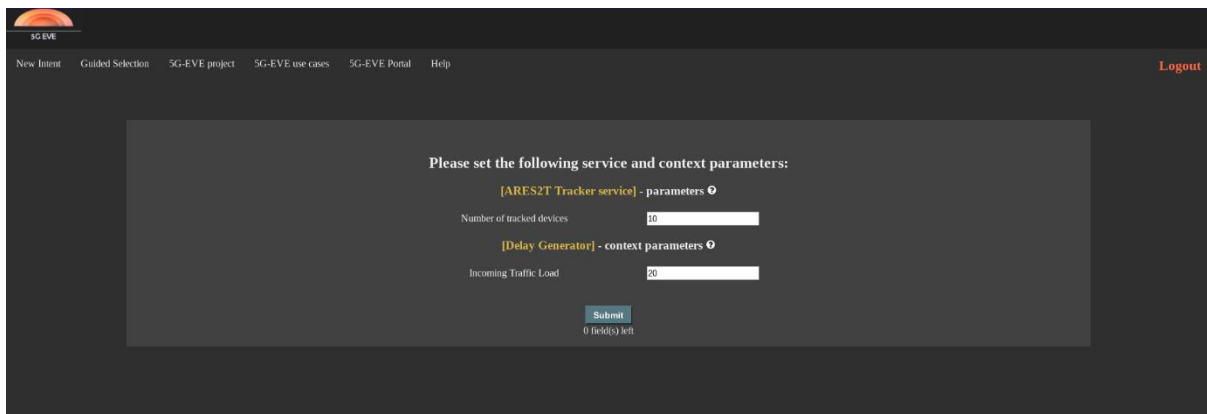
**Figure 62: 5G EVE IBN tool: Site- Available Services**

The experiments that contain the selected service are shown and the desired one must be chosen as illustrated in Figure 63.



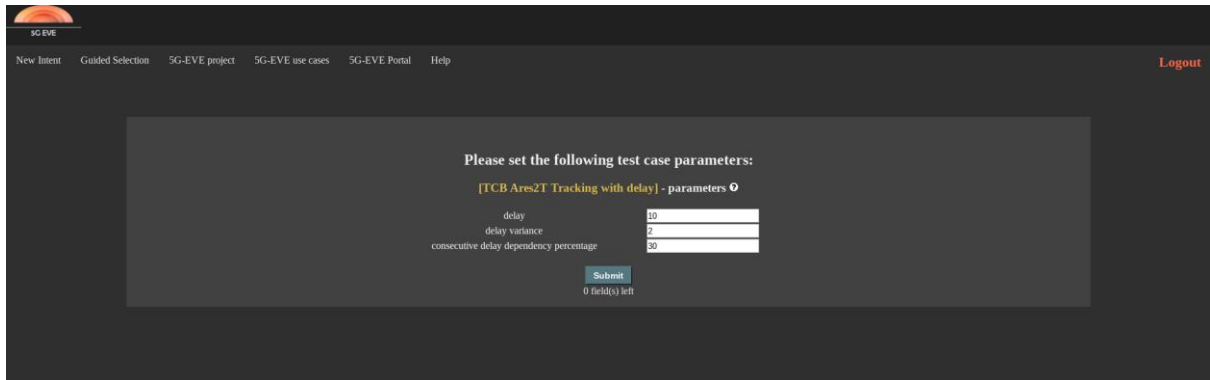
**Figure 63: 5G EVE IBN tool: Site- select desired experiment**

All the vertical service and context blueprint parameters are displayed, and the user is asked to fill in the desired values as shown in Figure 64



**Figure 64: 5G EVE IBN tool: Site- Select vsb and ctxb parameters**

All the test case parameters are displayed, per test case, and the user is asked to fill in the desired values as shown in **Figure 65**. If the user desires to exclude a test case, they may put the value "-" to a parameter of this test case.

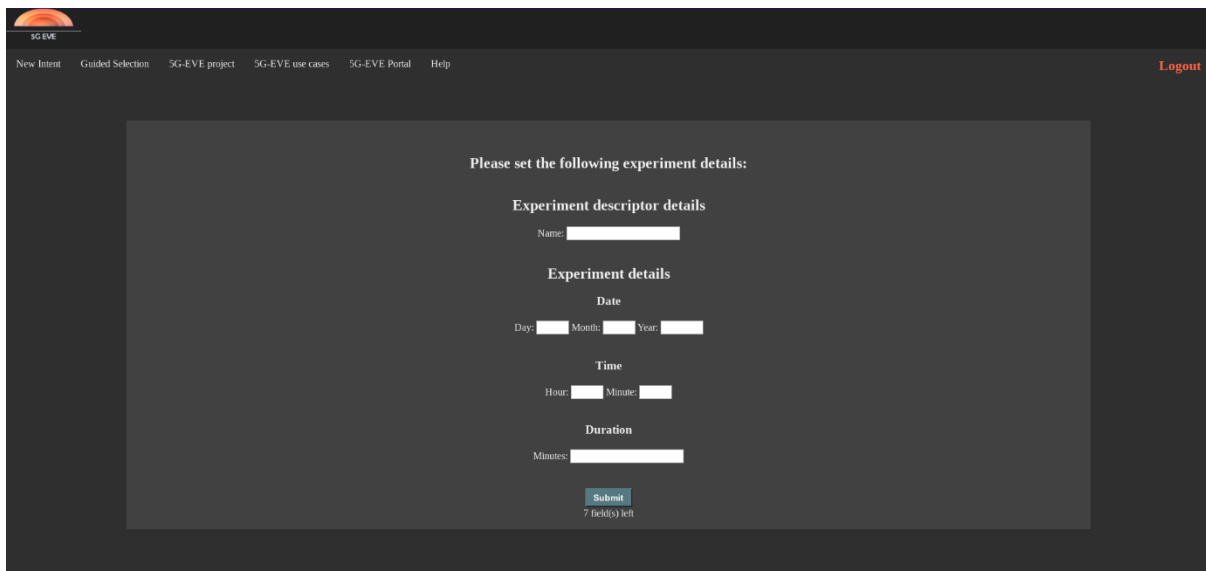

 The screenshot shows a web interface for the 5G EVE IBN tool. At the top, there is a navigation bar with links: "New Intent", "Guided Selection", "5G-EVE project", "5G-EVE use cases", "5G-EVE Portal", and "Help". A "Logout" link is in the top right corner. The main content area has a dark background and contains the text "Please set the following test case parameters:". Below this, there is a title "[TCB Ares2T Tracking with delay] - parameters 0". A table of parameters is displayed:
 

delay	<input type="text" value="10"/>
delay variance	<input type="text" value="2"/>
consecutive delay dependency percentage	<input type="text" value="30"/>

 Below the table is a "Submit" button and the text "0 field(s) left".

**Figure 65: 5G EVE IBN tool: Site- select test case parameters**

Finally, the date/time/duration parameters of the experiment are asked and if the input is valid the user can submit the form as shown in Figure 66.


 The screenshot shows the same web interface as Figure 65, but for scheduling experiment details. The text "Please set the following experiment details:" is at the top. Below it is the section "Experiment descriptor details" with a "Name:" label and an input field. The next section is "Experiment details", which includes:
 

- Date**: Three input fields for "Day:", "Month:", and "Year:".
- Time**: Two input fields for "Hour:" and "Minute:".
- Duration**: One input field for "Minutes:".

 At the bottom, there is a "Submit" button and the text "7 field(s) left".

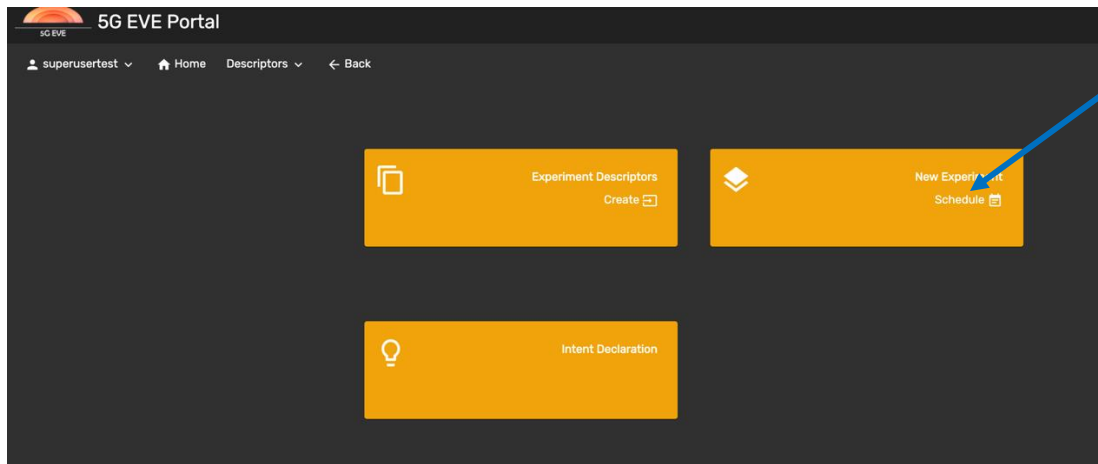
**Figure 66: 5G EVE IBN tool: Site-schedule experiment using the intent based tool**

- Once again, an experiment descriptor is created with all the information and is onboarded at the Portal Catalogue and the Experiment Lifecycle Manager (ELM) is notified about the desired experiment details and the scheduling takes place.

### 2.3.6 5G EVE Portal schedule experiment

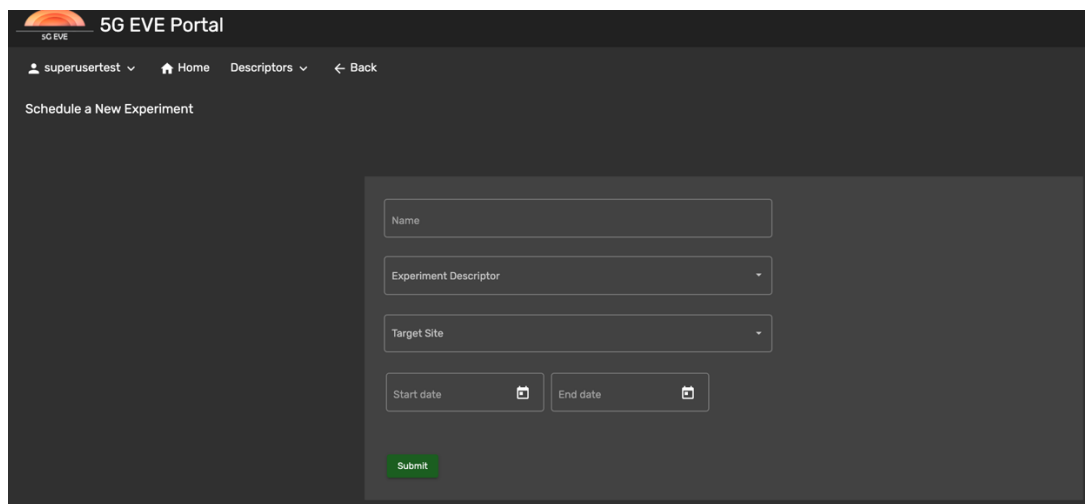
This option is only required for those who used the Experiment Descriptors create wizard to create their descriptors.

- For those who used the intent tool, the experiment was already scheduled using the intent tool when selecting the time, date and duration of the experiment.
- To schedule an experiment, experimenters have to click on the "New Experiment Schedule" section as shown in Figure 67.



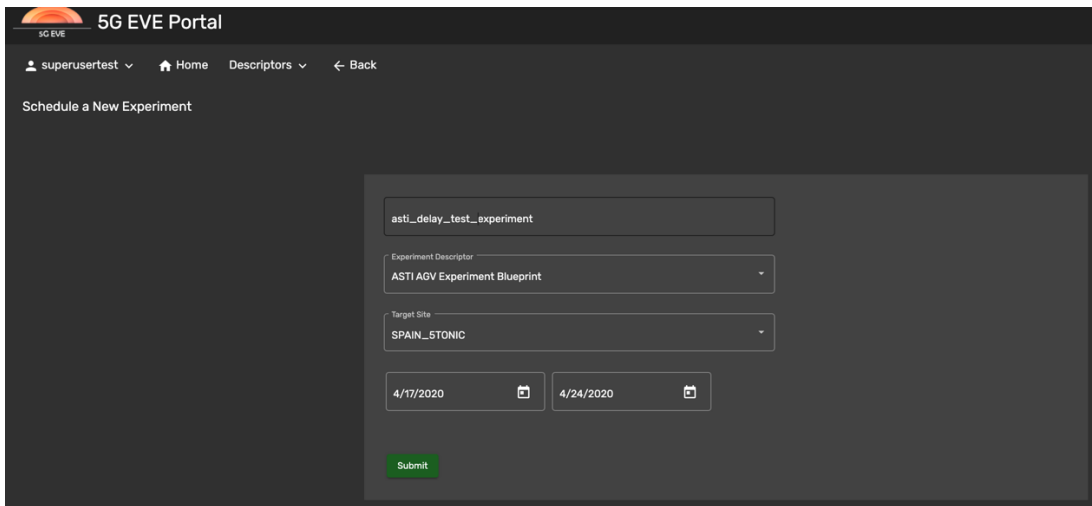
**Figure 67: 5G EVE Request Experiments page**

- On clicking the “*New Experiment Schedule*”, users are directed to the “*Schedule a New Experiment*” page given in Figure 68.



**Figure 68: 5G EVE Schedule a New Experiment**

- To schedule an experiment, users have to input the name of the experiment and then select the name of the Experiment Descriptor that they want to use by using the arrow icon.
- Thereafter, select the 5G EVE “*target site*” where experimenters want to carry out their experiment. This parameter corresponds to the value of the “*compatibleSites*” parameter inside the VSB.
- Then, please select the start and end date of the experiment.
- An example of a scheduled experiment for the ASTI Use-Case is illustrated in **Figure 69**.

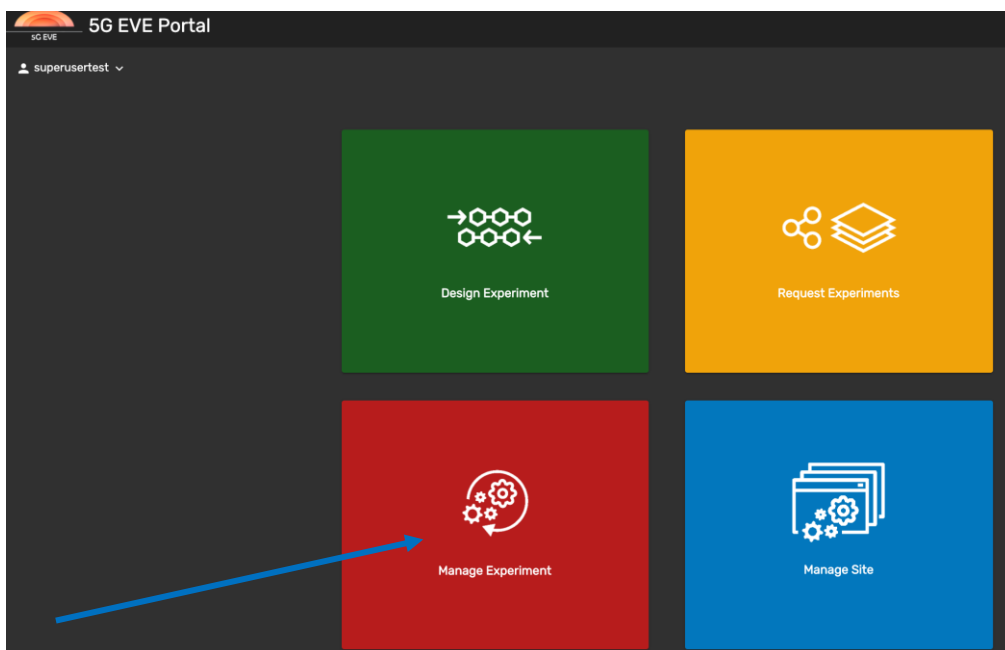


**Figure 69: 5G EVE ASTI Use-Case Scheduled Experiment**

- Finally, experimenters can click the “submit” button to schedule an experiment.

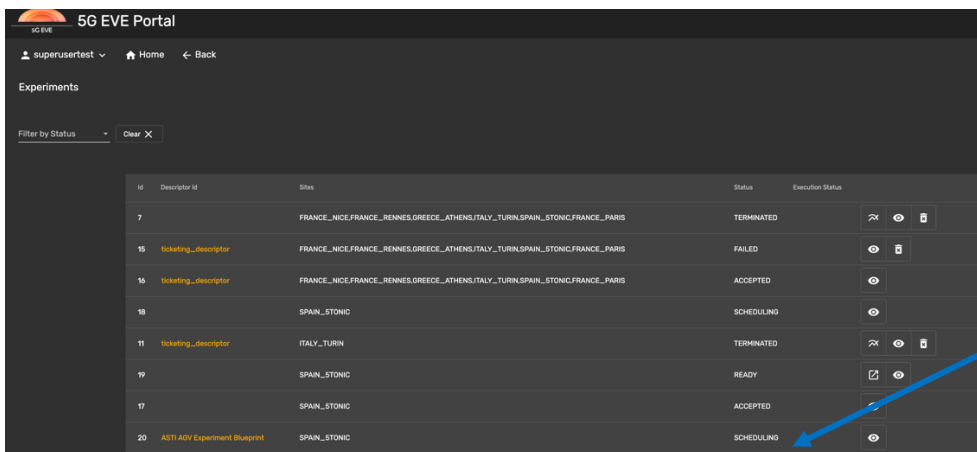
### 2.3.7 Manage experiment

To view the scheduled experiment, experimenters have to go back to the 5G EVE home page. Then, they have to click on the “manage experiment” section as shown in **Figure 70**.



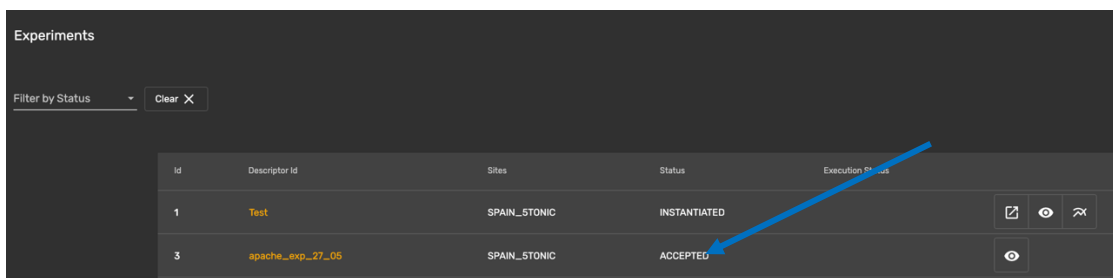
**Figure 70: 5G EVE Manage Experiment Section**

- Clicking on the “*Manage Experiment*” section, experimenters will see their scheduled experiment as shown in **Figure 71**.



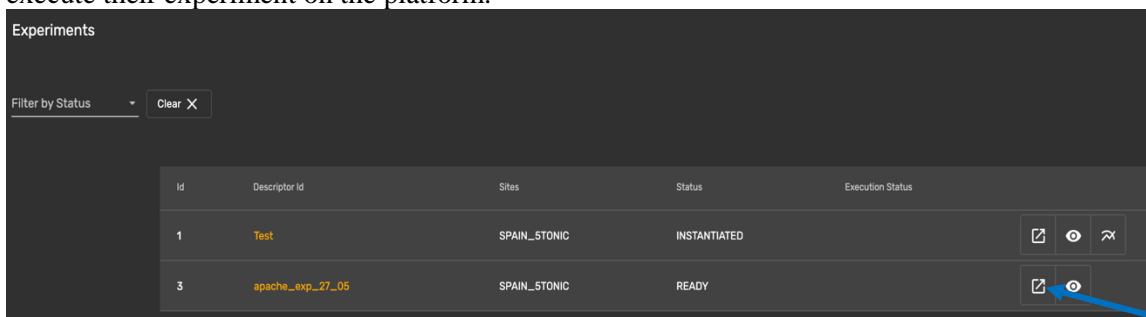
**Figure 71: 5G EVE EXPERIMENT LIST**

- At this stage, the site managers of the selected 5G EVE site will receive a ticket with the experiment request.
- If all the required experiment resources are available at the selected 5G EVE site and the timeline is not an issue, then the experiment will be accepted. Otherwise, it will be rejected with a reason for the rejection.
- Once an experiment is accepted in the 5G EVE portal, then its status will change from “*SCHEDULING*” to “*ACCEPTED*” as shown in **Figure 72**.



**Figure 72: 5G EVE Experiment Accepted**

- After an experiment is accepted on the portal, experimenters still need to wait for the experiment status to change to “*READY*” before being able to execute it on the platform. This is because in some vertical services, the site managers might need to do some manual configurations to prepare the 5G EVE sites for the experiment execution.
- Once the experiment status changes to ready as shown in **Figure 73**. Then, experimenters are ready to execute their experiment on the platform.

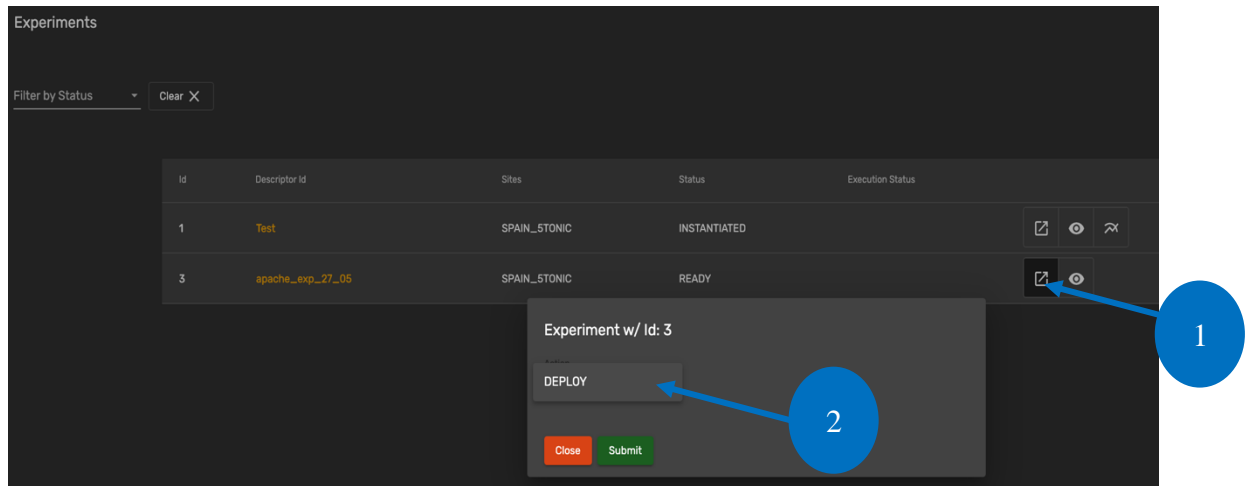


**Figure 73: 5G EVE Experiment Ready**

### 2.3.8 5G EVE PORTAL EXECUTE AN EXPERIMENT

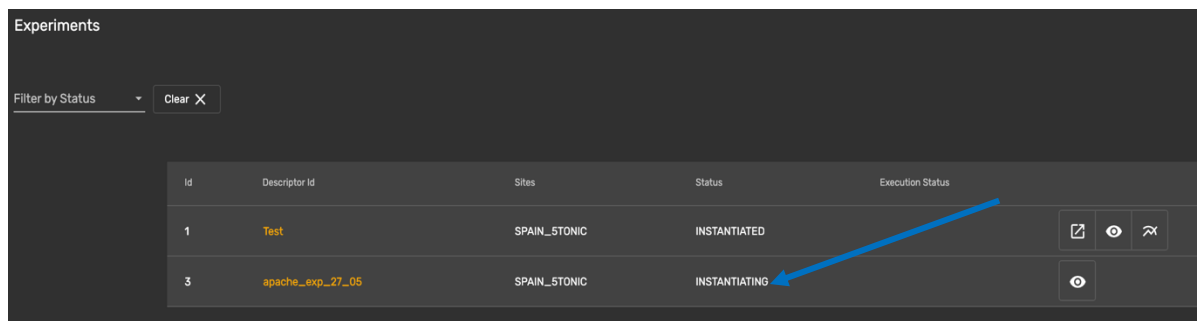
To execute an experiment on the 5G EVE platform, experimenters have to click on the icon indicated by the blue arrow in **Figure 73**.

- Clicking on that icon, experimenters will be presented with a screen similar to **Figure 74**. Users have to select “DEPLOY” and after click on submit to execute their experiment.



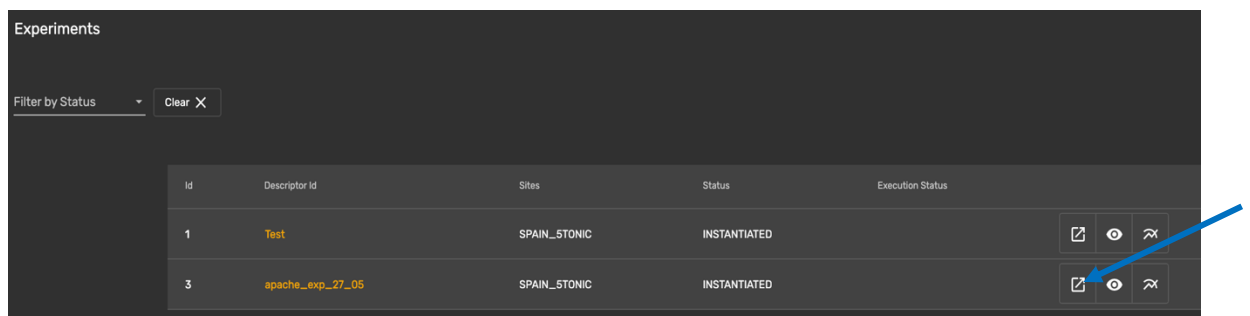
**Figure 74: 5G EVE Execute an experiment**

- On submission, the experiment will start executing on the portal, and the experiment status will change to “INSTANTIATING” as indicated in **Figure 75**.



**Figure 75: 5G EVE Experiment Instantiating**

- Once the platform is done instantiating the experiment, then the experiment status will change to “INSTANTIATED” as shown in **Figure 76** or “FAILED” in case the instantiation has failed for whatever reason.



**Figure 76: 5G EVE Instantiated experiment**

- To start executing tests on the instantiated experiment, experimenters can click on the icon indicated by the blue arrow in **Figure 76**.
- On clicking on that icon, experimenters will be presented by two options i.e., “EXECUTE” or “TERMINATE” as shown in **Figure 77**.



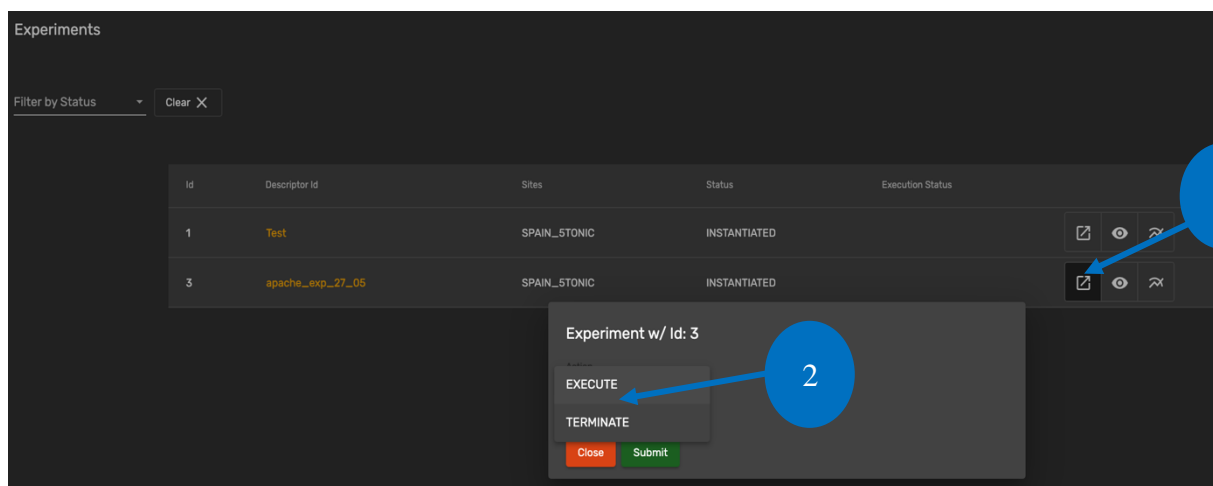


Figure 77: 5G EVE Execute experiment tests

- Experimenters have to select “EXECUTE” to start executing the test cases associated with their experiment. After selecting “EXECUTE”, they will be asked to input the name of their execution.
- When the experiment is executing, the experiment status will change to “EXECUTING” as indicated in Figure 78.

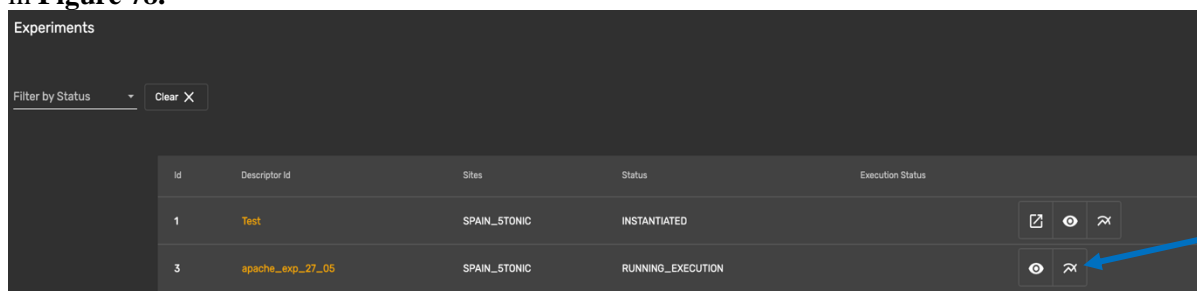


Figure 78: 5G EVE Experiment execution running

### 2.3.9 5G EVE PORTAL MONITORING & KPI REPORTING TOOLS

As the experiment is running, experimenters can view the monitoring metrics and 5G KPIs that they set in their context and experiment blueprints by clicking on the icon indicated by the blue arrow in Figure 78. For the simple Apache use case, the metrics and KPI graphs example are given in Figure 79.

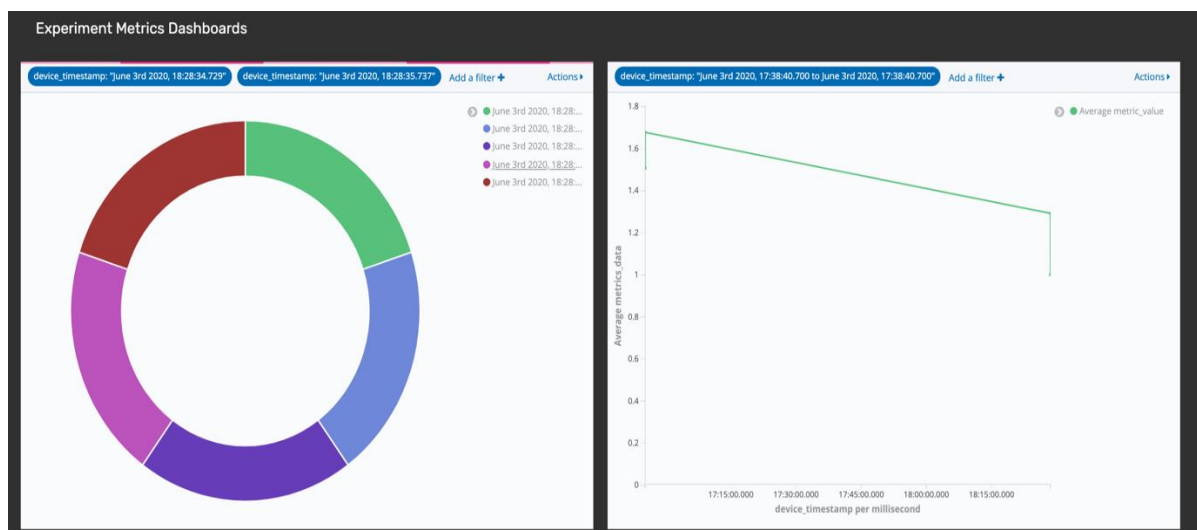
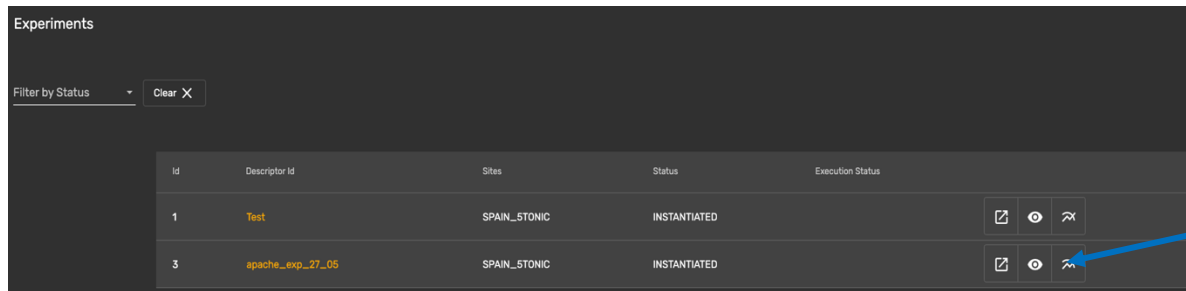


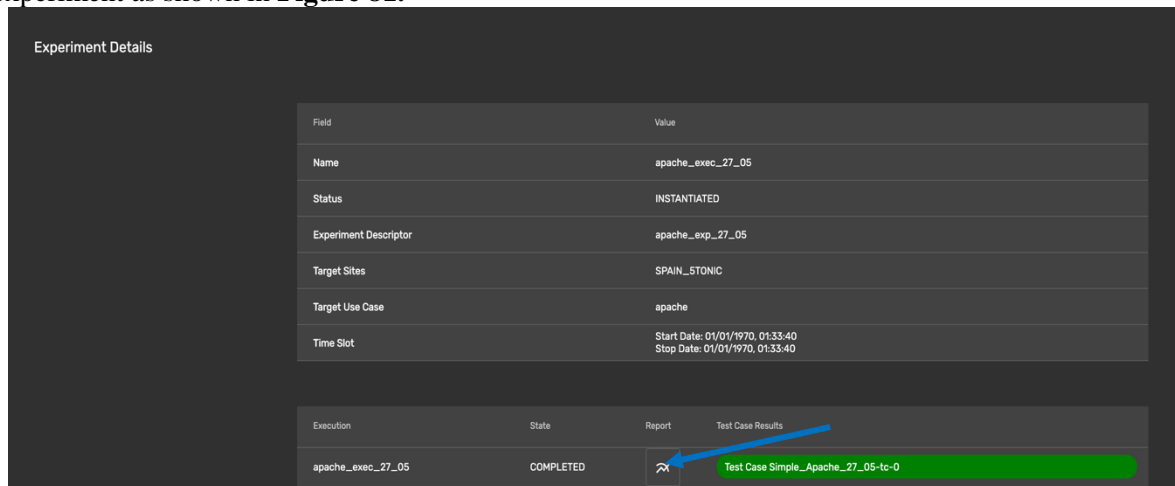
Figure 79: 5G EVE simple APACHE use case KPI and metrics graphs

- Once the experiment has finished executing, the status will change from “*RUNNING EXECUTION*” back to “*INSTANTIATED*”. To view the experiment tests details, experimenters can click on eye icon as shown in **Figure 80**.



**Figure 80: 5G EVE View experiment tests report**

- On clicking the eye icon, experimenters will be presented with some summary statistics about their experiment as shown in **Figure 81**.



**Figure 81: 5G EVE Experiment tests summary**

- To view the 5G EVE KPIs validation report for the tests carried out in an experiment, experimenters have to click on the report icon as shown in **Figure 81**. On clicking this icon for this simple use case, the validation report looks similar to the one indicated in **Figure 82**.

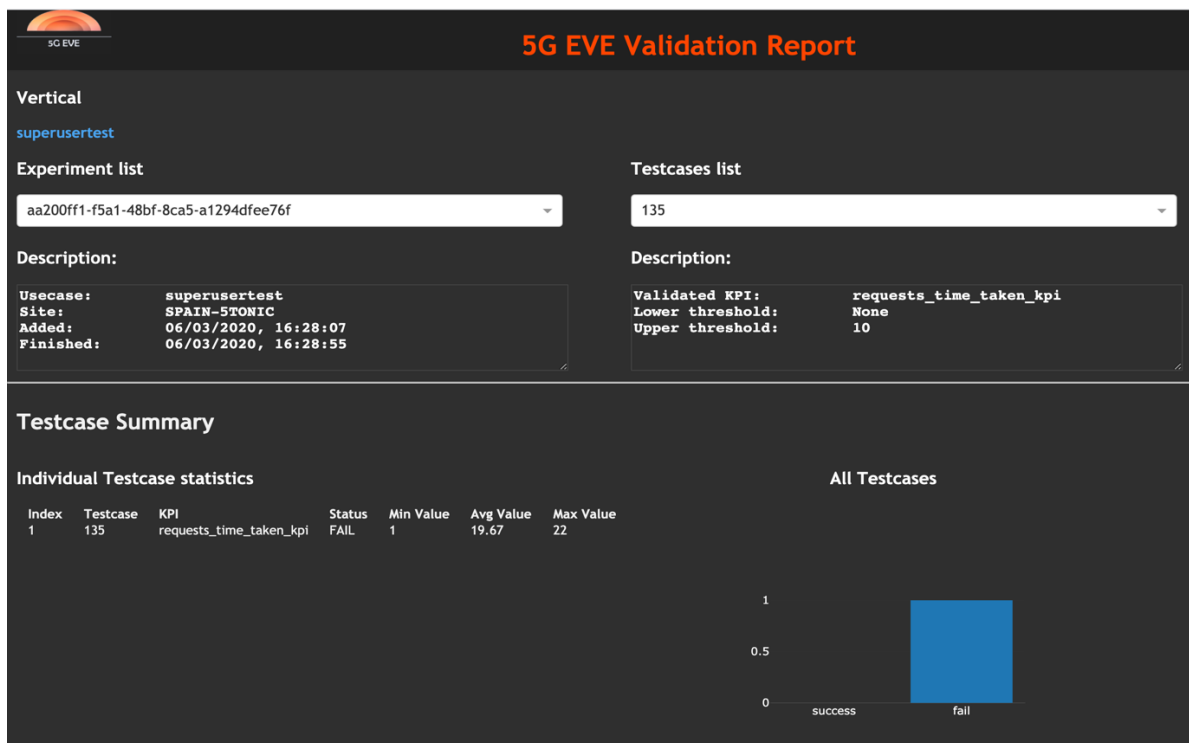


Figure 82: 5G EVE Validation report

- As you can see in **Figure 82**, the validated KPI i.e., *requests\_time\_taken\_kpi* is the KPI set when creating the experiment blueprint.
- And we can see that the KPI validation failed because we set the upper threshold (i.e. time taken by the requests KPI) to 10ms (refer to **Figure 83**) and yet the average time per request was **19.67ms** as indicated in **Figure 82**.



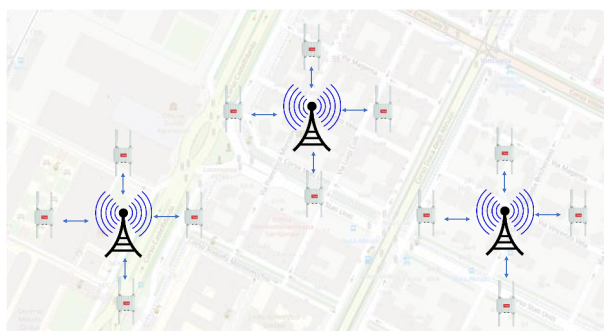
Figure 83: 5G EVE UC 5G KPI VALIDATION\_I

Finally, after executing the experiment and viewing the experiment report, it is important to go back to the experiments page and “*terminate*” your experiment so that your allocated use case resources can be available to the other 5G EVE use cases.

### 3 Stateful data planes to coordinate VNFs

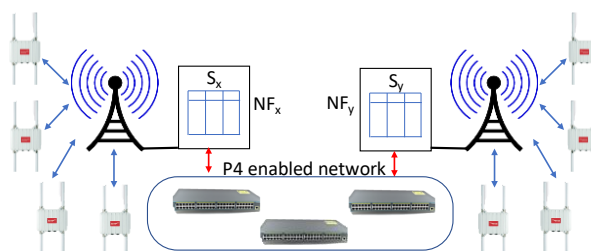
#### 3.1 Introduction

Advanced network applications are based on stateful virtual Network Function (NFs), i.e. an internal state is kept within the NF during the traffic operations. As a motivating example, consider the Smart City use case in 5G EVE project depicted in Figure 84. A set of WiFi scanners cover an area around an eNB and collect the information regarding the mobile devices of the people moving under their coverage. The WiFi scanners capture the Probe-Request messages periodically sent by the smartphones to advertise the list of the WiFi Access Points (APs) to which they have been connected in the past. From the MAC source address of the WiFi interface and the time when it was under the coverage of a WiFi scanner, it is possible to track the user mobility, in a completely transparent way for the users. In particular, as shown in Figure 84, we assume that the data collected by a one eNB is managed by a NF, which is responsible both for the collection and for the mobility tracking.



**Figure 84: Smart City UC in the 5G EVE project based on transparent WiFi scanners connected through eNB.**

In order to compute locally the mobility of a mobile user traversing the areas covered by different eNBs, the NF requires to know also the time at which the corresponding MAC address has been observed in the neighbouring eNB. For example, if the time at which the MAC was detected by the scanner under the coverage of eNB X is before the time during which it was detected by the scanner under the coverage of eNB Y, then it is possible to infer the direction  $X \rightarrow Y$  for the mobile user. The local information recorded by each scanner is a table with pairs of timestamps and MAC addresses and constitutes the local state of the corresponding NF that must be replicated to the other NFs. In the toy example of Figure 85, the state  $S_x$  of  $NF_x$  must be replicated to  $NF_y$  and, viceversa, the state  $S_y$  of  $NF_y$  must be replicated to  $NF_x$ . Replicating the local state of a NF is needed to infer locally the users’ mobility and the distributed nature of the approach naturally improves the overall scalability of the mobility tracking application.



**Figure 85: Stateful data plane for state replication in the Smart City use case**

Other motivating examples for state replication are applications as traffic classifiers, traffic shapers, firewalls, running as NFs. Scaling such network applications for large networks and/or for high data rates requires to replicate the same NF into different servers and to distribute the traffic across all the instances of the NF. The coordination between NFs requires that the internal states should be shared across the replicas. As a toy example, consider a distributed Deny-of-Service (DoS) detection application in which many replicas of the same NF are distributed at different ingress routers of a network. The detection is based on evaluating the overall traffic

entering the network from all edge routers. This application requires to share the metrics of the local traffic among the NF replicas in order to compute the network-wide traffic.

A possible solution for state replication is to implement a standard replication protocol directly in the NF (like Paxos, RAFT, etc), but this would require loading the NF with also this replication process, which can be quite complex and compute-intensive. An alternative solution is to leverage a stateful data plane, e.g., based on P4 [5]. This implies that the state replication is offloaded from the NFs to the P4 switches, which take the responsibility of coordinating the exchange of replication messages between NFs, with a beneficial effect on the NF load and thus on the overall scalability. In particular, the 5G EVE project has investigated how to implement a publish-subscribe scheme directly on P4 switches, according to which the NFs can publish the updates on their internal states and can subscribe on the updates from the other NFs. The proposed solution is denoted as STAtE REplication (STARE), based on a publish-subscribe paradigm. STARE allows to achieve a state replication process which is “light” for the NFs and that exploits the high processing speed of P4 switches. Furthermore, the SDN controller is not involved in the overall publish-subscribe operations, except during the initialization of the NF, and the experienced processing latencies are negligible with the one that would be needed to interact with an SDN controller.

In the following, Section 3.2 discusses the related works. Section 3.3 presents the proposed architecture for STARE and explains the integration within 5G EVE project for a specific use case. The two key implementation elements are the middleware present in the NFs, described in Section 3.4, and the P4 replication engine present in the switches, described in Section 3.5. We present the experimental evaluation of the proposed solution in Section 3.6. The final sections (Secs. 3.7-3.11) can be seen as appendices and report the detailed specifications of the proposed protocols and packet formats, and can be referred by the developers. Finally, the code for the proposed solution is open source and made available on github [6].

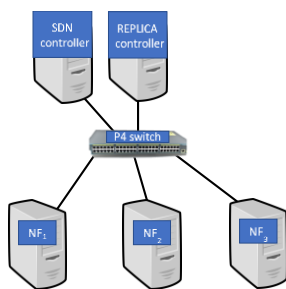
## 3.2 Related works

The problem of replicating states has been previously addressed only in the context of the data plane. Indeed, programmable P4 switches are stateful and the applications may operate on some states that are global, i.e., shared across multiple switches. This scenario is similar to the scenario of the NFs sharing some internal states across multiple NFs. The work in [7] has addressed the problem of how to optimally place the state within a P4 programmable data plane, taking into consideration both the data traffic (which must be eventually reroute to guarantee the correct behavior of the network application) and the replication traffic (necessary to replicate the state across the switches). The work has proposed a framework to optimize the number of replicas and their placement within the network, taking into account the main tradeoff between data traffic and replication traffic. Indeed, a larger number of replicas increases the replication traffic but minimizes the data traffic, affecting the overall network congestion and thus the performance. Differently from [7], STARE does not optimize the placement of the NFs within the network, even if this issue could be considered as a future activity. In terms of implementation, the work in [8] has proposed a programming abstraction to define the states to be replicated, but, differently from STARE, it focused just on internal states available at the switches. Furthermore, [8] has shown a P4 implementation of the state replication across the switches, but only in the data plane, without any interaction with possible NFs sharing states. The adopted consistency model for the state replication adopted in STARE is the same as the one considered in [8], for which the P4 implementation on the data plane provides a prototypal idea for the implementation of the publish/subscribe scheme in STARE.

## 3.3 STARE architecture and protocol

We consider a set of NFs that share a local state, that must be replicated across all the NFs. Even if the following description refer to a single state, the extension to multiple states is immediate. The NFs are running on servers connected through a network in which the switches are fully programmable with P4. Each server may host more than one NF. Each NF is not aware of the existence of the other NFs.

Our proposed solution is based on a publish-subscribe scheme that that allows an asynchronous state replication between the NFs. As shown in Figure 86, the main entities involved in the protocol are (1) the NFs acting as clients for the publish-subscribe scheme, (2) the P4 switch acting as broker for the publish-subscribe scheme, (3) the REPLICATION controller which coordinates the overall process.



**Figure 86: STARE architecture for state replication**

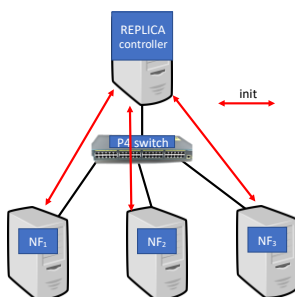
The protocol works in two phases:

- *Initialization phase:* The NF interacts with the REPLICA controller in order to get a unique identifier and to get the identifiers of the subscribed/published state variables. These identifiers are globally unique within the network and are required for the correct behavior of the STARE protocol.
- *Replication phase:* Anytime the state is locally updated at some NF, this NF sends a publish message with the state updates directly to the local P4 switch. The P4 switch, acting as a broker, has the responsibility to send the updates to all the ports where at least one NF has subscribed.

Notably the REPLICA controller is referred just during the initialization phase and not during the state replication, which is instead completely offloaded to the network. In the following we will refer to state variable or simply variable as the generic state to be replicated. With an abuse of notation, a state variable can include also “complex” data structures as tables, list, etc.

Different kinds of messages have been devised to be exchanged to replicate the state variables, here summarized at high level. For further details, Section 3.7 provides the specifications for the STARE protocol and for the adopted message format. As shown in Figure 87, during the initialization phase:

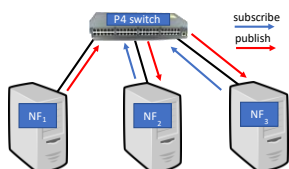
- the NF interacts with the REPLICA controller to get its global identifier;
- the NF interacts with the REPLICA controller to get the identifiers of the local state variables on which the NF will publish the updates;
- the NF interacts with the REPLICA controller to get the identifiers of the remote state variables on which the NF will subscribe the updates.



**Figure 87: Initialization phase according to the STARE protocol**

As shown in Figure 88, during the replication phase:

- the NF sends directly to the P4 switch the subscription messages;
- the NF sends directly the publish messages to the P4 switch with the updates which will be forwarded directly from the switch to the subscribed NFs.

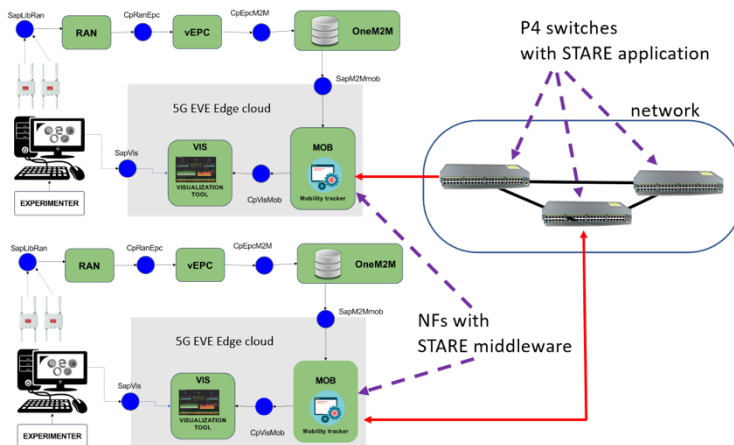


**Figure 88: State replication phase according to the STARE protocol**

The two main software components implementing the proposed architecture are the following:

- the *STARE middleware*, which runs in the servers hosting the NFs and acts as a kind of proxy between the NFs and the network; it is described in Section 3.4.
- the *P4 STARE application*, which runs in the programmable P4-enabled switches and provides the replication engine directly in the data plane of the network; it is described in Section 3.5.

In Figure 89 we show the integration of our proposed STARE architecture within the 5G EVE project, based on the general architecture depicted in Figure 86, in the context of the Smart City: Safety and Environment - Smart Turin use case, described in Deliverable 2.4. We show an example with two edge clouds, located in different eNBs, but the solution works for a generic number of NFs. In particular, a NF (denoted as “VIS”) provides the visualization tool for the experimenter, whereas another NF (denoted as “MOB”) is responsible for tracking the mobile users’ mobility by processing the live data feed arriving from the WiFi scanners. Each MOB NF must replicate its local table storing the coverage time of any observed MAC to all the neighboring MOB NFs, present in other edge clouds. The network connecting the edge cloud is assumed to be fully programmable, leveraging P4 switches, and thus enabling the adoption of STARE.



**Figure 89: Integration of STARE architecture for the Smart City: Safety and Environment - Smart Turin UC**

### 3.4 STARE Middleware

The replication protocol runs in the network, and the NFs are not directly involved in it. This approach allows to reduce the computation overhead to manage complex replication algorithms and to simplify the development of NF applications. The NFs interact among each other throughout a middleware module, as shown in Figure 90, which is the single endpoint of the replication scheme within each server, independently from the number of VMs and NFs running on the server. STARE middleware acts as a proxy between the local NFs and the other servers and thus it can group the different publish/subscribe/unsubscribe messages in order to reduce the number of messages exchanged in the network. The main tasks of the STARE middleware are the following:

- acting as a proxy for the replication protocol between the NFs and the network.

- acting as a proxy for the replication protocol between the NFs within the same server. This allows to share the state variables between NFs locally, within the same server, without the interaction with the network.
- forwarding in a transparent way all the in-coming and out-going traffic that is not related to the STARE protocol. This allows the NFs to behave as expected.

We implemented a simple protocol to allow the interaction between the NF and STARE middleware. The messages are mimicking the behavior of the publish/subscribe protocol adopted at network level by the P4 switches. The detailed specifications of the protocol and the messages are reported in Section 3.8. The software architecture of the Middleware is shown in Figure 91, but, for the sake of clarity, all the details have been presented in the Section 3.10.

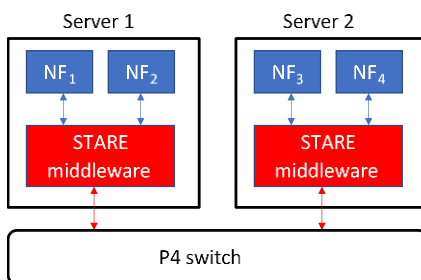


Figure 90: STARE middleware in a simplified scenario with 4 NFs, 2 servers and 1 P4 switch

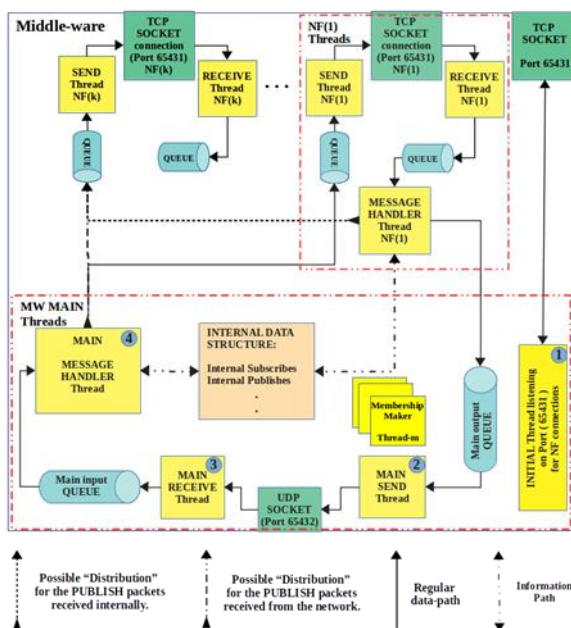


Figure 91: The software architecture for STARE Middleware

### 3.5 The STARE P4 application in the network switches

The switches are programmed through the P4 abstraction to provide the main replication engine directly in the network. Two possible implementation designs have been considered, described in the following two sections. Both implementations assume that the SDN controller has already setup a distribution tree spanning all the switches to which the servers running the NFs are connected, thus all the publish and subscribe messages are sent along such distribution tree.



### 3.5.1 Register-based P4 implementation

Three main messages are processed in the P4 switch: the subscribe, the unsubscribe and the publish messages. These are distinguished through 2 specific bits present in the IP destination address.

The main data structure implemented in the P4 switch is the state forwarding table, which maps the variable id to the destination bitmask corresponding to the group of switch ports where a publish message with such variable id must be forwarded. The table is implemented with an array of registers, where register k is responsible for storing the destination bitmask of the variable id equal to k. The number of bits for each register is equal to number of ports of the switch. The following table shows an example of the state forwarding table for an 8 × 8 switch, with 3 stored variables. For example, according to it, any publish for variable id equal to 1 must be forwarded to ports 1, 5 and 7.

Register (Variable-ID)	Destination bitmask
0	00011110
1	10001010
2	10000000

We now describe at high level how a packet is parsed and processed by a P4 switch according to the register-based P4 application of STARE. We refer to the implementation available in [9]. Packets belonging to the STARE protocol are identified checking the IP multicast destination address and the UDP port (equal to 65432), coherently with the packet format described in Section 3.7. For each subscribe message, the P4 switch generates the input-port bitmask (equal to  $2^i-1$ , where i is the port identifier) and uses the variable-id as a key in the state forwarding table to retrieve the current destination bitmask. Now an OR operation with the input-port bitmask allows to obtain the new destination bitmask. The P4 switch swaps the IP source and destination addresses and mirrors back to the port the received packet to reply back and to confirm the subscription to the generating NF. Furthermore, a copy of the original packet is sent to all the ports corresponding to the distribution tree, except the incoming port. A similar processing occurs for an unsubscribe message, but now a combined AND/NOT operation is required to delete the previous subscription. To process a publish message, the P4 will index the state forwarding table based on the enclosed variable id and will send a copy of the message to all the ports corresponding to the destination bitmask. Finally, all the other kinds of messages used in STARE protocols are sent to the port corresponding to the REPLICA controller.

### 3.5.2 Local controller-based P4 implementation

In this version of the application, the switch devotes a specific match-action table to store the state forwarding table. The entries of the tables are modified by a local embedded controller inside the switch. Notably, it is not possible to modify the tables directly through a P4 application, since P4 describes the structure of the tables and not how to populate them on live. Thus, the use of a controller is required to modify the table. We used P4Runtime API to define the protocol to interact with the table defined through our P4 program.

Match	Action
Variable-id=0	Forward to ports: 4,5,6,7
Variable-id=1	Forward to ports: 1,5
Variable-id=2	Forward to ports: 1

Anytime a subscribe or unsubscribe packet arrives at the switch, the switch forwards it to the local embedded controller, which modifies accordingly the entries in the switch match-action table through P4Runtime API. This requires the controller to have an internal data structure to mirror the match-action tables inside the switch. Furthermore, the controller sends back a copy of the subscribe/unsubscribe message to the switch in order to be forwarded to the other ports of the switch. In more details, the interaction between the P4-switch and the local P4-controller is shown in Figure 92 and described as follows:

- **(1)** When a subscribe or unsubscribe message arrives at the switch, it will be sent to the local embedded controller through a Packet-In message, whose header contains the incoming port of the packet.
- **(2)** The controller updates the mirrored version of the switch match-action table and sends the corresponding instructions to update the switch tables.
- **(3)** The switch acknowledges the configuration to the local embedded controller.
- **(4)** Now the local controller sends the subscribe packet received at **(1)**, after swapping the IP source-destination addresses to the switch, through a Packet-Out message, which will be forwarded to the incoming port of the original packet. Furthermore, a copy of the original subscribe packet is sent to the ports belonging to the state distribution tree.
- **(5)** The switch sends the packet carried by the Packet-out message to the destination ports indicated in the state forwarding table.

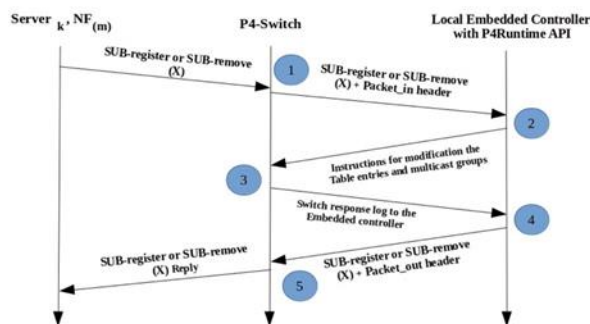


Figure 92: Example of interaction for the local controller-based P4 implementation

### 3.5.3 Comparison between the two implementations

The register-based implementation is simple and relies on a static number of registers, defined at compilation time. This implies that anytime the number of state variables exceed the available ones, it is necessary to recompile and reinstall the P4 STARE program again. In theory, this reconfiguration can be operated on live on specialized hardware.

The main advantage of the local-controller based solution with respect to the register-based one is that the table size, in terms of stored states, is independent from the width used to represent the variable id, giving higher level of flexibility. In other words, the maximum number of states and their representation are completely decoupled. On the contrary, in the register-based application the table size is always equal to the number of defined registers, independently from the number of recorded states. Notably, the maximum number of registers depends on the particular hardware architecture and is typically limited, whereas the size of match-action table depends on the actual use of the available memories. In terms of performance, the processing time of STARE messages of the local-controller solution is higher than the register-based solution because of the time required to access the match-action tables and the one required to interact with the embedded controller, which introduces a higher degree of unpredictability with respect to a pure hardware based solution. It is worth to highlight that both schemes do not require the interaction of the switch and the NF with the REPLICATOR controller and the SDN controller, thanks to the offloading process on P4 switches.

Finally, the maximum number of variable-id is always upper bounded by  $2^{22} \approx 4 \times 10^6$ , due to the adopted representation in the IP destination address, as described in Section 3.7.

### 3.6 Experimental validation

#### 3.6.1 Methodology

We used Mininet [10], a network emulator which provides software models for vanilla SDN switches and for P4 switches, running on 5G EVE edge cloud. In particular, we used Open vSwitch [11] to run the experiments based on standard OpenFlow switches, and we used BMV2 [12] virtual switch to run the experiments based on P4 switches. P4 version 16 has been used. We used Ryu [13] as SDN controller to interact with the Openflow and P4 switches. We chose Ryu for its simplicity of deployment and for its wide support of OpenFlow standards (up to Openflow 1.5). The software implementation was developed in Python and implemented through the standard Python socket library. All the experimental testbed was deployed on a Ubuntu 18.04 Virtual Machine. For the register-based solution, we defined 2049 registers inside the P4 program at the compile time.

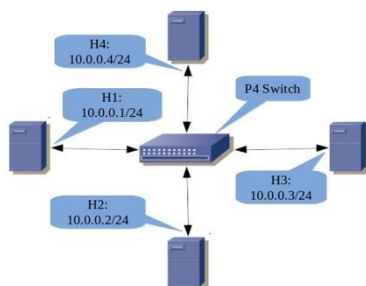
To measure the resources required in the P4 switches and compare the different solutions, we evaluated the memory occupancy in terms of Resident Set Size (RSS) associated to the process running the virtual switch. To evaluate experimentally the RSS, we used ps command for the Local Embedded-controller implementation and the time -v command for the register-based implementation.

#### 3.6.2 Experimental scenario

Figure 93 shows the topology we considered, which includes 4 NFs and one P4 switch. The NFs are emulated through Mininet “hosts” on which we developed the middleware in python, using the standard TCP socket libraries provided by the language. We deployed some prototypal NFs that track the mobility of users as shown in Figure 85. In more details, the NF receives the feed of WiFi sensor in JSON format every two minutes and updates a local table with the structure shown as follows, which represents the internal state of the NF.

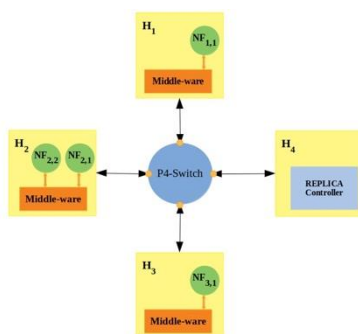
Time	WiFi scanner id	MAC address
...	...	...
...	...	...

Anytime the table is updated (typically with multiple entries added in the table), the new state must be replicated across all the other NFs. In order to maximize efficiency, only the new entries are sent within the publish messages.



**Figure 93: Emulated topology for the experimental evaluation.**

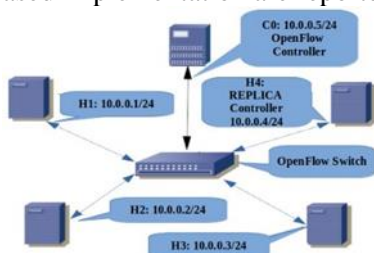
In our testbed, the REPLICATOR controller runs in H4 and STARE middleware runs in H1, H2 and H3 hosts, as shown in Figure 94.



**Figure 94: NFs, middleware and REPLICA controller running in the STARE testbed**

We start only one NF in H1 and H3, and two NFs in H2. To emulate the MQTT messages arriving from the WiFi scanner, every 20 seconds a random number of MAC/time pairs (between 10 and 50) are generated by each NF.

As a term of comparison, we considered a pure OpenFlow implementation, based on OpenFlow 1.3. A simple topology for this scenario is shown in Figure 95. The main idea is that the SDN controller takes the responsibility of keeping the state of publishing and subscribing NFs, thus the switch must interact with the SDN controller anytime a new subscription occurs. In addition, the initialization phase, required to assign the NF with its Global-ID and to know the Variable-ID of the state to publish and subscribe, is managed centrally by the controller. All the subscription messages received by a switch are sent to the controller, which will program the switch flow tables in order to associate the corresponding incoming port of the switch as the forwarding port for all the corresponding publish messages. This allows to avoid the interaction with the controller whenever a publish message arrives, since the packet will be directly sent to all the switch ports corresponding to subscribing NFs. The details of the OpenFlow-based implementation are reported in Section 3.11.



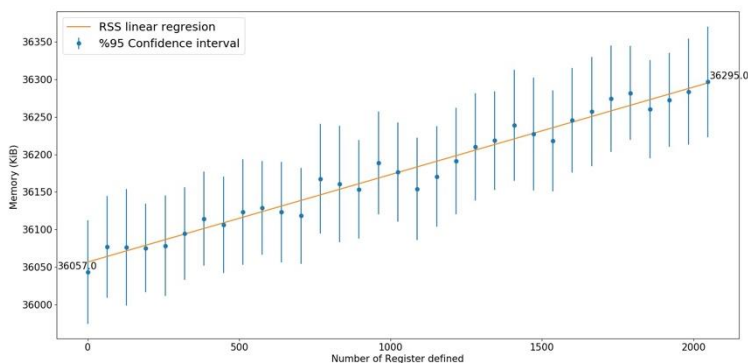
**Figure 95: Pure OpenFlow scenario**

### 3.6.3 Experimental results

We now compare the behavior of STARE with the solution based on OpenFlow switches in terms of internal resource consumption within the switch.

#### Registered-based P4 solution

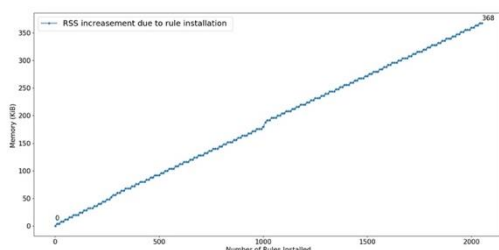
Figure 96 reports the average RSS memory, including the 95% confidence intervals and a linear regression over the average values. The achieved accuracy is very high, since the relative width of the confidence interval is around 0.2%. From the figure, the average amount of memory is 118.5 bytes for each rule, which is coherent with the fact that each register is mapped to a 32-bit length memory area after compilation, regardless of the programmer register length definition in the main P4 program. Also, it uses a 32-bit number as to index the mentioned register.



**Figure 96: RSS measurement for the register-based solution**

**Embedded controller-based P4 solution**

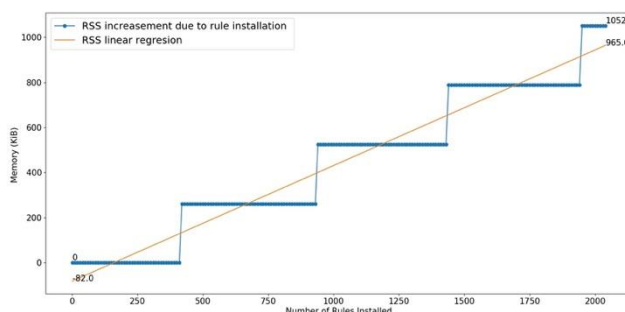
In the adopted match-action table, we started to evaluate the width of the key. In the flow match-action table, we use an exact matching on the IP address, so 32 bits are needed, and the corresponding action is an integer number representing the multicast group, which adds other 32 bits, thus we can expect a minimum of 64 bits for each rule. We used one NF inside one host of the Mininet and one P4 switch. The result of the measurement is shown in Figure 97, which shows that the memory occupancy increases linearly with the amount of installed rules and that the average memory occupancy is 184 bytes per installed rule.



**Figure 97: Increase in occupied memory for the embedded controller-based solution**

**OpenFlow-based solution**

We evaluated the RSS of the OVSK process and the measurements are collected by the remote SDN controller. The result of the measurements is shown in Figure 98, which shows a step-wise increasing function, due to the internal memory allocation scheme for the tables which works with a batch allocation process in the memory. Now the average occupancy is 523.5 bytes per installed rule.



**Figure 98: Remote Controller-based RSS measurement**

**3.6.4 Comparing the solutions**

As a summary, the memory occupancy for the three scenarios above is shown in the following table:

Scenario	Average memory per rule
Register-based P4 solution	118.5 bytes
Embedded-controller P4 solution	184.0 bytes
OpenFlow-based solution	532.5 bytes

The most efficient solution in terms of memory is based on registers, whereas the least efficient solution is the one based on a standard OpenFlow switch. The differences are due to the internal memory management process internal to the BMV2 software switch (for both P4-based solutions) and to the Open vSwitch software switch (for the OpenFlow-based solution). Notably, the performance in terms of latency are not reported here since the P4-based solution are much faster and not really comparable with respect to the OpenFlow solution, since the latter requires the switch to interact with a remote SDN controller anytime a subscription message is received. Instead, all the publish/subscribe messages between NF are managed internally at the P4 switch, with a processing delay which is negligible with respect to the latencies between the OpenFlow switch and the SDN controller.

### 3.7 STARE protocol and message specifications

All publish/subscribe messages are carried by UDP packets sent to port 65432. The corresponding IP packets are destined to a multicast IP address within the Locally-Scoped MultiCast Address range (starting from 239.0.0.0 to 239.255.255.255). The value of the destination address is carrying the control information required for the replication process. Figure 99 shows the destination address format.

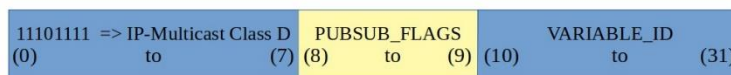


Figure 99: IPv4 destination address format for the state replication

The first 8 bits are constant to identify the address family, the next 2 bits are flags to identify the message kind (as described in the following table) and the remaining 22-bits carry the variable-ID. We defined the following messages: (1) INIT-NF-ID request, (2) INIT-PUB-VAR-ID request, (3) PUBLISH, (4) SUB-VAR-ID request, (5) SUB-Register, (6) SUB-Remove, (7) RECOVER. The INIT-NF-ID request, the INIT-PUB-VAR-ID request and the SUB-VAR-ID request are used in the initializing phase and exchanged between the NF and the REPLICATION controller. The SUB-register, SUB-remove, PUBLISH, and RECOVER messages are instead used in the data replication phase, as described in Section 3.3. The corresponding packet at MAC layer is sent in broadcast with destination ff:ff:ff:ff:ff:ff. In the following, we describe in details the two different phases of STARE protocol.

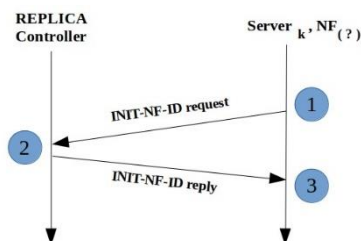
PUBSUB FLAGS	Message
11	SUB-register
10	SUB-remove
01	INIT-NF-ID-request, INIT-PUB-VAR-ID request, SUB-VAR-ID Request, RECOVER
00	PUBLISH

#### 3.7.1 Initializing phase

Two main problems are solved during this initial phase. First, the NF must be uniquely identified and, second, the state variable must be also uniquely identified within the network. The REPLICATION controller is responsible for uniquely choosing both identifiers as follows. Anytime a new NF is started, it enters the initializing phase during which a globally unique NF identifier, denoted as Global-ID, is assigned by the REPLICATION controller. The NF uses the INIT-NF-ID request to ask for its Global-ID from the REPLICATION controller, as shown in Figure 100.

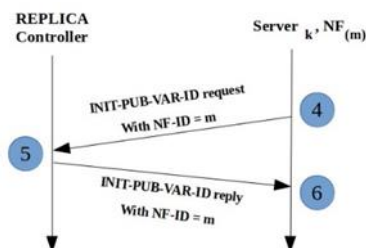
- **(1)** The just-started NF sends an INIT-NF-ID request to the REPLICATION controller.

- **(2)** The REPLICA controller replies with the assigned Global-ID to the NF.
- **(3)** The NF stores its Global-ID.



**Figure 100: Information Request phase to get the NF identifier Global-ID during the Initializing phase**

The NF publishes or subscribes on a state variable through a INIT-PUB-VAR-ID Request or a SUB-VAR-ID Request sent to the REPLICA controller. These messages carry some descriptive information about the variable names to identify the state variable or other information that may be needed in the REPLICA controller and can be in any format, e.g., JSON. On the other hand, the replies from the REPLICA controller should contain the same information and the assigned values to the requested information. Figure 101 shows the corresponding exchange of messages.



**Figure 101: Information Request phase to obtain the variable-id in the Initializing phase**

Assume that the initialized NF wants to publish the new value for a variable or it wants to subscribe on the updates, so it needs to know the corresponding assigned variable-id in the network (if any).

- **(4)** The NF sends a INIT-PUB-VAR-ID Request to the REPLICA controller, containing its Global-ID.
- **(5)** The REPLICA controller receives the request. If it is a request for a new state variable to publish, the REPLICA controller will assign a new variable-ID and send it to the NF. If it is a subscription to a variable-ID, the REPLICA controller will provide it, if available.
- **(6)** The NF stores the requested identifier.

### 3.7.2 State replication phase

When the NF wants to publish an update of a state, or to subscribe to a state, it uses the variable-id received during the initialization phase. Two kinds of messages are exchanged, whose format was shown before:

- SUB-register messages allow to register the subscription on a specified variable-id and SUB-remove messages allow to delete the subscription. Notably, the application layer payload is null.
- Publish messages are UDP packets sent from a NF to publish an update of a state variable. The application layer payload has a variable-size header (denoted as Publish-Header) and the remain part is devoted to carry the values carried by the update, as shown in Figure 102. The Publish-Header carries four fields: variable-ID with the corresponding variable id of the message, update-number which is a counter to distinguish between consecutive state updates, fragment-sqn which

is sequence number of the segment, and tot-fragments which is total number of segments of a message. Note that the last two fields are used in the case the publish message must be segmented into multiple IP packets and this possibility is required to manage updates of states with a large memory footprint (e.g., as in the case of large tables).

- RECOVER messages to notify the REPLICA controller about an incomplete delivery of the publish messages and to recover from this event. The detailed implementation is application specific.



Figure 102: Publish-Header at application layer

### 3.7.3 The P4-based broker

During the Initializing phase, the NF becomes aware of its NF-Global-ID and of the variable-id regarding the state to publish and to subscribe.

Figure 103 shows the main interaction between the NFs and the P4 switch acting as a broker:

- **(1)** and **(5)** The NF(*m*) with the Global-ID equal to (*m*) from the server(*k*), publishes data on the variable(*X*).
- **(2)** As no subscription on variable(*X*) is registered inside the P4 switch, so the switch will discard those packets.
- **(3)** The NF(*n*) with the Global-ID equal to (*n*) from server(*l*), sends a SUB-register request or SUB-remove request for the variable-ID of the variable(*X*) to the P4 switch.
- **(4)** The P4 switch will store or remove the registration of the incoming port of this request packet, over the variable-ID mentioned in the request packet.
- **(6)** As there is a subscription about that variable-ID from NF(*n*) of the server(*l*), the switch will forward a copy of the packet to the port that eventually is destined to the NF(*n*) of the server(*l*).
- **(7)** The NF receives the publish packets of an update.
- **(8)** If all fragments of the Update are not delivered or some data is corrupted, the NF(*n*) will send a RECOVER message to the P4 switch.
- **(9)** The P4 switch will forward the packet to the REPLICA controller.
- **(10)** The REPLICA controller receives the request and will decide about it.



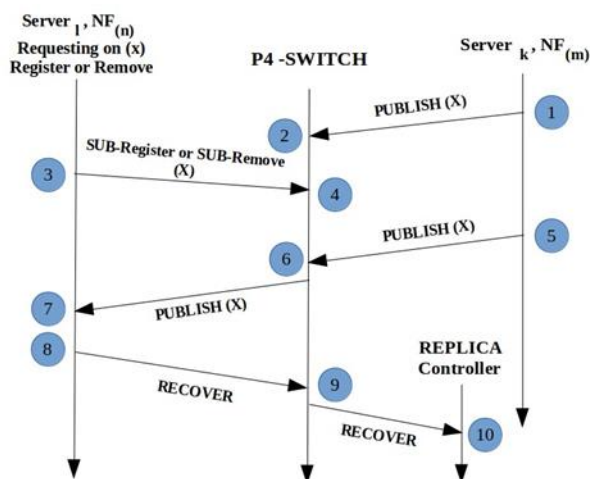


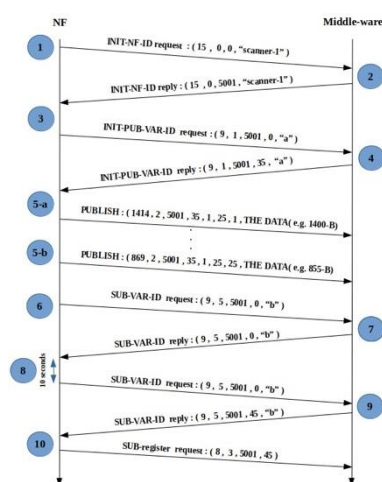
Figure 103: Protocol for the state replication through the P4 switch

### 3.8 The northbound interface of the middleware

We now described the protocol adopted by the NFs to interact with the middleware. All messages have two parts, a common part that has the same structure in all the messages, and a specific part which is message dependent:

- **Common part:** It is the first 6 bytes of the message, and is divided into three fields:
  - **Message length:** It is the first byte-pair of each message and contains an unsigned integer number which is the length of the message including itself in bytes. It is used in the TCP sockets for recovering the messages.
  - **Message kind:** It is the second byte-pair of each message and contains an unsigned integer number between zero and six, which is showing the message type:
    - \* **Type (0):** It is an INIT-NF-ID request message.
    - \* **Type (1):** It is an INIT-PUB-VAR-ID request message.
    - \* **Type (2):** It is a PUBLISH message.
    - \* **Type (3):** It is a SUB-register message.
    - \* **Type (4):** It is a SUB-remove message.
    - \* **Type (5):** It is a SUB-VAR-ID Request message.
    - \* **Type (6):** It is a RECOVER message.
    - \*
  - **NF-Global-ID:** It is the third byte-pair of each message and contains an unsigned integer number between 0 and 65535. The 0 is only being used for asking the INIT-NF-ID from the REPLICIA controller and in the rest of the cases it is a constant non-zero number distinguishing the NF from the others.
- **Specific part:** It contains the rest of the message, the details and the positions are described for each kind as bellow:
  - **Kind (0):** It carries the NF-name(*m*-Bytes). The *m* is the length of the NF-name. This message will have a reply similar to the request, while the number 0 in the Global-NF-ID field has been replaced with the assigned Global-NF-ID by the REPLICIA controller.
  - **Kind (1):** It carries the Variable-name(*n*-Bytes). The *n* is the length of the Variable-name. This message will have a reply similar to the request, while a part with 2 bytes length is inserted between the Global-NF-ID field and the Variable-name, containing the INIT-PUB-VAR-ID.

- **Kind (2):** It carries the Variable-ID(2 bytes) + the Update-number(2 bytes) + the Total-fragments(2 bytes) + the Fragment-ID + the Published data(*k*-bytes).
- **Kind (3):** It only carries the Variable-ID(2 bytes).
- **Kind (4):** It only carries the Variable-ID(2 bytes).
- **Kind (5):** It carries the Variable-name(*n* bytes). The *n* is the length of the Variable-name. This message will have a reply similar to the request, while a part with 2 bytes length is inserted between the Global-NF-ID field and the Variable-name, containing the SUB-VAR-ID. There are two kinds of replies to this request, if the SUB-VAR-ID in the reply is a non-zero number, the request was successfully answered, otherwise, zero means the Variable-name was not assigned before, so the NF will retry its request after a back-off time. This procedure will be repeated until receiving a non-zero SUB-VAR-ID.
- **Kind (6):** It carries the Variable-ID(2 bytes) + the Update-number(2 bytes) of the lost message parts.



**Figure 104: Protocol between the NFs and the middleware**

Figure 104 shows the protocol interaction between the NFs and the middleware. As example, we assume that the variable name is ‘a’.

- **(1)** Newly started NF wants to initialize, so it will send an INIT-NF-ID request containing Length = 15, Kind=0, Global-NF-ID: 0 and NF-name=‘scanner-1’, to the Middleware.
- **(2)** The Middleware replies to the NF with the same message format while rewriting the Global-NF-ID: 5001.
- **(3)** The NF needs to publish on its variable with the name e.g., ‘a’. So it will send an INIT-PUB-VAR-ID request containing Length=9, Kind=1, Global-NF-ID: 5001 and PUB-VAR-name=‘a’, to the Middleware.
- **(4)** The Middleware replies to the NF with the same message format while rewriting the Variable-ID: 35.
- **(5-a) till (5-b)** The NF wants to publish on variable with the PUB-VAR-name=‘a’ and the Variable-ID: 35, so after generating the update, it breaks the data into the chunks with the maximum size equal to e.g., 1400 Bytes, then it calculates the number of the fragments and will send sequentially the fragments one after each other with the Kind=2, the Update-number=1, the Total-fragments=(e.g., 25) and the Fragment-ID followed by the DATA.

- **(6)** The NF decides to subscribe on a variable that it is already aware of its name, so it will send a SUB-VAR-ID request containing Length=9, Kind = 5, Global-NF-ID: 5001 and SUB-VAR-name='b', to the Middleware.
- **(7)** If there is no previous id assignment for the variable 'b', then the Middleware replies to the NF with the same message format while rewriting the Variable-ID: 0 as an 'ERROR' message.
- **(8)** By receiving this 'ERROR' message, the NF will wait for 10 seconds and will repeat the step **(6)**.
- **(9)** The Middleware replies to the NF with the same message format while rewriting the Variable-ID: 45.
- **(10)** The NF sends a SUB-register request with the Kind=3 and the Variable-ID: 45 to the Middleware

### 3.9 NF implementation

The internal diagram of the implemented NF is demonstrated in Figure 105. Each NF has been implemented through 5 threads, started sequentially:

1. **Receive thread:** When the Receive thread starts, it tries to connect to the address tuple ('localhost', special-port-1). This thread will act as a blind receiver and tries to receive in an infinite loop and write the received data in a queue made for it.
2. **Send thread:** When the Send thread starts, if any, it tries to pop messages from a queue made for it, and sends them to the socket connection made previously by the Receive thread.
3. **Msg-handlr thread:** When the Msg-handlr thread starts, if any, it continuously pops items from the output queue of the Receive thread, and rebuilds the received packets by the NF through some information that we already put in the original messages. After that, it decides what to do with the messages:
  - **INIT-NF-ID reply:** It will store the INIT-NF-ID and will change the flag related to the Init-publish thread for moving to the INIT-PUB-VAR-ID request step.
  - **INIT-PUB-VAR-ID reply:** It will store the INIT-PUB-VAR-ID and will change the flag related to the Init-publish thread for moving to the PUBLISH step.
  - **SUB-VAR-ID reply:** It will store the SUB-VAR-ID reply for the Subscribe thread to decide whether to resend the request after some waiting, or make the SUB-register message.
  - **PUBLISH messages:** It will check for the possible packet losses, if there is no packet loss, it will store the PUBLISH messages in a file, otherwise it will make a RECOVER message and will add it to the input queue of the Send thread.
4. **Init-publish thread:** When the Init-publish thread starts, it tries to do the initializing phase by making the INIT-NF-ID request message and adding this message to the input queue of the Send thread. It waits until receiving the proper answer, then it makes the INIT-PUB-VAR-ID request message and adds this message to the input queue of the Send thread. It waits until receiving the proper answer, then it will start to make the PUBLISH packets and will add them to the input queue of the Send thread. It also stores a copy of the PUBLISH packets it is making in a file for further possible needs.
5. **Subscribe thread:** When the Subscribe thread starts, it tries to start the information request sub-phase, by making the SUB-VAR-ID Request message and adding this message to the input queue of

the Send thread. If it receives an ERROR message, it waits for 10 seconds and will retry by making the SUB-VAR-ID request message and adding this message to the input queue of the Send thread, until receiving the proper answer. Then it will create the SUB-register message, and will add this message to the input queue of the Send thread.

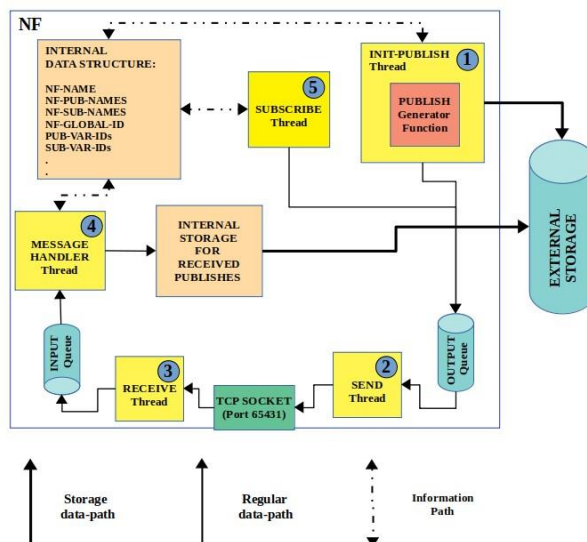


Figure 105: The internal architecture for the NFs.

### 3.10 STARE middleware implementation

The Middleware does not create any messages from zero by itself. When it receives a message from the NFs, if it needs to be sent to the network, then the Middleware will send the message to the proper multicast IP address. When it receives a message from the network, then based on the message kind, it will forward the message to the proper NFs. The Middleware, except for the cases of the INIT-NF-ID request and the INIT-NF-ID reply, will not change any field of the received messages. In those mentioned cases, for mapping the received INIT-NF-ID reply to the proper NF, the Middleware will assign a LOCAL-NF-ID to each of the NFs once it receives a INIT-NF-ID request in its internal data structure. Then it will add this value as a 2 bytes length field after the GLOBAL-NF-ID field in the message so that it can be recognized in the reply message. The Middleware will eliminate this field when it receives the INIT-NF-ID reply message, the NF will be completely unaware of this process.

Figure 106 shows the interaction between the middleware and the network, described as follows:

- **(1)** The Middleware adds the already assigned LOCAL-NF-ID to the INIT-NF-ID request as mentioned previously, then it sends the message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the REPLICA controller.
- **(2)** The P4 switch forwards the received packet based on its IPV4.lpm routing table.
- **(3)** The REPLICA controller will replace the Global-NF-ID field of the message with the one it assigned to the NF, and sends back the INIT-NF-ID reply message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the server which is hosting the Middleware.
- **(4)** The Middleware removes the LOCAL-NF-ID field and sends the message to the related NF.
- **(5)** The Middleware sends the INIT-PUB-VAR-ID request message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the REPLICA controller.

- **(6)** The REPLICATION controller will replace the Global-PUB-VAR-ID field of the message with the one it assigned to the Variable, and sends back the INIT-PUB-VAR-ID reply message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the server which is hosting the Middleware.
- **(7)** The Middleware sends the INIT-PUB-VAR-ID reply message to the related NF.
- **(8-a)** to **(8-b)** The Middleware will send the PUBLISH messages as UDP packets to the UDP port number 65432 and the IP address of the corresponding IP-Multicast group.
- **(9-a)** to **(9-b)** The P4 switch through its internal data structure will forward them to the registered ports, otherwise it will drop the packets.
- **(10)** The Middleware sends the SUB-VAR-ID request message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the REPLICATION controller.
- **(11)** The REPLICATION controller will check its internal data structure to see if any Variable-ID has already been assigned to this variable. If the answer is positive, then the REPLICATION controller will put this number in the Variable-ID field of the message, otherwise, it will put 'error' in the message. Finally, it sends the SUB-VAR-ID reply message as a Unicast UDP packet to the UDP port number 65432 and the IP address of the server which is hosting the Middleware.
- **(12)** The Middleware sends the SUB-VAR-ID reply message to the related NF. But as it was a reply containing 'error' message, after almost 10 seconds The Middleware will receive a repeated SUB-VAR-ID request message, and will send it to the REPLICATION controller again. Steps **(10)** to **(12)** will be repeated until instead of the 'error' message, a Variable-ID is received by the NF.
- **(13)** The Middleware will send the SUB-register messages as UDP packets to the UDP port number 65432 and the IP address of the corresponding IP-Multicast group.
- **(14)** The P4 switch will register the request inside its internal data structure.

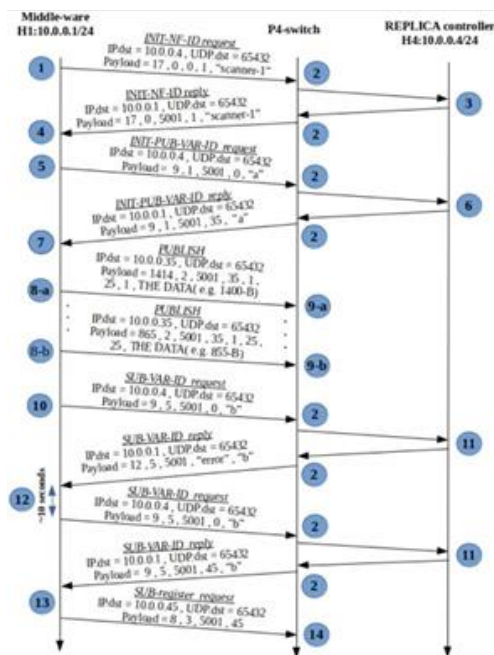


Figure 106: Interaction of the Middleware with the network.

We refer to Figure 105 for the software architecture referred in this section. Four initial threads start sequentially, and each NF connecting to the Middleware will be managed by 3 other threads, specifically for handling the communication with that NF. Thus, the total number of threads inside the Middleware is calculated as  $4 + m + 3 \times k$ , where the  $k$  is the number of the connected NFs and  $m$  is a value equal to the number of subscribed variables divided by the maximum IGMP membership limitation of the OS. In more details, the threads are the following:

1. **INITIAL thread:** The Middleware starts by making the first thread, which is responsible for making a server for a TCP socket, listening on a specific port (e.g., 65431 in our case) for incoming connections from the NFs. This port is predefined in all NFs, and it is used to start the primary connection between each NF and the Middleware. After accepting the connection from the NF, three threads will be made, responsible for the communications related to this NF:
  - **SEND thread:** It is responsible for sending messages to the connection of that NF. There is a queue assigned to this thread, which is continuously checked by this thread to see if any message is available for sending.
  - **RECEIVE thread:** It is responsible for receiving messages from the connection of that NF. There is a queue assigned to this thread. It continuously adds the data it receives from the NF connection to this queue.
  - **MESSAGE-HANDLER thread:** For each NF connected to the Middleware, a third thread will be created and will be responsible for reading from the output queue of the receiver thread and rebuilding the original message sent by the NF by using the Length field of the messages, if needed. This thread based on the message kind and the available information inside the internal data structure of the Middleware, may update the internal data structure of the Middleware or add a copy of this message into the queue for the send thread of the other NFs. Anyway, regardless of the message kind, it adds the original message to the main outgoing queue of the Middleware destined to the network.
2. **MAIN SEND thread:** This thread is responsible for creating a UDP socket and, if there is a message in the main outgoing queue of the Middleware, the thread will create an IP packet. Based on the message kind it will make a proper destination for the IP layer, sets the UDP layer destination port to another special port (e.g., 65432 in our case), and will send the packet to the network. In the case of subscriptions, it will do the membership in the proper IP-multicast group in the OS and if reached the maximum IGMP membership limitation of the OS, will build a new thread for a new range of the IGMP membership.
3. **MAIN RECEIVE thread:** This thread creates another UDP socket and will bind this socket to an address tuple made of the server IP address and the second-mentioned special port (65432). It is responsible for receiving from this socket and will add whatever it receives to the main incoming queue of the Middleware.
4. **MAIN MESSAGE-HANDLER thread:** This thread is responsible for reading from the main incoming queue of the Middleware, updating the internal data structure of the Middleware, and adding a copy of the message to the input queue of the proper NFs.

### 3.11 Alternative implementation with OpenFlow switches

This scenario requires the controller to pre-install the following 2 match-action rules inside the flow table of the Open vSwitch during the initialization phase.

The first rule is to recognize the SUB-register request and the SUB-remove request packets. Both messages are characterized, according to adopted coding, by the 9th bit equal to one. Thus:

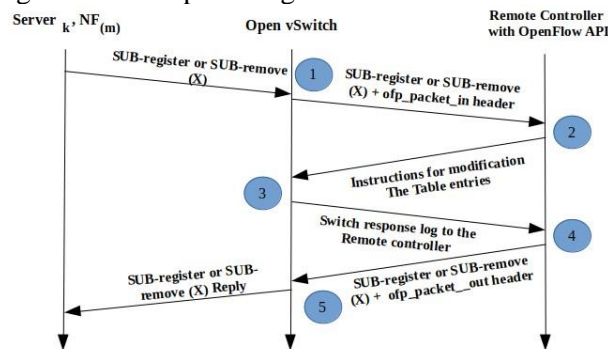
OpenFlow field	Value	Comment
OXM OF _ETH DST	ff:ff:ff:ff:ff:ff	Broadcast destination MAC
OXM OF _ETH TYPE	0x800	Ethernet payload is IPv4
OXM OF _IPV4 DST	239.128.0.0/9	IP for SUB-register/remove requests
OXM OF _IP PROTO	0x11	IP payload is UDP
OXM OF _UDP DST	65432	UDP destination port

The corresponding action will be to send the packets to the port corresponding to the SDN controller. Similar to the previous two scenarios, we have implemented the REPLICa controller within the SDN controller, such that REPLICa controller will reply to the NF with the proper NF-Global-ID (for the INIT-NF-ID request and the variable-ID (for the SUB-VAR-ID Request and the INIT-PUB-ID Request) has already pre-installed by the remote SDN controller.

The second rule is to drop the Publish messages that have not been registered so far and there MatchAction rules installed for them. The format of Publish messages requires both the 9th and 10th bit equal to zero. The corresponding rule is set at the lowest priority to refer only to unregistered Publish messages.

OpenFlow field	Value	Comment
OXM _OF ETH DST	ff:ff:ff:ff:ff:ff	Broadcast destination MAC
OXM _OF ETH TYPE	0x800	Ethernet payload is IPv4
OXM _OF IPV4 DST	239.0.0.0/10	IP for Publish message
OXM _OF IP PROTO	0x11	IP payload is UDP
OXM _OF UDP DST	65432	UDP destination port
priority	0	lowest priority

Anytime a SUB-register request or a SUB-remove request message is received, the switch will encapsulate these messages and will send them to the remote SDN controller for inspecting and installing the necessary rules inside the Open vSwitch. The general time-space diagram for this situation is demonstrated in Figure 107.



**Figure 107: Protocol for Remote Controller-based scenario with OpenFlow switch.**

The other steps including the communication between the NFs and the Middleware is topology independent and similar to the previously described one.

- **(1)** When a SUB-register or a SUB-remove packet arrives at the switch ports, it will be sent through OpenFlow API to the Remote SDN controller with an OFP-Packet-In header.
- **(2)** Due to the definition structure of the match fields in the OpenFlow SDN switches, if it is a SUB-register, the Controller will install a matching rule inside the switch to forward any further publishes

of this Variable-ID, to the port that this request already comes from it, with the priority higher than the priority of the default dropping rule, pre-installed for the Publish message at the start of the controller.

OpenFlow field	Value	Comment
OXM OF ETH DST	ff:ff:ff:ff:ff:ff	Broadcast destination MAC
OXM OF ETH TYPE	0x800	Ethernet payload is IPv4
OXM OF IPV4 DST	239.0.X.X/32	Publish message for variable-id X.X
OXM OF IP PROTO	0x11	IP payload is UDP
OXM OF UDP DST	65432	UDP destination port
priority	2	higher priority than default dropping

The Action will be to send the packet to the port from which the SUB-register packet arrived at the switch. If the match already exists, the controller just updates the Action part by adding this port to the previous ones. In the case of Remove, the controller will update the Action part of the corresponding match rule by removing the mentioned port from the Action list if it exists or will remove the rule if no other actions remain for that rule. The controller will use the information of the packet header fields, the input port and its previous data structure on table entries, and will send proper instructions to the switch for installing new rules or updating previous ones.

- **(3)** Switch will report the configuration procedure and results to the SDN controller.
- **(4)** The Remote SDN controller will update its data structure with the switch report and will send the Subscribe packet it was received at **(1)** after swapping the IP-destination and IP-source fields, removing the Packet-In header and adding a Packet-Out header containing the same input port existed in Packet-In header, as the new output port for the switch.
- **(5)** The switch will remove the Packet-out header after extracting its information, and will send the packet to the NF, through the port mentioned in that information.

If the incoming packet to the switch is one of the INIT-NF-ID request, the INIT-PUB-ID Request or the SUB-VAR-ID Request, the switch will send the packet to the REPLICA Controller.



## 4 Microservices-inspired VNF re-design

### 4.1 Motivation

There is currently a huge research effort on the softwarization of the mobile network. Among other efforts, 5G mobile communications are working towards the introduction of a fully softwarized architecture [14]. However, as compared to the cloud evolution, the telecommunications world is still half-way in this transition, despite the adoption of technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) which have helped towards the softwarization of network architectures, and the architectural trend towards their modularization' with a clean separation between the control and the user planes. That is, the trend is to split, already at the architectural level, the formerly monolithic nodes into several smaller logical entities. Thus, the 5G Next Generation NodeBs (gNBs) can be split into centralized and distributed units, denoted as gNB-CU and gNB-DU, respectively [15], while core network components grow both in number and in functionality.

The telco world is lagging in the adoption of novel software paradigms as compared vs. the cloud world, despite achievements such as:

- The standardization of 3GPP Release 15 [15], which specifies the Service Based Architecture (SBA). This represents a new paradigm for the 5G Core Network and is driven by the trend towards the modularization of the network. With this approach, the formerly static interface between different elements has evolved into a flexible bus, which hosts HTTP REST primitives between modules.
- The concept of Cloud-Native Network Functions (CNF), which is making its way into the current technology. In fact, there are already proposals for the design of cloud-native VNFs. However, they are in a very early stage and mostly involve Core Network VNFs only. We believe that the softwarization paradigm change shall involve all domains, including the most challenging one such as the RAN (as exemplified below).

Despite these achievements, we are still in the middle of this transition as the cloud-native paradigm has not been fully adopted into operational networks. This is caused by the poor agility of the current state-of-the-art solutions, and the fact that current VNFs are not truly agnostic to the underlying NFV infrastructure. While dynamic cloud resources orchestration algorithms are currently under study [17], the VNFs that will be running on such resources are still not optimized for this type of operation.

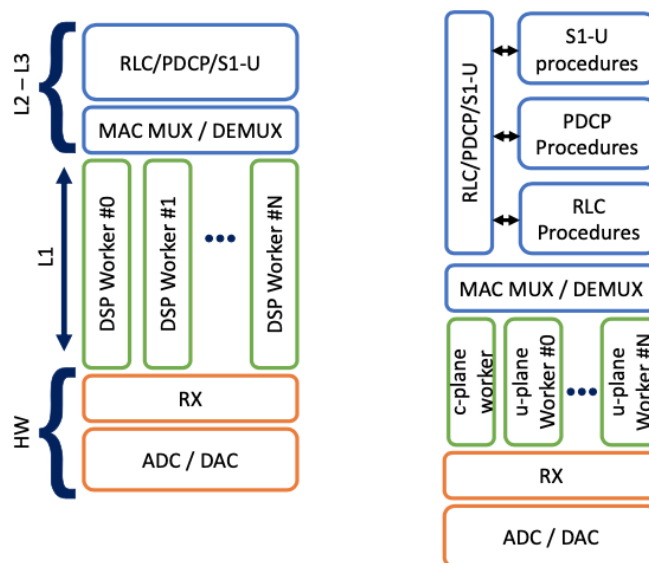
So even if the efforts towards the cloud-native transition of the NFV are still ongoing, the research community shall prepare for the next transition. This will introduce a complete re-design of the whole mobile protocol stack, which will certainly facilitate a dynamic resource orchestration and assignment, allowing hence higher efficiency. More specifically, we need a mobile protocol stack that is (1) more efficient in terms of both resource and time granularity scalability, and (2) capable to elastically adapt to the instantaneous demand. We claim that with such a protocol stack, the deployment and operational costs of the network will be reduced to their minimum. Given that this flexible and on-demand operation of the network closely resembles the current operation of cloud computing platforms, it should also follow similar principles.

### 4.2 Approach

The current way of implementing VNF is still very bound to the traditional way of implementing network functions. Current solutions do not embrace modularization: many commercial products are softwarized but very bounded to the hardware platform, while open source initiatives are practically mere translations of hardware functionality into software modules. To adopt the serverless approach, we need to change how VNFs are designed. To illustrate our point, in the previous section we focus on the user plane part of the RAN, but the same paradigm could be easily applied to other network functions. Our motivation is that, as the radio functions are the most resource-consuming ones (considering resources of all kinds: spectrum, transport network, and computational resources [18]), we expect that the transition towards high modularity will be especially beneficial for such functions.

We take as exemplary case study a well-known open source implementation of a 4G-LTE RAN stack, namely, srsLTE [19]. We illustrate on the left part of Figure 108 its threading architecture [20], which follows a *classical*

layered architecture for a great extent and has remarkable modularity, in particular considering that the software has been designed for stand-alone operation. Still, the division is quite coarse, and there are additional issues that would prevent the use of a serverless approach, e.g., the physical layer does not distinguish between control and data channels, which are processed sequentially.



**Figure 108: The simplified threading architecture of the srsLTE software (left) and a possible serverless design of the software stack (right).**

In this way, while the srsLTE design is perfectly valid for the working conditions initially considered by its developers, the architecture would benefit from a different design such as the one suggested in the right-hand part of Figure 108. Here, the control and data plane processing are placed in different modules: while the control plane functionality on the L1 is fixed and has to be performed independently of the load, the user plane could be placed and executed in different threads (or even containers) to scale them according to the load. Alternatively, the software can be modified to support intelligent resource assignment schemes.

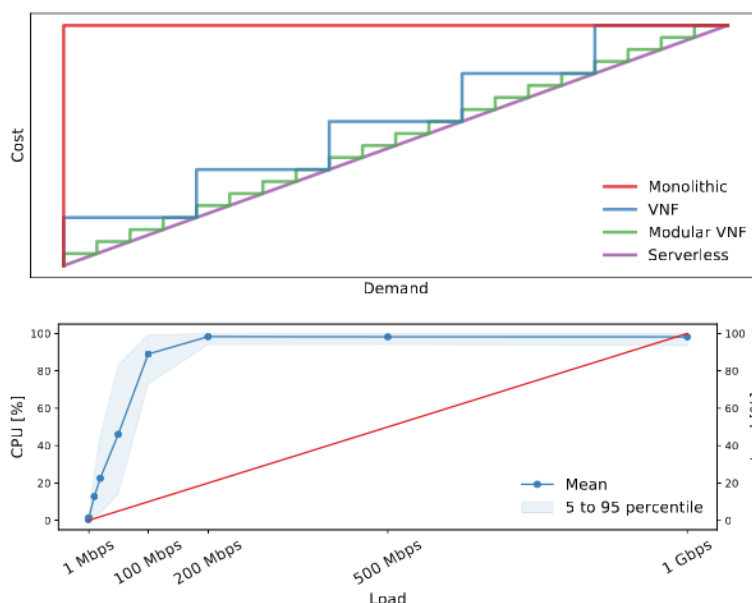
However, the current state of the art of networking (i.e., the Linux kernel) can hardly support this approach, as it is heavily API-based and requires fast inter-process communications. Moreover, current network functions (especially RAN functions) exhibit two fundamental characteristics that make them different when compared to other cloud computing applications: (1) they impose very high load on the CPU (i.e., encoding and decoding of wireless frames) and (2) they have very stringent timing requirements as usually communication protocols are time-partitioned and need time synchronization. While there are CPU intensive services running in the cloud (e.g., Netflix video transcoding) these are not real-time. These timing constraints were barely a problem in the (now old) PNF paradigm, but it is a challenge to address since the arrival of novel technologies such as e.g. cloud C-RAN.

Making tasks aware of their execution time is usually not conceived as a problem in general cloud computing systems, which barely have to provide near to real-time outputs. The Linux kernel system provides tools to address this problem, but real-time software usually runs in rugged embedded devices for industrial purposes (e.g. robotics) or in dedicated data centers for fast pace trading in stock exchanges, not in the cloud. So, VNFs shall be re-designed to also take advantage of e.g., the real-time kernel primitives or used jointly with the elastic network function design as investigated in [18].

### 4.3 Estimated gains

This novel approach to implement mobile network function will not lead to substantial gains when the systems operate at full capacity. However, in very dynamic environments such as the ones envisioned for 5G and beyond, the novel paradigm will support achieving liquid scalability, i.e., the highest level of modularization. This means that specific functions of a VNF can be scaled according to the real demand, avoiding the scaling of the full VNF instead, and achieving the liquid scalability depicted in Figure 109(top). We believe that this will be one of the fundamental pillars for the sustainable operation of the next-generation networks. In [21], we quantified

the cost in terms of resource overhead of deploying and operating the infrastructure needed to support multi-service networks. This study showed that the efficiency (i.e., the number of resources used by a not multi-tenant network compared to a multi-tenant one) is very low, and just with a very dynamic network reconfiguration (such as the ones envisioned here) it is possible to improve these figures. This study showed that the efficiency (i.e., the number of resources used by a not multi-tenant network compared to a multi-tenant one) is very low (15%) for edge resources (e.g., spectrum, antennas) and only slightly better (65%) when considering core resources (e.g., CPU in a cloud data center). The study also shows that only with a very dynamic network reconfiguration it is possible to improve these figures up to 60% and 90%, respectively. So, achieving the finest granularity (the analysis of [21] is performed at byte level) such as the one envisioned by the serverless paradigm will allow achieving such extreme gains in terms of resource utilization.



**Figure 109: The liquid scalability (top) and an empirical evaluation (bottom)**

Figure 109 (bottom) depicts an empirical evaluation of the liquid scalability concept, by evaluating the CPU footprint of a state-of-the-art VNF (OpenVSwitch), running inside a KVM Virtual Machine in Linux. This setup summarizes the VNF approach sketched in the upper part of the figure as it can be scaled on a VM-basis (new VMs can be added or removed according to the load), this being the only way of scaling up or down according to the load.

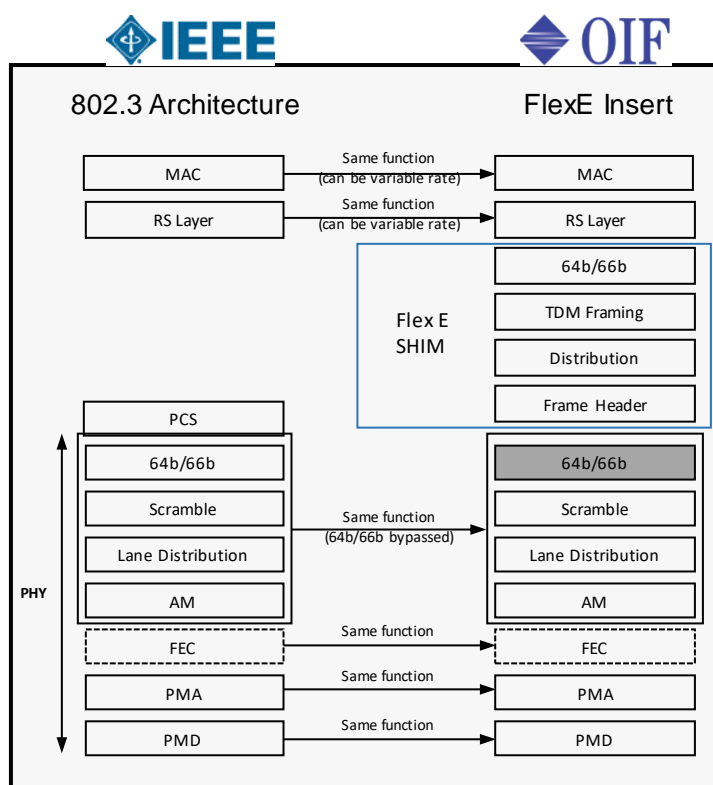
We can observe that this way of softwarizing clearly falls short when the objective is finer scalability. We can observe that while growing the offered load (in our experiments we span 4 orders of magnitude, from 100 Kbps to 1 Gbps) the resource footprint utilization has a clear on-off behaviour. Indeed, the CPU utilization shortly hits close to 100% utilization with offered loads that barely reach the 10% of the total. This means that there is still a lot of room for improvement in the software design of VNF with respect to IT resources consumption.

## 5 Flexible Ethernet and Preferred Path Routing

This section details two further prospective experiments related to new networking features. Specifically, a novel data plane such as Flexible Ethernet was evaluated to infer characterization of such capabilities for enabling uRLLC services, and a new source routing protocol encapsulation method, named Preferred Path Routing (PPR) was exercised.

### 5.1 Latency and jitter with Flexible Ethernet data plane

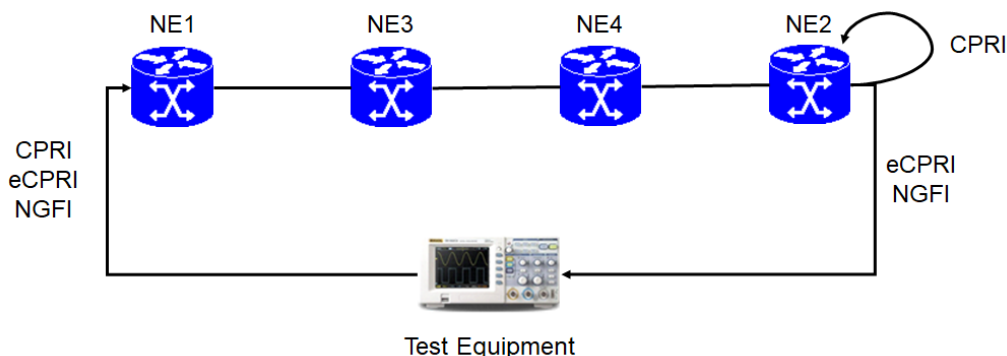
Flexible Ethernet (FlexE) emerges as a new alternative transport technology at packet switching level, for a multiservice (fronthaul, backhaul, corporate and broadband). FlexE basically decouples the MAC binary rates from the underlying PHY rate. This allows different schemas of transmission. Essentially, a new sub-layer named FlexE Shim is defined between traditional MAC and PCS layers of Ethernet. This sub-layer supports 5G mobile synchronization requirements for fronthaul and backhaul and permits distributing or aggregating clients across different data blocks. Thus, 100 GE PHY interfaces are sliced in 20 blocks, generating a granularity of 5 Gbps channels (in a TDM-like fashion). The assignment of those channels is flexible, allowing for different schemas like channelization (multiple FlexE clients sharing the same PHY interface) or bonding (aggregating channels from multiple PHY interfaces).



**Figure 110: FlexE Shim layer as standardized by OIF**

The purpose of the tests was to characterize this novel data plane and observe characteristics that could be incorporated in 5G environments in order to accomplish stringent requirements such as high bandwidth and very low latency.

An experiment was executed pursuing the characterization of the behavior of node elements incorporating Flexible Ethernet capabilities. Vendors participant in the tests are subject to NDA. The basic setup is shown in Figure 111.



**Figure 111: Flexible Ethernet test setup**

Ports connecting NE1, NE2 and NE3 are 100 GE ports, supporting FlexE capabilities. Up to four nodes are considered in the setup in order to measure a more realistic aggregation scenario in terms of router hops. Due to test equipment limitations, the CPRI signal is sent-back from NE2 router.

Three different client signals are simultaneously injected in the setup: CPRI (option 7, 9.83 Gbps), eCPRI (as 25 Gbps) and NGFI (as 10 Gbps). The purpose of the test is to measure the delay induced by the introduction of intermediate network elements in the path to determine the ultra-low latency forwarding support in intermediate nodes (as demanded by 5G uRLLC service type). The actions taken are:

- As first step, the three client services are configured to be delivered solely between NE1 and NE2 (i.e., without the intermediate nodes NE3 and NE4). In these circumstances, the observed latency T1 is measured.
- As second step, the same services are then configured between NE1 and NE2 but including NE3 and NE4 in the path, configuring Flexible Ethernet timeslot cross-connections in NE3 and NE4 for the three services respectively. With that, the test equipment records the value of service latency T2.

The latency at the intermediate node is calculated through the formula  $(T2-T1)/2$ .

The following table reflects the results obtained:

	CPRI (round trip)		eCPRI (unidirectional)		NGFI (unidirectional)	
	Latency (ns)	Jitter (ns)	Latency (µs)	Jitter	Latency (µs)	Jitter
2 nodes	2192.6	0	0.93	0	1.32	0
4 nodes	3986.8	2	1.88	0	2.28	0

With these results, the contribution per intermediate node can be determined as:

- CPRI – 448.9 ns
- eCPRI – 475 ns
- NGFI – 480 ns

As conclusion, on the consideration of FlexE as data plane technology to be integrated in scenarios as the ones in 5G-EVE, a delay budget of about 0.5 µs should be accounted. Such a minimal delay could be convenient for some vertical cases experimenting uRLLC-like services.

## 5.2 Experimental evaluation of Preferred Path Routing (PPR)

Preferred Path Routing (PPR) is a new encapsulation protocol in the source routing space. The motivation for PPR is minimizing the data plane overhead caused in SR, which impacts both the header and processing overhead per packet. This is more relevant for the case of small packets. This could be critical on many 5G applications, especially for verticals. In principle, PPR can work on top of IP data planes without needing to replace existing hardware or even to upgrade the data plane.

This first experience intends to show configuration capabilities of PPR extensions on FRR open source stack. The following test topology has been taken as a guide, as shown in Figure 112, both for setting the test scenario and for configuring PPR: <https://github.com/opensourcerouting/frr/wiki/PPR---Basic-Test-Topology#software> In this topology we have an IS-IS network consisting of 5 routers. Host 1 and Host 2 are the connected to R11 and R13, respectively.

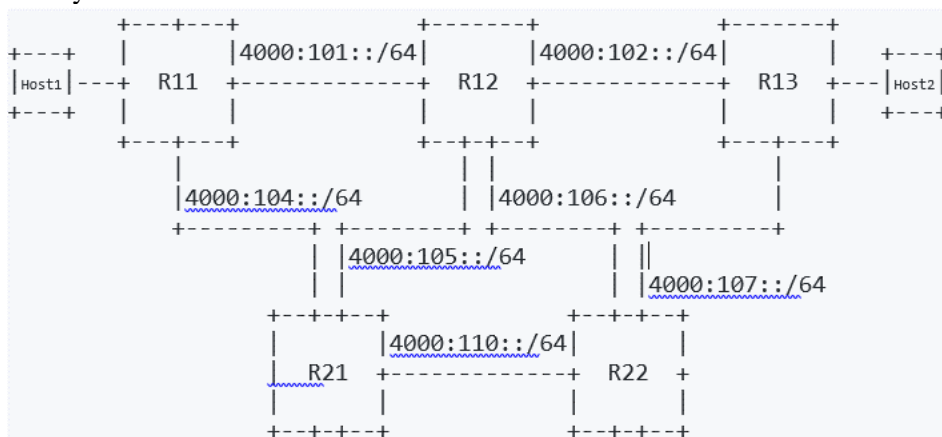


Figure 112: PPR test topology

### 5.2.1 PPR configuration

Router R11 advertises 6 PPR TLVs which corresponds to three bi-directional GRE tunnels:

- 6000:1::1 <-> 6000:2::1: {R11 - R21 - R22 - R13} (IPv6 Node Addresses only)
- 6000:1::2 <-> 6000:2::2: {R11 - R21 - R12 - R22 - R13} (IPv6 Node Addresses only)
- 6000:1::3 <-> 6000:2::3: {R11 - R12 - R13} (IPv6 Node Addresses only)

PBR rules are configured on R11 and R13 to route the traffic between Host 1 and Host 2 using the first PPR tunnel (6000:1::1 <-> 6000:2::1).

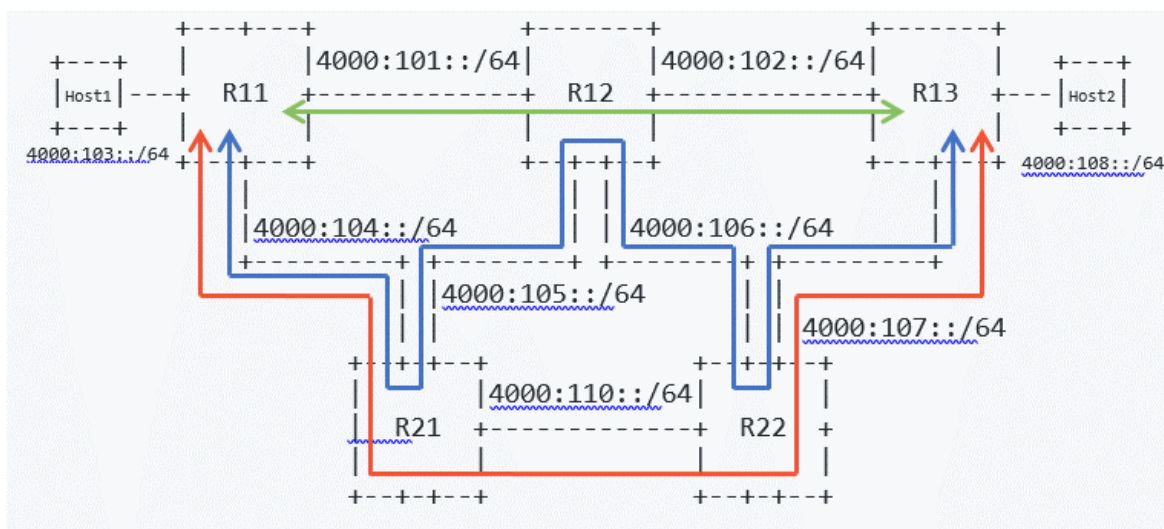


Figure 113: PPR configured paths

#### RT11:

```
# GRE tunnel for preferred packets (PPR)
ip -6 tunnel add tun-ppr mode ip6gre remote 6000:2::1 local 6000:1::1 ttl 64
ip link set dev tun-ppr up

# PBR rules
```

```
ip -6 rule add from 4000:103::/64 to 4000:108::/64 iif ens9 lookup 10000
ip -6 route add default dev tun-ppr table 10000

interface ens6
  ipv6 address 4000:101::11/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface ens7
  ipv6 address 4000:104::11/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface ens8
  ipv6 address 6000:1::1/128
  ipv6 address 6000:1::2/128
  ipv6 address 6000:1::3/128
!
interface ens9
  ipv6 address 4000:103::11/64
!
interface lo
  ipv6 address 5000::11/128
  ipv6 router isis 1
  isis circuit-type level-1
  isis passive
!
ppr group INTERFACE_PDES
  ppr-id ipv6 6000:2::1/128 prefix 5000::13/128
  pde ipv6-node 5000::11/128
  pde ipv6-node 5000::21/128
  pde ipv6-node 5000::22/128
  pde ipv6-node 5000::13/128
!
  ppr-id ipv6 6000:1::1/128 prefix 5000::11/128
  pde ipv6-node 5000::13/128
  pde ipv6-node 5000::22/128
  pde ipv6-node 5000::21/128
  pde ipv6-node 5000::11/128
!
!
ppr group INTERFACE_PDES2
  ppr-id ipv6 6000:1::2/128 prefix 5000::11/128
  pde ipv6-node 5000::13/128
  pde ipv6-node 5000::22/128
  pde ipv6-node 5000::12/128
  pde ipv6-node 5000::21/128
  pde ipv6-node 5000::11/128
!
  ppr-id ipv6 6000:2::2/128 prefix 5000::13/128
  pde ipv6-node 5000::11/128
  pde ipv6-node 5000::21/128
  pde ipv6-node 5000::12/128
  pde ipv6-node 5000::22/128
  pde ipv6-node 5000::13/128
!
!
ppr group INTERFACE_PDES3
  ppr-id ipv6 6000:1::3/128 prefix 5000::11/128
  pde ipv6-node 5000::13/128
  pde ipv6-node 5000::12/128
  pde ipv6-node 5000::11/128
!
  ppr-id ipv6 6000:2::3/128 prefix 5000::13/128
  pde ipv6-node 5000::11/128
  pde ipv6-node 5000::12/128
  pde ipv6-node 5000::13/128
!
!
router isis 1
  net 49.0000.0000.0000.0011.00
  lsp-mtu 1447
  topology ipv6-unicast
  ppr on
  ppr advertise INTERFACE_PDES
  ppr advertise INTERFACE_PDES2
  ppr advertise INTERFACE_PDES3
!
line vty
!
end
```

**RT12:**

```

interface ens6
  ipv6 address 4000:101::12/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface ens7
  ipv6 address 4000:105::12/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface lo
  ipv6 address 5000::12/128
  ipv6 router isis 1
  isis circuit-type level-1
  isis passive
!
router isis 1
  net 49.0000.0000.0000.0012.00
  lsp-mtu 1447
  topology ipv6-unicast
  ppr on
!
line vty
!
end

```

**RT13:**

```

# GRE tunnel for preferred packets (PPR)
ip -6 tunnel add tun-ppr mode ip6gre remote 6000:1::1 local 6000:2::1 ttl 64
ip link set dev tun-ppr up

# PBR rules
ip -6 rule add from 4000:108::/64 to 4000:103::/64 iif ens9 lookup 10000
ip -6 route add default dev tun-ppr table 10000

interface ens6
  ipv6 address 4000:102::13/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface ens7
  ipv6 address 4000:107::13/64
  ipv6 router isis 1
  isis circuit-type level-1
  isis hello-multiplier 3
  isis network point-to-point
!
interface ens8
  ipv6 address 6000:2::1/128
  ipv6 address 6000:2::2/128
  ipv6 address 6000:2::3/128
!
interface ens9
  ipv6 address 4000:108::13/64
!
interface lo
  ipv6 address 5000::13/128
  ipv6 router isis 1
  isis circuit-type level-1
  isis passive
!
router isis 1
  net 49.0000.0000.0000.0013.00
  lsp-mtu 1447
  topology ipv6-unicast
  ppr on
!
line vty
!
end

```

**RT21:**

```

interface ens6
  ipv6 address 4000:104::21/64
  ipv6 router isis 1

```



```
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface ens7
ipv6 address 4000:105::21/64
ipv6 router isis 1
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface ens9
ipv6 address 4000:110::21/64
ipv6 router isis 1
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface lo
ipv6 address 5000::21/128
ipv6 router isis 1
isis circuit-type level-1
isis passive
!
router isis 1
net 49.0000.0000.0000.0021.00
lsp-mtu 1447
topology ipv6-unicast
ppr on
!
line vty
!
end
```

## RT22:

```
interface ens6
ipv6 address 4000:106::22/64
ipv6 router isis 1
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface ens7
ipv6 address 4000:107::22/64
ipv6 router isis 1
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface ens8
ipv6 address 4000:110::22/64
ipv6 router isis 1
isis circuit-type level-1
isis hello-multiplier 3
isis network point-to-point
!
interface lo
ipv6 address 5000::22/128
ipv6 router isis 1
isis circuit-type level-1
isis passive
!
router isis 1
net 49.0000.0000.0000.0022.00
lsp-mtu 1447
topology ipv6-unicast
ppr on
!
line vty
!
end
```

## 5.2.2 Verification

We report here the outcomes of the performed verifications. Basically, the approach taken has been to check the operational behavior of the protocol in order to ensure proper functioning and stability of the version in FRR. This verification process consisted on checking the functionality of both the control plane (supported on top of IS-IS) and the forwarding plane, verifying the overall behavior of the protocol.

### 5.2.2.1 Verification of PPR Control Plane

Verify that R11 has flooded the PPR TLVs correctly to all IS-IS routers:

```

rt11# show isis database detail 0000.0000.0011
Area 1:
IS-IS Level-1 link-state database:
LSP ID                PduLen  SeqNumber  Chksum  Holdtime  ATT/P/OL
rt11.00-00            *      994      0x00000f52  0x6e61    831    0/0/0
  Protocols Supported: IPv4, IPv6
  Area Address: 49.0000
  MT Router Info: ipv4-unicast
  MT Router Info: ipv6-unicast
  Hostname: rt11
  TE Router ID: 192.168.165.111
  Router Capability: 192.168.165.111 , D:0, S:0
    Algorithm: 0: SPF 0: SPF
  MT Reachability: 0000.0000.0012.00 (Metric: 10) ipv6-unicast
  MT Reachability: 0000.0000.0021.00 (Metric: 10) ipv6-unicast
  IPv4 Interface Address: 192.168.165.111
  MT IPv6 Reachability: 5000::11/128 (Metric: 10) ipv6-unicast
  MT IPv6 Reachability: 4000:101::/64 (Metric: 10) ipv6-unicast
  MT IPv6 Reachability: 4000:104::/64 (Metric: 10) ipv6-unicast
  PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
    PPR Prefix: 5000::11/128
      ID: 6000:1::1/128 (Native IPv6)
      PDE: 5000::13/128 (IPv6 Node Address), L:0 N:0 E:0
      PDE: 5000::22/128 (IPv6 Node Address), L:0 N:0 E:0
      PDE: 5000::21/128 (IPv6 Node Address), L:0 N:0 E:0
      PDE: 5000::11/128 (IPv6 Node Address), L:0 N:1 E:0
    PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
      PPR Prefix: 5000::13/128
        ID: 6000:2::1/128 (Native IPv6)
        PDE: 5000::11/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::21/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::22/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::13/128 (IPv6 Node Address), L:0 N:1 E:0
    PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
      PPR Prefix: 5000::11/128
        ID: 6000:1::2/128 (Native IPv6)
        PDE: 5000::13/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::22/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::12/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::21/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::11/128 (IPv6 Node Address), L:0 N:1 E:0
    PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
      PPR Prefix: 5000::13/128
        ID: 6000:2::2/128 (Native IPv6)
        PDE: 5000::11/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::21/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::12/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::22/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::13/128 (IPv6 Node Address), L:0 N:1 E:0
    PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
      PPR Prefix: 5000::11/128
        ID: 6000:1::3/128 (Native IPv6)
        PDE: 5000::13/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::12/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::11/128 (IPv6 Node Address), L:0 N:1 E:0
    PPR: Fragment ID: 0, MT-ID: ipv4-unicast, Algorithm: SPF, F:0 D:0 A:0 U:1
      PPR Prefix: 5000::13/128
        ID: 6000:2::3/128 (Native IPv6)
        PDE: 5000::11/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::12/128 (IPv6 Node Address), L:0 N:0 E:0
        PDE: 5000::13/128 (IPv6 Node Address), L:0 N:1 E:0
  
```

The Wireshark screenshot (Figure 114) of the ISIS LSP sent by R11 shows the population of path identifiers by means of the ISIS protocol.

```

155 80.430182546 fa:16:3e:f2:18:c1 DEC-MAP-(or-OSI?)-Intermediate-System-Hello? ISIS HELLO 1464 P2P HELLO, System-ID: 0000.0000.0012
61 30.848861373 fa:16:3e:9b:02:be DEC-MAP-(or-OSI?)-Intermediate-System-Hello? ISIS LSP 1011 L1 LSP, LSP-ID: 0000.0000.0011.00-00
63 30.848878831 fa:16:3e:04:03:ef DEC-MAP-(or-OSI?)-Intermediate-System-Hello? ISIS LSP 1011 L1 LSP, LSP-ID: 0000.0000.0011.00-00
4
▶ Frame 61: 1011 bytes on wire (8088 bits), 1011 bytes captured (8088 bits) on interface 0
▶ IEEE 802.3 Ethernet
▶ Logical-Link Control
▶ ISO 10589 ISIS InTRA Domain Routeing Information Exchange Protocol
▼ ISO 10589 ISIS Link State Protocol Data Unit
  PDU length: 994
  Remaining lifetime: 1141
  LSP-ID: 0000.0000.0011.00-00
  Sequence number: 0x00000f56
  Checksum: 0x6665 [correct]
  [Checksum Status: Good]
  ▶ Type block(0x01): Partition Repair:0, Attached bits:0, Overload bit:0, IS type:1
  ▶ Protocols supported (t=129, l=2)
  ▶ Area address(es) (t=1, l=4)
  ▶ Multi Topology supported (t=229, l=4)
  ▶ Hostname (t=137, l=4)
  ▶ Router Capability (t=242, l=5)
  ▶ Traffic Engineering Router ID (t=134, l=4)
  ▶ Multi Topology IS Reachability (t=222, l=24)
  ▶ IP Interface address(es) (t=132, l=4)
  ▶ Multi Topology Reachable IPv6 Prefixes (t=237, l=52)
  ▶ PPR (t=155, l=139)
  ▶ PPR (t=155, l=139)
  ▶ PPR (t=155, l=161)
  ▶ PPR (t=155, l=161)
  ▶ PPR (t=155, l=117)
  ▶ PPR (t=155, l=117)
  } PPR
  
```

Figure 114: ISIS LSP.

The structure of PPR using Type-Length-Value (TLV) is described in [22] and shown in Figure 115 and Figure 116. It includes:

- (1) Tail-End/FEC Prefix, for the tail-end node’s routable prefix, prefix length, address family (IPv4 or Ipv6 as needed for some IGPs) and Multi-Topology ID this prefix belongs to.
- (2) Encoding of PPR-ID information, with the PPR-ID information, as the kind of data plane.
- (3) Description of ordered path, identifying the next hop on the path.
- (4) Optional sub-TLVs, for describing any other parameters of the path.

TLV Type (PPR)		Len in Octets		
FEC/Tail-End Prefix	Prefix Len	AF	MT	①
PPR-ID Information (Sub-TLV)				
PPR-FID1(Sub-TLV)	PPR-FID2	...	PPR-FID-n	③
Other PPR Sub-TLVs	...			④

Figure 115: PPR TLV structure.

```

▶ PPR (t=155, l=139)
▶ PPR (t=155, l=139)
▼ PPR (t=155, l=161)
  Type: 155
  Length: 161
  ▶ Flags: 0x1000, Ultimate
  Fragment: 0
  .... 0000 0000 0000 = Topology ID: Standard topology (0)
  IGP Algorithm: Shortest Path First (SPF) (0)
  ▶ Prefix (t=1, l=18): 5000::11/128 (1)
  ▶ ID (t=2, l=21) - Native IPv6: 6000:1::2/128 (2)
  ▼ PDE (t=3, l=110) - 5 items (3)
    PPR sub-TLV type: 3
    PPR sub-TLV length: 110
    ▶ PDE - 5000::13/128
    ▶ PDE - 5000::22/128
    ▶ PDE - 5000::12/128
    ▶ PDE - 5000::21/128
    ▶ PDE - 5000::11/128
  ▶ PPR (t=155, l=161)
  ▼ PPR (t=155, l=117)
    Type: 155
    Length: 117
    ▶ Flags: 0x1000, Ultimate
    Fragment: 0
    .... 0000 0000 0000 = Topology ID: Standard topology (0)
    IGP Algorithm: Shortest Path First (SPF) (0)
    ▶ Prefix (t=1, l=18): 5000::11/128
    ▶ ID (t=2, l=21) - Native IPv6: 6000:1::3/128
    ▼ PDE (t=3, l=66) - 3 items
      PPR sub-TLV type: 3
      PPR sub-TLV length: 66
      ▶ PDE - 5000::13/128
      ▶ PDE - 5000::12/128
      ▶ PDE - 5000::11/128
  ▶ PPR (t=155, l=117)
  
```

Figure 116: PPR TLVs

To verify that all routers installed the PPR-IDs for the paths they are part of (show isis ppr command):

**RT11:**

```

rt11# show isis ppr
Area Level ID Prefix Metric Position Status Uptime
-----
1 L1 6000:1::1/128 (Native IPv6) 5000::11/128 0 Tail-End - -
1 L1 6000:1::2/128 (Native IPv6) 5000::11/128 0 Tail-End - -
1 L1 6000:1::3/128 (Native IPv6) 5000::11/128 0 Tail-End - -
1 L1 6000:2::1/128 (Native IPv6) 5000::13/128 0 Head-End Up 23:27:12
1 L1 6000:2::2/128 (Native IPv6) 5000::13/128 0 Head-End Up 00:00:10
1 L1 6000:2::3/128 (Native IPv6) 5000::13/128 0 Head-End Up 00:00:40
  
```

**RT12:**

```

rt12# sh isis ppr
Area Level ID Prefix Metric Position Status Uptime
-----
1 L1 6000:1::1/128 (Native IPv6) 5000::11/128 0 Off-Path - -
1 L1 6000:1::2/128 (Native IPv6) 5000::11/128 0 Mid-Point Up 00:52:54
1 L1 6000:1::3/128 (Native IPv6) 5000::11/128 0 Mid-Point Up 00:53:24
1 L1 6000:2::1/128 (Native IPv6) 5000::13/128 0 Off-Path Down -
1 L1 6000:2::2/128 (Native IPv6) 5000::13/128 0 Mid-Point Up 00:40:40
1 L1 6000:2::3/128 (Native IPv6) 5000::13/128 0 Mid-Point Up 00:41:15
  
```

**RT13:**

```

rt13# sh isis ppr
Area Level ID Prefix Metric Position Status Uptime
-----
1 L1 6000:1::1/128 (Native IPv6) 5000::11/128 0 Head-End Up 00:55:19
1 L1 6000:1::2/128 (Native IPv6) 5000::11/128 0 Head-End Up 00:53:50
1 L1 6000:1::3/128 (Native IPv6) 5000::11/128 0 Head-End Up 00:42:11
1 L1 6000:2::1/128 (Native IPv6) 5000::13/128 0 Tail-End - -
1 L1 6000:2::2/128 (Native IPv6) 5000::13/128 0 Tail-End - -
1 L1 6000:2::3/128 (Native IPv6) 5000::13/128 0 Tail-End - -
  
```

**RT21:**

```
rt21# sh isis ppr
Area Level ID Prefix Metric Position Status Uptime
-----
```

Area	Level	ID	Prefix	Metric	Position	Status	Uptime
1	L1	6000:1::1/128 (Native IPv6)	5000::11/128	0	Mid-Point	Up	00:56:12
1	L1	6000:1::2/128 (Native IPv6)	5000::11/128	0	Mid-Point	Up	00:54:43
1	L1	6000:1::3/128 (Native IPv6)	5000::11/128	0	Off-Path	-	-
1	L1	6000:2::1/128 (Native IPv6)	5000::13/128	0	Mid-Point	Up	00:56:12
1	L1	6000:2::2/128 (Native IPv6)	5000::13/128	0	Mid-Point	Up	00:54:43
1	L1	6000:2::3/128 (Native IPv6)	5000::13/128	0	Off-Path	-	-

**RT22:**

```
rt22# sh isis ppr
Area Level ID Prefix Metric Position Status Uptime
-----
```

Area	Level	ID	Prefix	Metric	Position	Status	Uptime
1	L1	6000:1::1/128 (Native IPv6)	5000::11/128	0	Mid-Point	Up	00:57:09
1	L1	6000:1::2/128 (Native IPv6)	5000::11/128	0	Mid-Point	Up	00:43:26
1	L1	6000:1::3/128 (Native IPv6)	5000::11/128	0	Off-Path	-	-
1	L1	6000:2::1/128 (Native IPv6)	5000::13/128	0	Mid-Point	Up	00:57:09
1	L1	6000:2::2/128 (Native IPv6)	5000::13/128	0	Mid-Point	Up	00:55:40
1	L1	6000:2::3/128 (Native IPv6)	5000::13/128	0	Off-Path	-	-

### 5.2.2.2 Verification of the Forwarding Plane

A traceroute from RT11 to each PPR-ID (6000:2::1, 6000:2::2 and 6000:2::3) has been executed to ensure that the PPR paths were installed correctly in the network:

```
rt11# traceroute 6000:2::1
traceroute to 6000:2::1 (6000:2::1), 30 hops max, 80 byte packets
 1 4000:104::21 (4000:104::21) 1.380 ms 1.367 ms 1.361 ms
 2 4000:110::22 (4000:110::22) 1.802 ms 1.969 ms 1.976 ms
 3 6000:2::1 (6000:2::1) 2.912 ms 2.924 ms 3.210 ms

rt11# traceroute 6000:2::2
traceroute to 6000:2::2 (6000:2::2), 30 hops max, 80 byte packets
 1 4000:104::21 (4000:104::21) 1.201 ms 1.192 ms 1.186 ms
 2 4000:105::12 (4000:105::12) 1.724 ms 1.856 ms 1.861 ms
 3 4000:110::22 (4000:110::22) 2.620 ms 2.603 ms 2.619 ms
 4 6000:2::2 (6000:2::2) 3.136 ms 3.500 ms 3.498 ms

rt11# traceroute 6000:2::3
traceroute to 6000:2::3 (6000:2::3), 30 hops max, 80 byte packets
 1 4000:101::12 (4000:101::12) 2.736 ms 2.711 ms 2.693 ms
 2 6000:2::3 (6000:2::3) 2.675 ms 2.612 ms 2.593 ms
```

Using traceroute, to Host 2 from Host 1, it is also possible to see that the ICMP packets are being tunneled through the IS-IS network:

```
ubuntu@vm1:~$ traceroute 4000:108::2 -s 4000:103::1
traceroute to 4000:108::2 (4000:108::2), 30 hops max, 80 byte packets
 1 _gateway (4000:103::11) 1.285 ms 1.306 ms 1.290 ms
 2 * * *
 3 4000:108::2 (4000:108::2) 4.408 ms 4.389 ms 4.405 ms
```

Wireshark capture in R11 to verify that the traffic generated with iperf from Host 1 to Host 2 is being tunneled using GRE and following the {R11 - R21 - R22 - R13} path:

```
ubuntu@vm1:~$ iperf -V -c 4000:108::2
-----
Client connecting to 4000:108::2, TCP port 5001
TCP window size: 45.0 KByte (default)
-----
[ 3] local 4000:103::1 port 34090 connected with 4000:108::2 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0-13.1 sec 139 MBytes 89.2 Mbits/sec

ubuntu@vm2:~$ iperf -s -V
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
```

```

-----
[ 4] local 4000:108::2 port 5001 connected with 4000:103::1 port 34090
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-13.7 sec   139 MBytes  85.4 Mbits/sec
    
```

In the screenshot of Figure 117, it can be notice how the origin and destination of the GRE tunnel is the PPR-ID of the path R11 - R21 - R22 - R13:

No.	Time	Source	Destination	Protocol	Length	Info
204	142.596626075	4000:103::1	4000:108::2	TCP	1464	34090 → 5001 [ACK]
205	142.596629046	4000:103::1	4000:108::2	TCP	1464	34090 → 5001 [ACK]
206	142.598234494	4000:108::2	4000:103::1	TCP	138	5001 → 34090 [ACK]
207	142.598476487	4000:108::2	4000:103::1	TCP	138	5001 → 34090 [ACK]
208	142.598564118	4000:108::2	4000:103::1	TCP	138	5001 → 34090 [ACK]
209	142.598649973	4000:108::2	4000:103::1	TCP	138	5001 → 34090 [ACK]
210	142.598850122	4000:103::1	4000:108::2	TCP	1464	34090 → 5001 [ACK]
211	142.598876806	4000:103::1	4000:108::2	TCP	1464	34090 → 5001 [ACK]

```

▶ Frame 205: 1464 bytes on wire (11712 bits), 1464 bytes captured (11712 bits) on interface 0
▶ Ethernet II, Src: fa:16:3e:04:03:ef (fa:16:3e:04:03:ef), Dst: fa:16:3e:dd:3b:66 (fa:16:3e:dd:3b:66)
▶ Internet Protocol Version 6, Src: 6000:1::1, Dst: 6000:2::1
▶ Generic Routing Encapsulation (IPv6)
▶ Internet Protocol Version 6, Src: 4000:103::1, Dst: 4000:108::2
▶ Transmission Control Protocol, Src Port: 34090, Dst Port: 5001, Seq: 40821, Ack: 1, Len: 1326
▶ Data (1326 bytes)
▶ Frame 205: 1464 bytes on wire (11712 bits), 1464 bytes captured (11712 bits) on interface 0
▶ Ethernet II, Src: fa:16:3e:04:03:ef (fa:16:3e:04:03:ef), Dst: fa:16:3e:dd:3b:66 (fa:16:3e:dd:3b:66)
▼ Internet Protocol Version 6, Src: 6000:1::1, Dst: 6000:2::1
  0110 .... = Version: 6
  ▶ .... 0000 0000 .... .. = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... .. 0111 1011 0010 0111 1000 = Flow Label: 0x7b278
  Payload Length: 1410
  Next Header: Destination Options for IPv6 (60)
  Hop Limit: 64
  Source: 6000:1::1
  Destination: 6000:2::1
  ▶ Destination Options for IPv6
  ▶ Generic Routing Encapsulation (IPv6)
  ▼ Internet Protocol Version 6, Src: 4000:103::1, Dst: 4000:108::2
    0110 .... = Version: 6
    ▶ .... 0000 0000 .... .. = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .. 1011 0100 1101 0111 0110 = Flow Label: 0xb4d76
    Payload Length: 1358
    Next Header: TCP (6)
    Hop Limit: 63
    Source: 4000:103::1
    Destination: 4000:108::2
  ▶ Transmission Control Protocol, Src Port: 34090, Dst Port: 5001, Seq: 40821, Ack: 1, Len: 1326
  ▶ Data (1326 bytes)
    
```

Figure 117: Capture traffic R11.

As conclusion, PPR can emerge as potential encapsulation protocol following the source routing paradigm. Current implementation on FRR open source stack is operational and allow further experimentation. One of the most evident advantages of PPR if compared with other source routing solutions is the reduced overhead, which can be relevant for vertical cases that could be affected for processing capabilities of the routers underneath, or that could require large MTUs (which can be subject of fragmentation if using other options, depending on the topology scenario of deployment).

## 6 Fully Cloud-Native Implementation of OpenAirInterface RAN and Core

In this section we provide an overview of the work that is ongoing in the context of 5G-EVE to further advance cloud-native radio-access networks. We start by describing the development of `openairinterface-k8s`, a cluster deployment framework for OpenAirInterface (OAI) based on Kubernetes container platforms. It is a key element used in the Sophia Antipolis site and will include an extension with functional components in Paris in order to experiment with disaggregated RAN and Core over long distances. We then provide details of the computing and radio infrastructure used to prototype real-time 5G applications using `openairinterface-k8s`. We also provide an overview of the testing scenario that is currently being put in place to benchmark the feasibility of `openairinterface-k8s` under realistic conditions characteristic of private 5G networks.

The second part of this section deals with architectural redesign of `srsLTE` to demonstrate support for virtualized deployment. `srsLTE` is an open-source 4G software implementation of the 3GPP radio-access procedures designed for monolithic eNodeB deployments. The study here shows the steps to modify software task scheduling in order to parallelize processing in a manner more amenable to virtualized and potentially containerized implementations.

### 6.1 Current OAI Functional Decomposition Options (Radio-Access Network)

OpenAirInterface is a real-time implementation of the 4G/5G 3GPP protocol stack and comprises both core-network (`openairCN`) and radio-interface protocols (`openairinterface5g`). This overview considers primarily the radio-interface implementation whose functional entities and their interconnections are shown in Figure 118. This functional decomposition is similar to the Centralized-Unit/Distributed-Unit (CU/DU) split from 3GPP where the CU function is denoted here Radio Cloud Center (RCC) and the DU function is denoted here as the Radio Aggregation Unit (RAU).

In addition to the 3GPP functional decomposition, OAI supports an additional functional split known as the NF-API interface [57] which is specified by the small-cell forum. NF-API allows the physical layer to be executed in a different networking element than the layer-2 protocol stack. This is depicted in Figure 118. It is based on the original interface known as F-API which allowed software and chipset vendors to interoperate at the level of a system-on-chip but without an explicit networking specification between the two entities. The NF-API functional entities defined by the small-cell forum are shown in Figure 119. The first main functional entity is the so-called Virtualized Network Function (VNF), which comprises the layer2/3 protocol entities from the 3GPP CU and DU function (in OAI terminology, MACRLC, PDCP, RRC) as well as the networking interfaces with the 4G core network (S1C and S1U). The VNF terminology makes reference to the target implementation, namely cloud-native architectures, where the networking functions can be virtualized. The second main functional entity is the so-called Physical Network Function (PNF) which comprises the layer1 protocol and signal processing entities as well as the necessary interfaces for radio equipment. The small-cell forum specifies two primary interfaces to interconnect the PNF and VNF functions, P5 and P7. P5 is responsible for passing semi-static capability and configuration messages between the PNF and VNF. It is also responsible for controlling the activation/deactivation of the PNF and for radio-network discovery functions. The transport protocol used on the P5 interface is SCTP. P7 is responsible for exchanging real-time protocol information for every transmission-time interface (subframe in LTE, slot in 5G NR). The PDUs exchanged on P7 correspond to all of the 4G/5G transport channels and a configurable subset of the physical channels which are under dynamic control of the MAC layer. P7 uses the UDP transport protocol. NF-API was conceived to be robust to imperfect network conditions and potentially long-latency links. In the latter case, the underlying LTE network has to be configured to tolerate the added latency and suffers some throughput loss. An additional network management interface, P4, is also part of the NF-API specifications but not currently used in `openairinterface5g`.

The 5G NR version of NF-API is currently being specified by the small-cell forum based on the recently specified NR F-API interface [56], which comprises the same P5 and P7 interfaces as NF-API but with PDUs reflecting the transport and physical channels of 5G NR. An additional interface P19 is used to control active antenna

arrays in its underlying radio unit. The latter allows the MAC scheduler to influence the beam-pattern of the active antenna array, whether implemented in an analog or fully digital fashion.

OAI currently implements the following entities in the openairinterface5g repository

- a) MODEM comprises the basic functionality for preparing and processing the signals going to and coming from the radio units, which can be either local or remote. It implements the eNodeB/gNodeB OFDM Fourier transform operators for modulation/demodulation as well as arbitrary precoding functions for sharing physical antenna ports across several logical protocol instances. In addition, the interconnections between the MODEM entity and the remote-radio units (RRU) are Ethernet-based (UDP) and can be compressed in both directions. Computation can be partitioned between the central and remote units. In the current OAI implementation, the MODEM operations are referred to as radio units (RU).
- b) L1 comprises the remaining physical layer eNodeB/gNodeB procedures. This block receives protocol data units (PDU) on the LTE transport channels and implements the mapping to downlink physical channels (BCH/PBCH, DCI/PCFICH-PDCCH, HI/PHICH, DL-SCH/PDSCH, MCH/PMCH) based on the configurations received from the MACRLC for each PDU. It also generates indications for the received PDUs from the uplink physical channels (PUCCH/SR-HARQ-CQI, PUSCH/UL-SCH-CQI-HARQ, PRACH/RACH, SRS) The input and output of this entity on the interface with the RU (MODEM) are physical signals in the frequency-domain. The input and output of this entity on the transport channel interface are structured packets according to the NFAPI specifications for the so-called P7 interface. L1 also receives configuration information from the MACRLC entity in the form of structured packets according to the NFAPI P5 interface specifications. These information elements originate in the RRC entity and are relayed by the MACRLC.
- c) MACRLC implements the 3GPP Medium-Access Control (MAC) and Radio-Link Control (RLC) procedures. This block receives control and user-plane PDUs from the Radio Resource Controller (RRC) (Master and System Information) and Packet Data Convergence Protocol (PDCP) (Signalling and Data Radio Bearers) entities as well as configuration from the RRC entity (common and dedicated radio-resource configurations). These interfaces follow the 3GPP specifications for the so-called F1-U and F1-C interfaces. The configuration information contains information elements both for the MACRLC entity and the L1 entity.
- d) PDCP implements the 3GPP Packet Data Convergence protocol. It receives information from the LTE-RRC (signalling radio bearers) and core network via the S1-U interface. The interface with LTE-RRC is OAI-specific but will later follow the 3GPP specifications for the so-called E1 interface.
- e) RRC implements the 3GPP Radio Resource Control protocol. It receives non-access-stratum (NAS) information from the core network (MME) via the S1-C interface and provides configuration information to PDCP and MACRLC. Signalling information is transported by PDCP (signalling radio-bearers) and transparently by MAC (Master and System Information).

The various entities and their functional decomposition allow experimenters to evaluate performance under different deployment scenarios and virtualization paradigms. Some of these are reported in the following sections using Kubernetes (k8s).





## 6.1.1 Cloud-native Deployment of OAI

During 2019, EURECOM and RedHat have developed and currently maintain a cloud-native flavour of OAI known as `openairinterface-k8s` [63] in order to test real-time container-based radio-network innovations in a k8s environment. It is a fully open-source implementation provided under an Apache V2.0 license. This is an integral part of the Sophia Antipolis 5G-EVE network infrastructure and in the process of being extended to Paris in a small portion of Orange's 5G-EVE network infrastructure. It is also used in the context of the Linux Networking Foundation OPNFV VCO 3.0 project [58] and soon the Linux Foundation AKRAINO project. It is a fully containerized version of a subset of the OAI software modules, both the RAN entities described earlier in this section and the core network entities [62] are included in `openairinterface-k8s`. It was primarily designed to make use of the OpenShift 4 k8s implementation provided by RedHat. More specifically, `openairinterface-k8s` currently includes the 4G core network entities (Enhanced Packet Core or EPC) with the so-called control-plane user-plane separation (CUPS) using the 3GPP PFCP protocol between the combined S/P-Gateway control (SPGW-C) and user-plane (SPGW-U) functions. The Mobility Management Entity (MME) is release-15 compliant so it allows 5G Non-Standalone (NSA) operation. Finally, the Home Subscriber Station (HSS) and an associated Cassandra database are provided. The current RAN entities include the 4G Rel-15 eNodeB and Rel-15 NSA gNodeB. For testing purposes, the OAI UE and RAN emulator can also be deployed via `openairinterface-k8s`.

`Openairinterface-k8s` is fully self-contained in the sense that it includes scripts to pull the source code from the master OAI repositories, to build container images for Redhat Enterprise Linux (RHEL) and to create a single namespace with the networking interconnections for deployment on a single k8s cluster. It has been enhanced by the community to include support for non-RHEL systems such as CentOS and Ubuntu although in the work reported in this deliverable the target environment was OpenShift 4.2 using RHEL images with real-time support. The real-time support is critical for containerized RAN functions.

As shown in Figure 120, `openairinterface-k8s` comprises hard and soft real-time components. The hard real-time components are monolithic instantiations of the 4G eNodeB and 5G NSA gNodeB functions, each comprising both the RCC or CU entities (RRC,PDCP), and the MACRLC and L1 entities for LTE and NR in two k8s pods (one eNodeB and one for gNodeB). These are deployed on worker nodes with the real-time variant of the Redhat Enterprise Linux (RHEL) operating system (RHEL 7.6 currently) in order to provide deterministic processing with minimal and manageable operating-system overhead for Layer 1 5G functions. Both the gNodeB and eNodeB functions require multiple network interfaces

- 2x10G physical Ethernet for radio fronthaul
- Virtual Ethernet for X2C interconnection between eNodeB and gNodeB
- Virtual Ethernet for S1C (eNodeB – MME interconnection)
- Virtual Ethernet for S1U (eNodeB and gNodeB interconnection with SPWGu)
- Virtual Ethernet for internode communications in the cluster

In order to support multiple network interfaces in a single pod, `openairinterface-k8s` makes use of the Multus container network interface (CNI) plugin for k8s.

The real-time components have various fronthaul interfaces with the radio nodes and can make use of dedicated 10Gbit/s Ethernet links between certain worker nodes to radio units (e.g. USRPs described in Section 6.1.3) or using the cluster network itself using the native OAI 7.1 fronthaul interface (IF4p5). Two additional types of 4G/5G radio-units with industry-grade fronthaul solutions (eCPRI, ORAN 7.2) requiring DPDK are being integrated with `openairinterface-k8s` at the time of this writing.

The core network components execute on regular soft real-time pods. They also make use of multiple network interfaces to support the S1C, S1U, S11, S6A, SxA (PFCP protocol) and SGI interfaces ensuring the communications between the different core network entities and outside networks (local breakout/edge network, internet, etc.). The initial implementation of `openairinterface-k8s` was presented and demonstrated at the 2019 Kubecon Conference [64] using the 5G-EVE Sophia Antipolis site.

### 6.1.1.1 Openairinterface-k8s Evolution for Disaggregated RAN and Core Network

In order to demonstrate a fully disaggregated Core Network and RAN, the current Sophia Antipolis deployment is being migrated to the one described in Figure 121. Under the same cluster orchestrator (OpenShift 4.2), we

add worker nodes in a remote location, Orange Paris in the 5G-EVE context, which can be used to disaggregate both the Core Network and RAN. In the figure, we see that the control-plane core network functions (MME, HSS, SPGWc) and control-plane radio-network functions (CU-C) are located on the workers at the remote location, differently from the RAN and Core Network user-plane functions (CU-U, SPGWu) and Layer1/2 RAN functions (DU-eNB, DU-gNB). This requires an additional 3GPP RAN control-plane interface (F1C). One of the interesting aspects of this experimental disaggregated RAN will be to investigate the feasibility of geographically separating the control functions using a lower-speed long-distance link under the control of a common cluster orchestrator, OpenShift 4 in our case.

We envisage also to include additional user-plane functions (RAN and Core) in the remote network in Paris towards the end of the 5G-EVE project.

### 6.1.1.2 Support for ONAP Orchestration in openairinterface-k8s

Recently, support has been added for interfacing openairinterface-k8s elements with the ONAP Dublin/EI Alto cluster deployed by Orange in Paris. This requires wrapping the openairinterface-k8s pods in so-called Helm charts. Helm [59] combined with the Multicloud-k8s Plugin-service API [60] is the chosen technique for deploying k8s network-functions from a remote location. Openairinterface-k8s now includes Helm charts for the SPGWc and SPGWu pods in order to perform the ONAP integration work. After completing the integration of these elements, which is planned for end June 2020, the Helm-charts for the remaining network functions will be created to allow complete automation of the Sophia Antopolis RAN and Core Elements from ONAP and hence the 5G-EVE portal. To the best of our knowledge, the Sophia Antipolis site is the sole site in 5G-EVE to have automated containerized network function deployment and one of very few test networks for ONAP with containerized network functions. It is most likely the only public one including RAN functions. The companion VCO 3.0 site in Montreal (Kaloom Networks) has recently demonstrated ONAP automated deployment of a 4G NSA commercial Core Network [61] with RAN emulators.

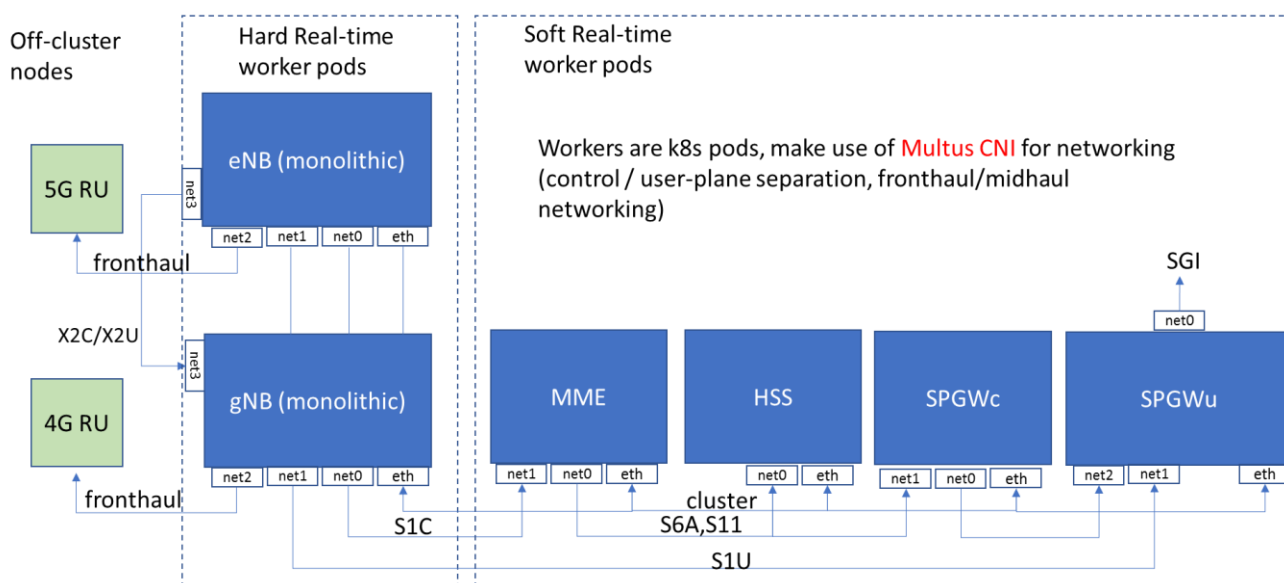


Figure 120: Current openairinterface-k8s functional decomposition

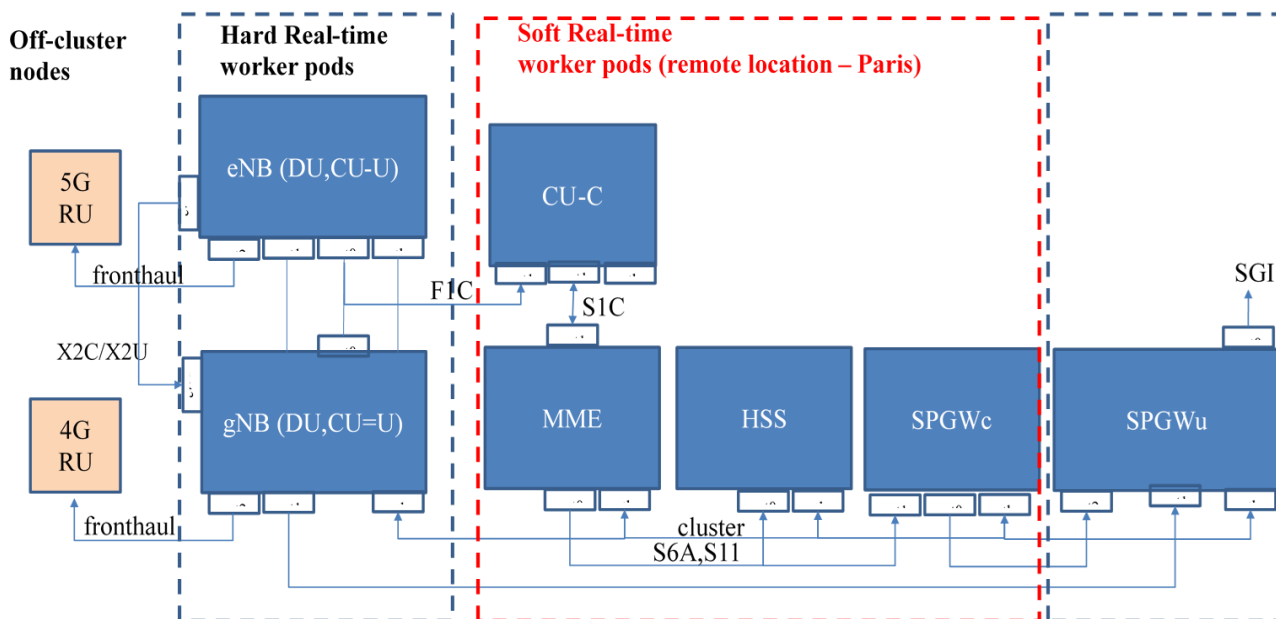


Figure 121: Evolution of openairinterface-k8s

## 6.1.2 Experimentation Infrastructure

The experimentation infrastructure put in place at EURECOM was designed to be able to test OAI’s cloud-native implementation. It consists of high-end compute nodes, a Terabit switching fabric and optically-interconnected radio-units distributed in a mixed indoor-outdoor environment and somewhat characteristic of a private 4G/5G network.

### 6.1.2.1 Computing and Switching

The current computing cluster and sandbox area are shown in Figure 122. The cluster’s switching fabric currently make use of three Edgecore 7312-54XS Data Center switches, running CumulusOS, interconnected in a leaf-spine configuration with 6x100 Gbit/s between the three switches. The two leaf switches driver a rack of Intel Architecture servers. These include three Dell R440 (Xeon Silver 20-core 2.4 GHz) and 4 Dell R640 (Xeon Gold 6154 36-core 3 GHz). One of the worker nodes is equipped with dedicated 4x10 Gbit/s Ethernet for driving two 5G radio units in a point-to-point fashion. The remainder of the nodes are connected to the switch fabric for interconnection.

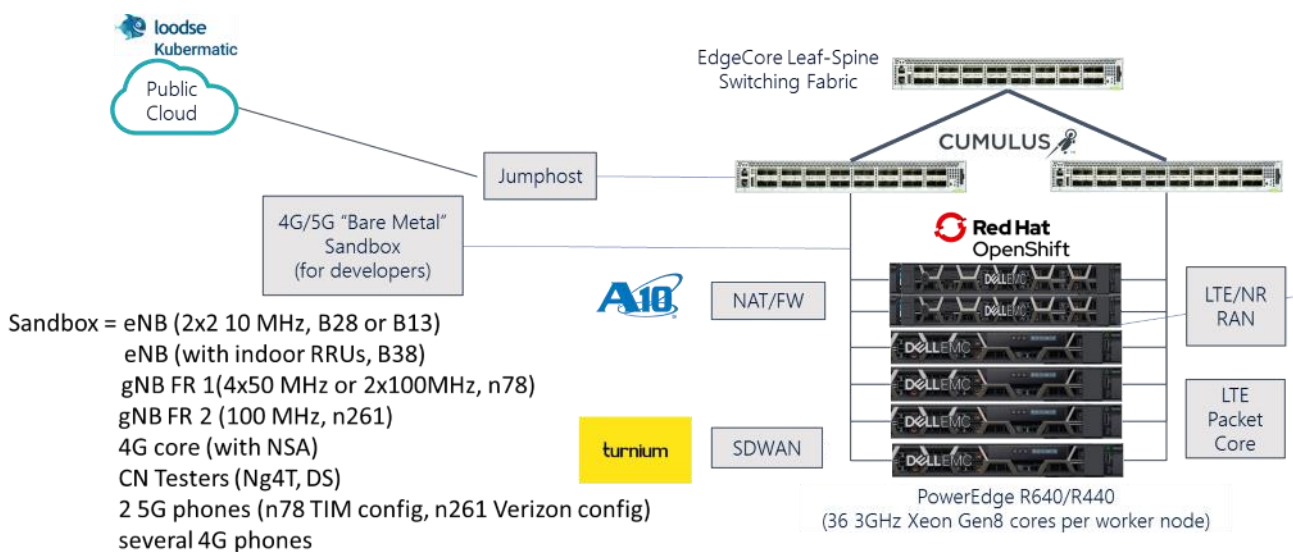


Figure 122: Computing cluster

Some bare-metal nodes (sandbox) are equipped with in-lab 5G-capable radio devices (FR1 and FR2) and are available as a sandbox that can be used by experimenters and developers and are interconnected with the k8s cluster. External access for onboarding software, collecting measurement data and developing basic software for the site is available for partners using secure-shell access. The nodes of the site are also used by OpenAir-Interface Jenkins-based continuous integration / continuous delivery (CI/CD) framework.

### 6.1.3 Experimental Radio units

The outdoor radio units and their internal architecture are depicted in Figure 123. These are made up of commercially-available off-the-shelf RF and baseband components. The baseband to RF converters are two National Instruments USRP N310 which provide eight total RF chains (the lower part of Figure 123 depicts one of the two N310 devices). They are synchronized via a GPS reference driven by an external antenna. The 8 RF chains drive two custom-made amplification units manufactured by Zhixun Wireless. These are TDD radios for 3.3-3.8 GHz consisting of 2 Watt power amplifier output power per RF port, 20 dB low-noise amplifiers and TX/RX switching. The amplifier TX/RX ports are connected to two Kathrein 3.5 GHz panel antennas. The radio unit fronthaul interface is UHD (USRP hardware driver) over 2x10 Gbit/s Ethernet, which provides a point-to-point link with one of the worker nodes in the k8s cluster. UHD is a time-domain I/Q sample fronthaul protocol. An additional management interface is provided to control remotely monitor the radio-units or to upgrade their firmware.

The indoor radio units are shown in Figure 123 and are interconnected with the cluster using the native OAI 7.1 fronthaul protocol IF4p5 over 1 Gbit/s copper PoE+ links. There are 12 deployed indoor units which are interconnected via two PoE+ switches, which aggregate the RRU traffic and provide interconnection with the k8s cluster via 2x10Gbit/s fiber links. The RRUs can be driven by any of the nodes in the k8s cluster.

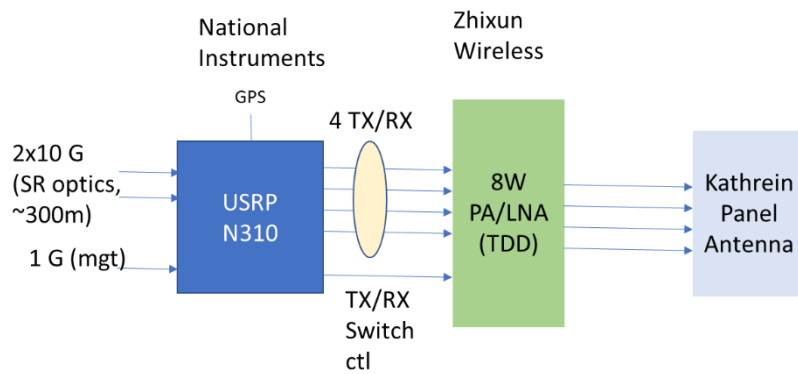


Figure 123: Outdoor 3.5 GHz Radio units

### 6.1.4 Experimental Validation Scenario

The current testing scenario for openairinterface-k8s in a live deployment in Sophia Antipolis is shown in Figure 125.

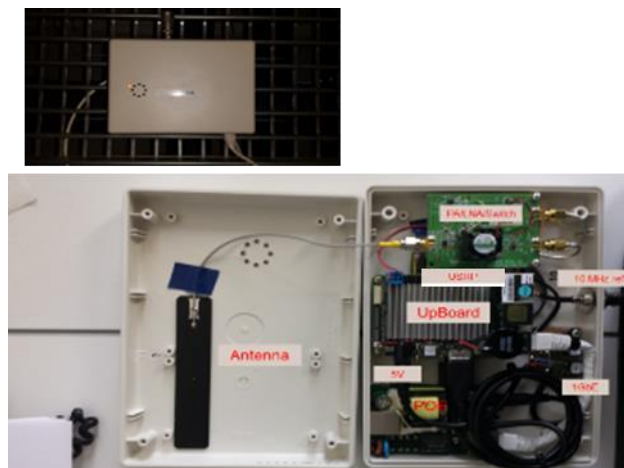
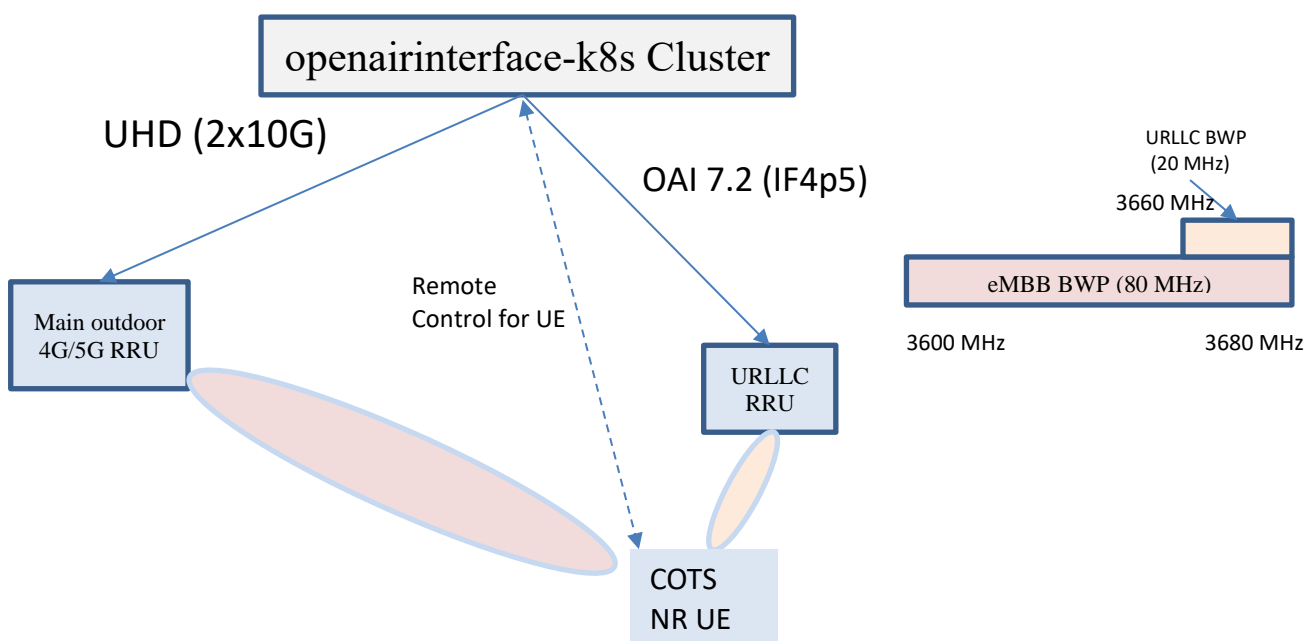


Figure 124: Indoor 2.6/3.5 GHz radio units

The scenario comprises an NSA system with eNodeB/gNodeB configured with two radio units, one outdoor (Band n78 NR, Band 38 LTE) with maximum 80 MHz bandwidth and one indoor 3.5 GHz with maximum 20 MHz bandwidth. Two bandwidth parts are configured, one for standard eMBB services with 30 kHz subcarrier spacing and a second for URLLC with 60 kHz subcarrier spacing. It consists of a standard COTS 5G NSA UE for performance measurement. Both current smartphones (OPPO Reno 5G, Galaxy S20) and embedded modules (Quectel RM500Q-GL) are being used and can be connected via WIFI or Ethernet for control, KPI monitoring and measurement trace collection on the UE side.

The main objectives of this performance evaluation scenario are:

- Testing of multi-numerology configurations, in particular self-interference management and scheduling.
- Efficiency of different URLLC TDD framing configurations, in particular DL/UL periodicity, mini-slot usage.
- Ability to satisfy low-latency constraints for URLLC with openairinterface-k8s and recent DPDK/Ethernet-based fronthaul protocols. This involves proper tuning of the real-time operating and is part of an ongoing joint effort with RedHat in the context of OPNFV.



**Figure 125: URLLC Testing Scenario**

## 7 Segment Routing over IPv6: Linux Data Plane improvements for Performance Monitoring

Segment Routing over IPv6 (SRv6 in short) is a networking solution for IP backbones and datacenters, which has been recently adopted in several of large scale network deployments. The SRv6 research, standardization and implementation activities are going on at a remarkable pace. In particular, a number of Internet Drafts have been submitted related to the Performance Monitoring (PM) of flows in an SRv6 network. In this section we discuss the proposed PM approaches, considering both data plane and control plane aspects and focusing on loss monitoring. Then we describe the implementation of a per-flow packet loss measurement (PF-PLM) solution based on the “alternate marking” method. Our implementation is based on Linux kernel networking and it is open source. We describe a platform that can be used to validate the standardization proposals from a functional perspective and the implemented solution from the performance point of view. We analyze three different design choices for the implementation of PF-PLM and evaluate their impact on the maximum forwarding throughput of a software based (Linux) router.

### 7.1 SRv6 and SRv6 PM (Performance Monitoring)

Segment Routing for IPv6 (SRv6 in short) is the instantiation of the Segment Routing (SR) architecture [23,24] for the IPv6 dataplane. SR is based on loose source routing: a list of *segments* can be included in the packet headers. The segments can represent both topological way-points (nodes to be crossed along the path towards the destination) and specific operations on the packet to be performed in a node. Example of such operations are: encapsulation and decapsulation, lookup into a specific routing table, forwarding over a specified output link, Operation and Maintenance (OAM) operations like time-stamping a packet. More in general, arbitrarily complex behaviors can be associated to a segment included in a packet. In SRv6, the segments are represented by IPv6 addresses and are carried in an IPv6 *Extension Header* called *Segment Routing Header* (SRH) [25]. The IPv6 address representing a segment is called SID (Segment ID). According to the SRv6 *Network Programming* concept [26], the list of segments (SIDs) can be seen as a “packet processing program”, whose operations will be executed in different network nodes. The SRv6 Network Programming model offers an unprecedented flexibility to address the complex needs of transport networks in different contexts like 5G or geographically distributed large scale data centers. With the SRv6 Network Programming model it is possible to support valuable services and features like layer 3 and layer 2 VPNs, Traffic Engineering, fast rerouting. A tutorial on SRv6 technology can be found in [27]. The standardization activities for SRv6 are actively progressing in different IETF Working Groups, among which the SPRING (Source Packet Routing In NetworkinG) WG is taking a leading role. Recently, several large scale deployments in operator networks have been disclosed, as reported in [28]. Performance Monitoring (PM) is a fundamental function to be performed in SRv6 networks. It allows to detect issues in the QoS parameters of active flows that may require immediate actions and to collect information that can be used for the offline optimization of the network. The most important performance parameters that need to be monitored are packet delay and packet loss ratio. A number of Internet Draft are under discussion in the IETF SPRING WG related to Performance Monitoring of flows in an SRv6 network (SRv6 PM in short). These drafts rely on existing work for performance measurement in general IP and MPLS networks and extend them for the SRv6 PM case. In this section, we study and discuss the proposed SRv6 PM approaches, considering both data plane and control plane aspects. The first goal of our work is to support the standardization activity by building an open source platform for validation and comparison of proposed SRv6 PM solutions. The platform should be usable also to evaluate specific design and implementation choices through testbed experiments. In the longer term the specific PM solutions that we have implemented can become an asset for SRv6 PM on Linux routers and hosts in production. To this aim we designed and describe our open source implementation of per-flow packet loss measurement (PF-PLM), based on Linux kernel networking. To the best of our knowledge no open source implementation of SRv6 PM mechanism is available. The proposed solution relies on the “alternate marking” method described in RFC 8321 [29] and provides an accurate estimation of flow level packet loss (it achieves single packet loss granularity). We discuss the processing load aspects of PF-PLM in Linux software routers, comparing two implementations based on different design choices.



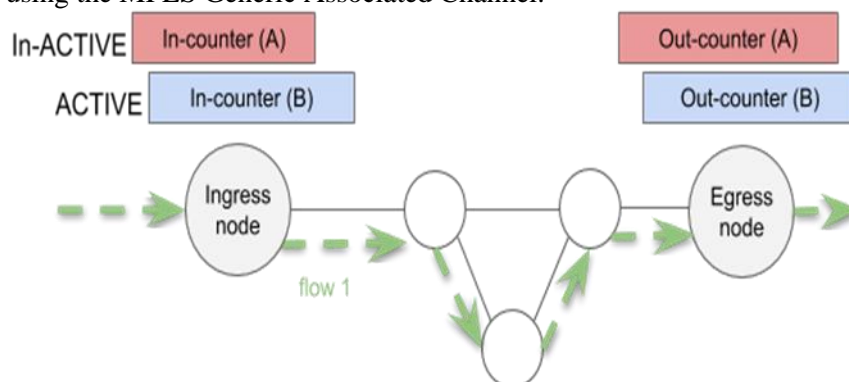
## 7.2 Performance Monitoring methodologies and their standardization

In this subsection we introduce the relevant standards for PM in IP and MPLS networks and then we discuss the solutions for SRv6 network proposed in the IETF standardization.

### 7.2.1 IP and MPLS networks

Being able to monitor the performance metrics for packet loss, one-way delay and two-way delay is fundamental for a service provider. RFC 4656 “The OneWay Active Measurement Protocol (OWAMP)” [30] provides capabilities for the measurement of one-way performance metrics in IP networks, like one-way packet delay and one-way packet loss. RFC 5357 “Two-Way Active Measurement Protocol (TWAMP)” [31] introduces the capabilities for the measurements of two-way (i.e. round-trip) metrics. These specifications describe both the *test protocol*, i.e. the format of the packets that are needed to collect and carry the measurement data and the *control protocol* that can be used to setup the test sessions and to retrieve the measurement data. For example OWAMP defines two protocols: “OWAMP-Test is used to exchange test packets between two measurement nodes” and “OWAMP-Control is used to initiate, start, and stop test sessions and to fetch their results” (quoting [30]). Note that in general there can be different ways to setup a test session: the same test protocol can be re-used with different control mechanisms.

RFC 6374 [32] specifies protocol mechanisms to enable the efficient and accurate measurement of performance metrics in MPLS networks. The protocols are called LM (Loss Measurement) and DM (Delay Measurement). We will refer to this solution as MPLS-PLDM (Packet Loss and Delay Measurements). In addition to loss and delay, MPLS-PLDM also considers how to measure throughput and delay variation with the LM and DM protocols. Differently from OWAMP/TWAMP, RFC 6374 does not rely on IP and TCP and its protocols are streamlined for hardware processing. While OWAMP and TWAMP support the timestamp format of the Network Time Protocol (NTP) [33], MPLSPLDM adds support for the timestamp format used in the IEEE 1588 Precision Time Protocol (PTP) [34]. There are several types of channels in MPLS networks over which loss and delay measurement may be conducted. Normally, PLDM query and response messages are sent over the MPLS Generic Associated Channel (G-ACh), which is described in detail in RFC 5586. RFC 7876 [35] complements the RFC 6374, by describing how to send the the PLDM response messages back to the querier node over UDP/IP instead of using the MPLS Generic Associated Channel.



**Figure 126: Alternate coloring method (RFC 8321)**

Let us now focus on the procedures to monitor the packet loss experienced by a flow, from an ingress node to an egress node. In general, it is necessary to count the number of packets belonging to the flow that are sent by the ingress node and the number of packets that are received by the egress node in a reference period and then make the difference between the two counters. If we want to achieve the granularity to accurately detect single packet loss events, while the flow is active, we need to properly consider the “in flight” packets, e.g. the packets counted by the ingress node but not by the egress one. The presence of in flight packets makes it difficult to obtain an accurate evaluation the number of lost packet, as discussed in RFC 8321 [29]. The solution proposed by RFC 8321 is called *alternate marking* method, as shown in Figure 126. It consists in coloring (marking) the packets of the flows to be monitored with at least two different colors, with the colors that alternate over time. For example, a continuous block of packets of a flow is colored with color A (e.g. for a configurable duration T), then the following block of packets is colored with color B (again for a duration T), and so on. Separate counters are needed in the ingress node and in the egress node to count the flow packets colored with

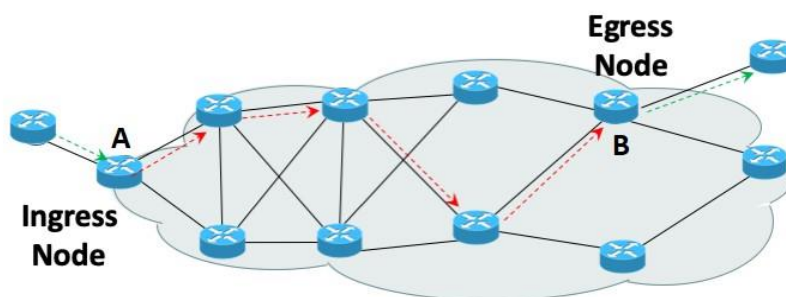
color A and with color B. In the time interval in which the packets are colored with color B, it is possible to read counters for color A from the ingress and egress nodes, and evaluate their difference which exactly corresponds to the number of lost packet in the previous interval. RFC 8321 describes a generic method that can be applied to different networks. When the alternate marking method is applied to a specific network, the mechanism to color the packets must be specified. The motivations for choosing the alternate marking method and its advantages are further described in the introduction section of RFC 8321 [29].

## 7.2.2 SRv6 networks

The standardization activity regarding the Performance Monitoring of SRv6 networks is very active. Two internet drafts have been proposed that extend the two families of Performance Measurement solutions discussed in the previous subsection (OWAMP/TWAMP and MPLS-PLDM):

- Performance Measurement Using TWAMP Light for Segment Routing Networks [36]
- Performance Measurement Using UDP Path for Segment Routing Networks [37]

We propose a solution based on the extension of the TWAMP Light protocol defined in RFC 5357 Appendix I and its simplified extension Simple Two-way Active Measurement Protocol (STAMP), proposed for standardization in [38].



**Figure 127: Reference SRv6 network scenario for Performance Monitoring**

Both solutions provide the possibility of measuring delay and loss of a single SRv6 flow, characterized by a SID list. The data collection takes place with test UDP packets transmitted on the same measured path. The test UDP packets collect the one-way or two-way PM data and makes them available to the node which requested the measurement.

The solution in [37] is likely a bit more mature (its standardization started in March 2018) and it includes more features with respect to the TWAMP based one. For example it includes a Return Path TLV used to carry reverse SR path information. It also defines a packet to collect counters for loss measurement and timestamps for delay measurement with a single message.

## 7.3 SRV6 Loss Measurement

In this subsection we detail a Loss Measurement (LM) solution compliant with the two proposed drafts ([36] and [37]). Our reference network scenario is depicted in Figure 127. We have an SRv6 network domain. IP traffic arrives at an ingress edge node of the network where it can be classified and encapsulated in an SRv6 flow. In SRv6 terminology, an *SR policy* is applied to the incoming packets. The SR policy corresponds to a *SID List* that is included in the Segment Routing Header (SRH) of the outer IPv6 packet. The outer IPv6 packet crosses the network (according to its SID list) and arrives to the egress edge node where the inner IP packet is decapsulated (the outer IPv6 and SRH are removed). For example in Figure 127, node A acts as ingress node while node B as egress node for the green packets. The ingress node A applies the SR policy, i.e. it writes the *SID list* into the SRH header.

### 7.3.1 Packet Counting

In order to perform Per-Flow Loss Measurement we need to implement packet counters associated to SRv6 flows, both in the ingress node and in the egress node. For our purposes, an SRv6 flow corresponds to an SR

policy, i.e. to a SID List. We want to be able to explicitly activate the counting process for a set of flows (identified by their SID Lists). In an ingress node this means to process all outgoing SRv6 packets and count the packets belonging to the set of monitored flows (by comparing the SID List of the outgoing packets with the SID Lists of the monitored flows). Likewise, in an egress node, this means to process all incoming SRv6 packets, check if the packets belong to the set of monitored flows and increment the counters as needed.

These counting operations can have a high computing cost for a software router, so it is important to carefully design their implementation (see Section 7.4 to evaluate their impact on the processing performance).

### 7.3.2 Traffic Coloring

The Internet drafts [36] and [37] do not specify how the coloring must be implemented and thus several solutions are possible. In case of SRv6, we considered two solutions: i) modification of the DS field, previously known as IP Type of Service (TOS) field; ii) encoding the color in a SID of the SID list present in the SRv6 header.

The first solution is simple but has some drawbacks: the number of bits available in the DS field is limited (6 or 8) and they are considered precious. Using two colors we need a bit, in addition we can use a second bit to differentiate between colored traffic to be monitored and uncolored traffic not to be monitored. This can be useful to avoid comparing the full SID List to decide if a packet is part of a flow under monitoring or not. In our current implementation, described in Section 7.4, we have used two bits of the DS field.

The solution that encodes the color in a SID, exploits the fact that according to [26] an IPv6 address representing a SID is divided in LOCATOR:FUNC:ARGS. The LOCATOR part is routable and allows to forward the packet towards the node where the SID needs to be executed. The FUNC and ARG parts are processed in the node that executes the SID. In particular, the ARG part can reserve a number of bits for the alternate marking procedures. This may allow using more than two colors. This solution however has an implementation drawback: due to the variable position of these bits, implementing an hardware processing solution is much harder and can be out of reach for current chips that need to operate at line speed. Moreover periodically changing the ARG bits in a SID of a running flow can cause an interference with the SRv6 forwarding plane (e.g., for Equal Cost MultiPath) when the SID is used as IPv6 destination addresses.

### 7.3.3 Data Collection

The two Internet drafts proposed for the PM of SRv6 specify the use of dedicated protocols for the collection of meters and the loss evaluation. Both standards are based on the sending of a UDP packet (query) by the ingress node (called Sender in [36]) to the egress node (called Reflector in [36]). The packet is used to collect the counters of a given color, in order to evaluate the loss. If path monitoring is bidirectional, the Reflector sends to the Sender a response packet that goes through the network in the reverse direction, collecting the counters of the return path. The draft [36] specifies that the query and the response packets must comply with the TWAMP light or STAMP protocol format. Both packets use a fixed structure.

The UDP query packet includes the following fields:

- Sender Sequence Number: a 32 bit number incremented by one each query message;
- Block Number: The color of the direct path;
- Sender Counter: the output counter of the Sender.

The UDP response packet includes the following fields:

- Receiver Sequence Number: a 32 bit number incremented by one each response message;
- Receiver Counter: the input counter of the direct path;
- Transmit Counter: the output counter of the Reflector;
- Block Number: the color of the reverse path;
- the three fields of the Query Packet.

## 7.4 Linux Implementation

The Per Flow Packet Loss Monitoring (PF-PLM) system has been realized in Linux extending the kernel based SRv6 implementation and different frameworks for packet processing, namely Netfilter/Xtables and *IP set*. Furthermore, an additional implementation of PF-PLM is provided by exploiting the packet processing capabilities of the extended Berkeley Packet Filter (eBPF) Virtual Machine in combination with some facilities made available in the networking data path.

At first, we will present and compare a simpler solution based on *iptables* a more efficient one based on *IP set*. Subsequently, we will compare the performance achieved through *IP set* with that of eBPF, showing how the latter is particularly suitable for creating high-performance network flow monitoring tools.

All the software components we have developed are available as open source [39]. In the following subsections we will provide some basic tutorials on the tools that we have extended and the descriptions of our contributions.

### 7.4.1 Linux SRv6 subsystem

The Linux kernel SRv6 subsystem [40] supports the basic SRv6 operation described in [25] and most of the operations defined in [41]. A Linux node can classify incoming packets and apply SRv6 policies, e.g., encapsulate the packets with an outer packet carrying the list of SRv6 segments (SIDs). A Linux node can associate a SID to one of the supported operations, so that the operation will be executed on the received packets that have such SID as IPv6 Destination Address. More details on the Linux SRv6 implementation with a list of the currently supported operations can be found in [42].

### 7.4.2 Linux Netfilter/Xtables/iptables subsystem

The kernel-space Netfilter/Xtables allows the system administrator to insert chains of rules for the processing of packets inside predefined tables (raw, mangle, filter, nat). Each table is associated with different types of packet processing operations. In each chain, packets are processed by sequentially evaluating the rules in the chains. As shown in Figure 129, there are 5 processing phases (PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING) in which the default chains associated with specific tables are processed. Moreover, it is possible to create additional chains as needed. For example, in the POSTROUTING phase, the default postrouting chains associated to the mangle and nat table are processed. *iptables* is a user-space CLI (Command Line Interface) utility program that allows a system administrator to configure the tables/chains/rules provided by the Netfilter/Xtables subsystem. Figure 129 shows two additional chains (BLUE-CHAIN, RED-CHAIN) that we added in our packet loss monitoring solution, visible in the rightmost part of the figure. For simplicity we will use *iptables* to refer in general to the Netfilter/Xtables/iptables subsystem, including the IPv6 specific modules.

*iptables* is highly modular and can be extended. In particular, it is possible to develop a custom packet matching module, to specify a rule which refers to the custom module name and includes extra commands that depend on the specific extension. We have followed this approach, as described in Section 7.4.4.

### 7.4.3 Iptables based PF-PLM implementation

In order to evaluate Per-Flow Packet Loss metrics, we need to collect information on both Ingress and Egress nodes in a coordinated way. In the ingress node, the traffic is classified (based on IP Destination Addresses) and can be associated to an SRv6 policy (step 1 in Figure 128). This means that a matching packet can be encapsulated in an outer IPv6 packet with a new IPv6 header followed by the Segment Routing Header with the proper Segment List (SID List) (step 2 in Figure 128). The encapsulated packet continues its journey in the networking stack until he reaches the POSTROUTING chain of the mangle table (step 3 in Figure 128). In this chain, we put a “jump” rule with the aim to divert all SRv6 traffic to a custom chain for statistic collection purposes (step 4 in Figure 128). In particular, we have two custom chains, referred as *coloring chains*, as we need to use two colors for traffic marking according to the “alternate marking” approach. Each chain contains a set of rules, each rule is used to match on a specific SID list included in the SRH header. At any given time, only one of the two chains is *active* and is referred to by the “jump” rule.

When an SRv6 packet enters the active *coloring chain*, the rules contained in the chain are applied to the packet in sequence until: 1) There is a rule that matches the SID list of the packet. The match counter of this rule is incremented. The packets needs to be colored with the color corresponding to the active coloring chain. When the coloring based on DS field is used, the *color bit* of the DS field will be set to the proper color (0 or 1). Once this is done, the packet processing comes back to the calling chain, i.e. the POSTROUTING chain in the mangle table. 2) All rules have been considered and no match with SID list of the packet has been found, so the packet processing simply jumps back to the calling POSTROUTING chain.

In the egress node, the packets of the flows to be monitored enter from an incoming interface and arrive encapsulated in an IPv6 outer header. Therefore we need to match on the SID List of incoming packet. We use the PREROUTING chain of the mangle table to match on the different colors, considering the color bit of the DS

field (step 1 of Figure 129). Then for each color, we process the packet in a chain which includes one rule for each SID List that we want to monitor (step 2 of Figure 129). We refer to this custom chain as *color-counting chain*. When a rule matches the SID List in the packet, the match counter of the rule is incremented. After the processing in the color-counting chain, the normal packet processing continues. As we are in the egress node, the destination address will be a SID that corresponds to a packet decapsulation operation. This SID is matched in the routing operation (step 3 of Figure 129), then the decapsulation operation is executed by the SRv6 kernel processing (encap seg6local, step 4 of Figure 129).

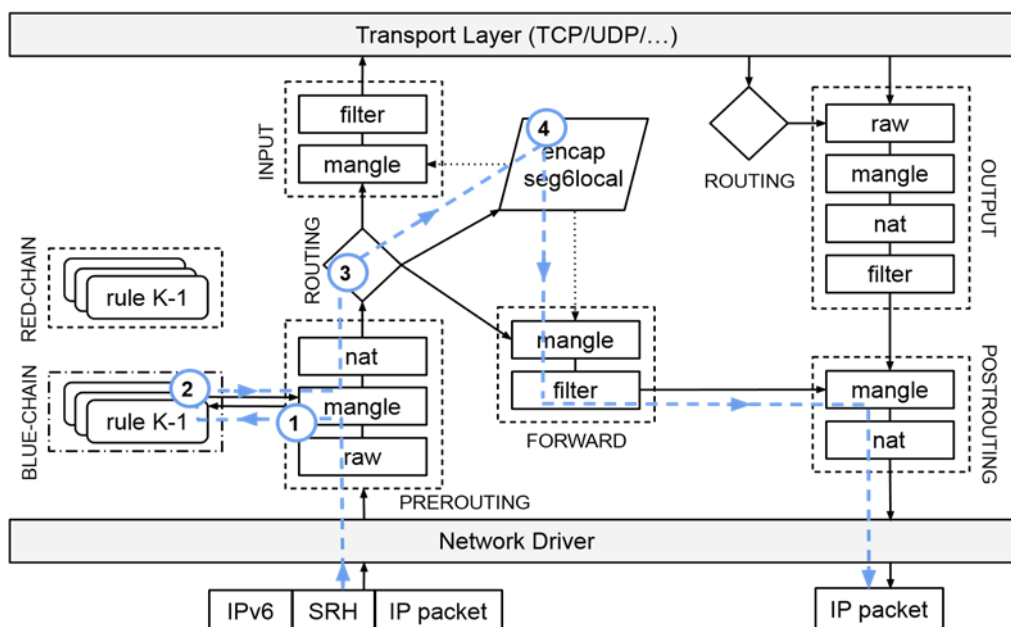


Figure 128: Packet processing in the Egress node

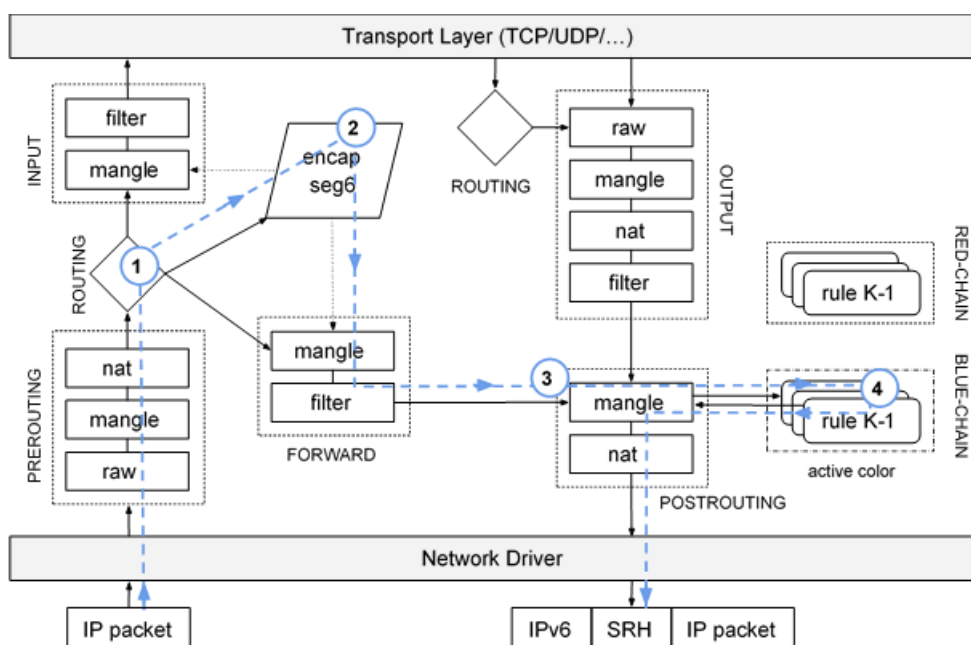


Figure 129: Packet processing in the Ingress node

### 7.4.4 Improvements to the SRH match extension

The Netfilter/Xtables already provides a module extension that matches packets based on the Segment Routing header of a packet. The module is named `ip6t_srh` and it has been included in the Linux kernel tree since version 4.16. With the `ip6t_srh` extension, it is possible to write Netfilter/Xtables rules that match the current SID, the next SID and the number of the left segments in a SRH. It is not possible to match packets by comparing a full SID list. Therefore, we extended `ip6t_srh` to support matching on the SID list.

The improvements to the `ip6t_srh` module are needed in the kernel-space and in the user-space sides. In the kernel-space side, we have allowed to store the SID list provided by user-space associated to a rule. Furthermore, we have also created a match function that compares this SID list with the one present in the examined SRv6 packet. When the two SID lists are identical, the match function returns successfully and we proceed with the execution of further match extension modules present in the rule and/or with the execution of a target action. If the two lists differ, the match function returns an error that allows Netfilter/Xtables to move onto the next rule, if any.

For what concerns the user-space side, changes have focused on `lib6t_srh.c`. This file is used to parse the commands provided by the user on the CLI and then to create, initialize and populate the data structures required to build a rule. The same data structures are used by `ip6t_srh` module to carry out matches on SRv6 packets. Therefore, we have added the parsing function that reads and validates a list of SIDs supplied by the user and the fields used to store the validated SID list. We have also implemented the helper functions to retrieve and display the SID list and the number of matched packets for each rule.

```

1) ip6tables -A POSTROUTING -t mangle \
   -m rt --rt- type 4 -j blue-chain

2) ip6tables -A blue-chain -t mangle \
   -m srh --srh-sid-list sid0,sid1 \
   -j TOS --set-tos 0x01/0x01
  
```

Figure 130: Iptables user space tool

The command (2) shown in Figure 130 adds a rule to the *blue-chain* of the mangle table and asks Netfilter/Xtables framework to use the *srh* module to match a SRv6 packet considering the SID list indicated by the attribute `--srh-sid-list`. In this example, the action (`-j --set-tos 0x01/0x01`) specified in the target consists in marking the packet by setting the first bit of the *DS* field of the outer IPv6 header.

### 7.4.5 Performance issues of *iptables* based PF-PLM

The rules that are included in the coloring chains (ingress node) and in the colorcounting chains (egress node) are tested sequentially, until a matching rule is found (or all the rules have been tested).

This sequential scan approach can have a significant impact on the node throughput due to the processing load for comparing the SID list in the SRH packets with the SID lists in the coloring chain or in the color-counting chain. On average, the processing load increases linearly with the number of rules. This number of rules corresponds to the number of flows for which we want to monitor the packet loss. In the performance experiments we will analyze the impact of the number of monitored flows on the node throughput due to the sequential scan approach.

### 7.4.6 The *IP set* framework

*IP set* [43] is an extension to Netfilter/Xtables/iptables that is available through Xtables-addons [44]. *IP set* allows to create rules that match entire *sets* of elements at once. Unlike normal *iptables* chains, which are stored and traversed linearly, elements in the sets are stored in indexed data structures, making lookups very efficient, even when dealing with large sets. Depending on the type, an IP set may store IP host addresses, IP network addresses, TCP/UDP port numbers, MAC addresses, interface names or combination of them in a way, which ensures lightning speed when matching an entry against a set. To use *IP set*, we create and populate uniquely

named sets using the *IP set* command-line and then we can reference those sets in the match specification of one or more *iptables* rules.

By using *IP set*, the *iptables* command refers the set with the match specification `-m set --set foo dst`, which means “match packets whose destination is contained in the set with the name *foo*”. As a result, a single *iptables* command is required regardless of the number of elements in the *foo* set. If we want to achieve the same result with the use of *iptables* only, it would be necessary to create a chain and insert as many rules as the elements contained in the *foo* set. *IP set* provides different types of data structures to store the elements (addresses, networks, etc). Each set type has its own rules for the type, range and distribution of values it can contain. Different set types also use different types of indexes and are optimized for different scenarios. The best/most efficient set type depends on the situation. The hash sets offer excellent performance in terms of speed of the execution time of lookup/match operation and they fit perfectly to our needs. Assuming to insert  $N$  IP addresses in the hash set *foo*, the cost of searching for an address is asymptotically equal to  $O(1)$ . Conversely, the same operation with *iptables* would have a cost of  $O(N)$ .

The *IP set* framework is designed to be extensible: there are several header files to be included that contain the helper functions and templates which, through an intelligent use of C macros and callbacks, allow to redefine and adapt the code to our needs.

### 7.4.7 The *IP set* based PF-PLM implementation

Using the *iptables* PF-PLM implementation described in Section 7.4.3, the packet processing throughput decreases when the number of flows to be monitored increases. To overcome this shortcoming, we have designed and implemented an enhanced solution based on *IP set*.

The implementation of *IP set* does not natively allow to store elements of SID list type within a hash set. Therefore, in order to use *IP set* for our purposes we have patched it by creating a new hash set called *sr6hash* so that we can insert the SID lists that we want to monitor.

To support the new *sr6hash* hash type, we patched the *IP set* framework on both the user-space and the kernel-space sides. In the user-space side, we defined a new data structure, the *nf\_srh*, which contains the SRH header with a SID list whose maximum length is fixed and set at compilation time (16 SIDs in our experiments). Moreover, two new functions have been added to the *IP set* framework libraries: the parse function (`ipset_parse_srh()`) and the print function (`ipset_print_srh()`). The `ipset_parse_srh()` is used to parse the SID list supplied by the user with the *IP set* CLI and to create and populate the *nf\_srh* data structures. The `ipset_print_srh()` is used to display all the SID lists in a given *sr6hash* set along with their match counters.

In Figure 131 we show the patched user-space commands used to: 1) create a hash set of type *sr6hash*; 2) add a SID list consisting of two SIDs; 3) add an *iptables* rule, in which the match is performed on the hash set *blue-ht* and the action consists in marking the first bit of the DS (tos) field in the outer IPv6 packet.

```

1) ipset -N blue-ht sr6hash counters

2) ipset -A blue-ht 2001:db8::1,2001::db8::2

3) ip6tables -A blue-chain -t mangle \ -m set --match-set blue-ht dst \
-j TOS --set-tos 0x01/0x01
  
```

**Figure 131: IPset userspace tool**

In the kernel part, we introduced a new *IP set* module which is the implementation of the hash set *sr6hash*. In particular, we have defined:

- the data structure of the element of the hash set (*hash\_sr6\_elem*) which contains the SRH with SID list to be stored. The user-space and kernelspace data structures are identical so that it facilitates the exchange of information between the two contexts;
- the equality function `hash_sr6_data_equal()` to compare two SID lists;
- the functions `hash_sr6_kadt()` and `hash_sr6_uadt()` which are used for adding, deleting an element to/from the hash set and testing the membership.

- the `ip_set_type` structure where we set the properties, policies and extensions supported by our module.

Therefore, this structure is used as a "glue" that sticks parts together and is used to, actually, register the hash set when the module is loaded into the kernel and to deregister it when it is unloaded.

### 7.4.8 The extended Berkeley Packet Filter (eBPF)

Proposed in the early '90s, the Berkeley Packet Filter (BPF) [45] is designed as a solution for performing packet filtering directly in the kernel of BSD systems. BPF comes along with its own set of RISC-based instructions used for writing packet filter applications and it provides a simple Virtual Machine (VM) which allows BPF filter programs to be executed in the data path of the networking stack. The bytecode of an BPF application is transferred from the userspace to the kernel space where it is checked for assuring security and preventing kernel crashes. BPF also define a packet-based memory model, few registers and a temporary auxiliary memory.

The Linux kernel has supported BPF since version 2.5 and the major changes over the years have been focused on implementing dynamic translation [46] between BPF and x86 instructions. Starting from release 3.18 of the Linux kernel, the Extended BPF (eBPF) [47] represents an enhancement over BPF which adds more resources such as new registers, enhanced load/store instructions and it improves both the processing and the memory models. eBPF programs can be written using assembly instructions later converted in bytecode or in restricted C and compiled using the LLVM Clang compiler. The bytecode can be loaded into the system through the `bpf()` syscall that forces the program to pass a set of sanity/safety-checks in order to verify if the program can be harmful for the system. Only safe eBPF programs can be loaded into the system, the ones considered unsafe are rejected. To be considered safe, a program must meet a number of constraints such as limited number of instructions, limited use of backward jumps, limited use of the stack and etc. These limitations [48] can impact on the ability to create powerful network programs.

eBPF programs are triggered by some internal and/or external events which span from the execution of a specific syscall up to the incoming of a network packet. Therefore, eBPF programs are hooked to different type of events and each one comes with a specific execution context. Depending on the context, there are programs that can legitimately perform socket filtering operations while other can perform only traffic classification at the TC layer and so on. eBPF programs are designed to be stateless so that every run is independent from the others. The eBPF infrastructure provides specific data structures, called *maps*, that can be accessed by the eBPF programs and by the userspace when they need to share some information.

### 7.4.9 The eBPF based counters implementation

The Loss Monitoring implementations presented in [49] were based on some Linux kernel modules which carried out all the needed operations for parsing packets, updating flow counters and coloring. Kernel modules allowed us to implement the counting system in a very effective way, but this approach comes with a number of disadvantages that we briefly report hereafter. Every time we need to update the Linux kernel, we also have to maintain the counting modules updated. Furthermore, if we want to add a functionality to our counting implementation, we have to remove the loaded modules and to replace them with the updated ones. Replacing a module is not an atomic operation: the time between the removal and the loading of the updated module causes a down-time of the monitoring service which may not be tolerated if we are accounting statistics on high-rate data flows. A kernel module can have unconditional access to most of the internals of the system and, if there is a bug, this can compromise the overall system security and stability. In addition, the *iptables* and *IP set* modules are hooked to the Netfilter/Xtables which can not guarantee the maximum performance in terms of packet processing.

An eBPF network program, despite its limitations, can solve the problems that arise by implementing the counter using kernel modules. Therefore, we designed two eBPF programs attached to specific networking hooks with the purpose of entirely replacing the previous implementations.

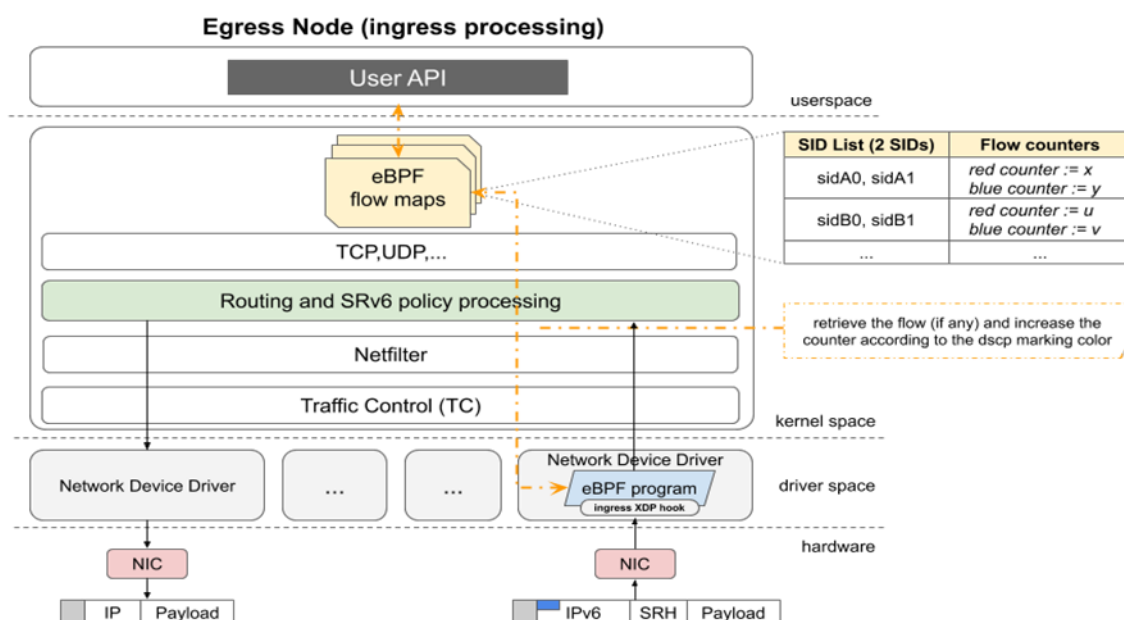
Both the `tc_srv6_pfplm_egress` and the `xdp_srv6_pfplm_ingress` eBPF programs are written in C and compiled through the LLVM Clang compiler. The result is a single object file called `srv6_pfplm_kern.o` where each program is placed in a specific text section so that it can be loaded independently from the other. The two eBPF programs are designed to account individually the ingress and egress flows that match some given SR policies taking into account also the color marking.

Flows lookup and match operations are fast and efficient thanks to the `BPF_MAP_TYPE_PERCPU_HASH` map type which help us to retrieve the flow counters using the SID list as key in a lock-free fashion and in constant



time, regardless of the number of policies present in the table. Currently, all the hashtable data structures provided by the eBPF infrastructure support fixed length keys. Thus, to maximize the lookup performance and minimize memory waste, we created  $N$  maps/hashtables and in each map we store all the flows with the same SID list length. The value of  $N$  can be decided at compile time and in our implementation the default max SID list length is equal to  $N = 16$  SIDs and therefore, we come up with a total of 16 maps/hashtables. Furthermore, we also have decided to store and keep separated the ingress flows to be monitored from the egress ones, hence the total number of flow maps/hashtables is doubled.

We provided another map that keeps track of the active color used by the egress node to mark the packets whose flows match an egress SR policy. The eBPF maps are created automatically when the `xdp_srv6_pfplm_ingress` and `tc_srv6_pfplm_egress` programs are loaded using respectively the `ip` and `tc` commands of the `iproute2` suite. All our maps are persistent and accessible as files through the `/sys/fs/bpf/pfplm` directory and for interacting with them we provided an User API (UAPI) released in the form of a shared library. Our UAPI hides the complexity of the underneath data structures and allows the user to change the active color for marking packets, to easily add/remove flows in ingress/egress directions and to retrieve flow statistics. The `xdp_srv6_pfplm_ingress` eBPF program is hooked on eXpress Data Path (XDP) which allows intercepting RX packets right out of the NIC driver and (possibly) before the socket buffer (skb) is allocated.



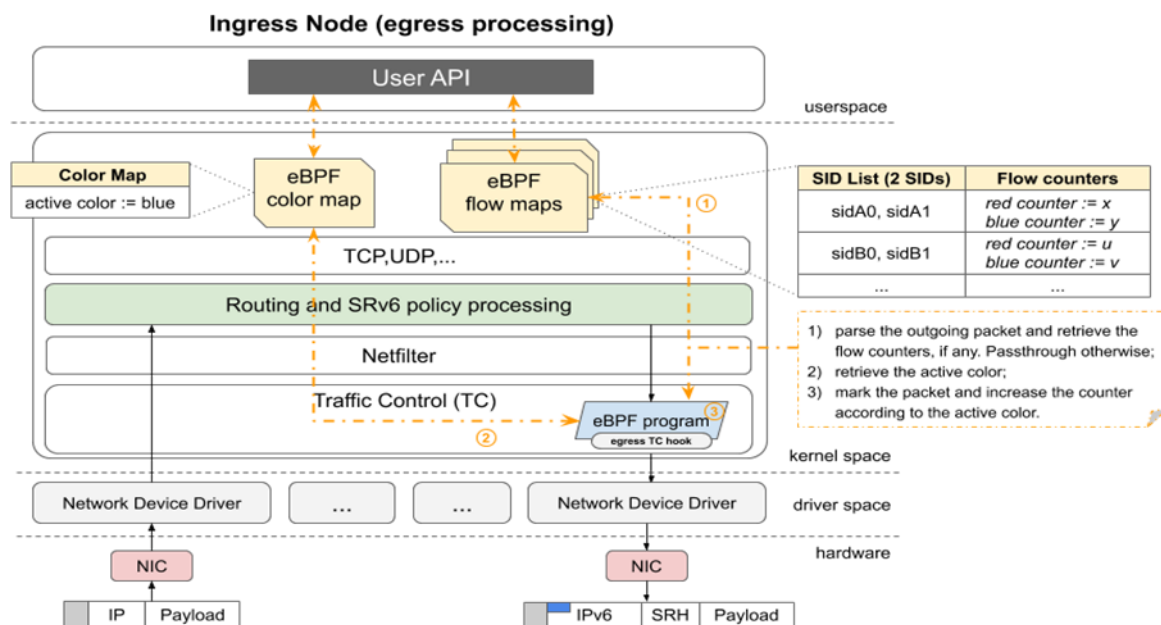
**Figure 132: Packet processing and statistics management in the Egress node using the PF-PLM eBPF implementation**

Figure 132 shows how the eBPF program processes the incoming packets and how it keeps track of per-flow statistics in the egress node. The `xdp_srv6_pfplm_ingress` eBPF network program parses the headers of every incoming packet on a specific network interface, it extracts the flow (if any) and it determines, by looking at the right flow map/hashtable, whether the flow has to be monitored or not. If the flow does not need to be monitored, no actions are taken and the packet continues to traverse the networking stack. Instead, if the extracted flow matches with one found in a flow map/hashtable, the correspondent counter is retrieved from the map and it is increased according to the marking color carried by the DSCP field of the outer IPv6 header.

The `tc_srv6_pfplm_egress` eBPF program leverages the hook offered by Linux Traffic Control (TC) for intercepting the TX packets just before they leave the node. We attached the eBPF program to the TC hook because, unlike XDP that intercepts only RX packets, it also allows dealing with TX packets.

Figure 133 shows how the eBPF program processes the outgoing packets and how it keeps track of per-flow statistics in the ingress node. The `tc_srv6_pfplm_egress` eBPF network program parses the headers of the outgoing packets on a given interface, it extracts the flow (if any) and it determines, by looking at the right flow map/hashtable, whether the flow has to be monitored or not. If the flow does not need to be monitored, no further action are taken by this eBPF program. Otherwise, if the extracted flow matches with one found in a

flow map/hashtable, the active color is retrieved from the color map, the packet is marked and the corresponding flow counter (obtained from the flow map/hashtable) is increased.



**Figure 133: Packet processing and statistics management in the Ingress node using the PF-PLM eBPF implementation.**

### 7.4.10 Data collection

To test the effectiveness of the proposed solution we developed a prototype implementation of the Sender and Reflector based on the python Scapy project [50] that support SRv6 packets. Both the Sender and the Reflector periodically change the active color. The Sender reads the local counter and generate the query packet that is sent using the SRv6 path. The Reflector receives the packet, reads the missing counters and sends the response packet back to the Sender, that eventually is able to evaluate the packet loss. An open-source implementation of the python code is available online [39].

## 7.5 Results

In order to evaluate the impact of the proposed solutions in a real Linux implementation, we have set up a testbed according to RFC 2544 [51], which provides the guidelines for benchmarking networking devices. We have executed two performance experiments. In the first performance experiment, we have compared the performance of the *iptables* implementation with the performance of the *IP set* based implementation. As expected, the performance of *IP set* is much better as it scales well with the number of monitored flows. In the second performance experiment, we have compared the performance of the *IP set* based implementation with the performance of the eBPF based implementation.

### 7.5.1 Testbed Description

Figure 134 depicts the testbed architecture, made of two nodes denoted as *Traffic Generator and Receiver (TGR)* and *System Under Test (SUT)*. In our experiments we only consider the traffic in one direction: the packets are generated by the TGR on the Sender port, enter the SUT from the IN port, exit the SUT from the OUT port and then they are received back by the TGR on the Receiver port. Thus, the *TGR* can evaluate all different kinds of statistics on the transmitted traffic including packet loss, delay, etc.

The testbed is deployed on the CloudLab facilities [52], a flexible infrastructure dedicated to scientific research on the future of Cloud Computing. Both the TGR and the SUT are bare metal servers with Intel Xeon E5-2630 v3 processor with 16 cores (hyper-threaded) clocked at 2.40GHz, 128 GB of RAM and two Intel 82599ES 10-Gigabit network interface cards. The SUT node runs a vanilla version of Linux kernel 5.4 and hosts the PF-PLM system. We consider two configurations to evaluate the SUT performance:

- the SUT is configured as the ingress node of the SRv6 network, i.e. it executes the encapsulation operation.
- the SUT is configured as the egress node of the SRv6 network, i.e. it executes the decapsulation operation.

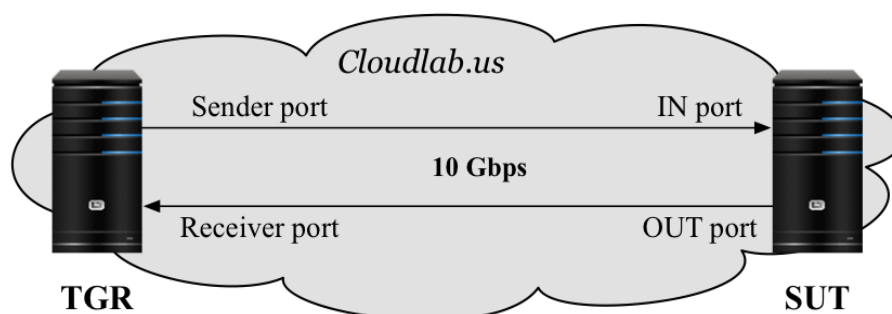


Figure 134: Testbed architecture

In the TGR node we exploit TRex [53] that is an open source traffic generator powered by DPDK [54]. We used SRPerf [55], a performance evaluation framework for software and hardware implementations of SRv6, which automatically controls the TRex generator in order to evaluate the maximum throughput that can be processed by the SUT. The maximum throughput is defined as the maximum packet rate at which the packet drop ratio is smaller than or equal to 0.5%. This is also referred to as Partial Drop Rate at a 0.5% drop ratio (in short PDR@0.5%). Further details on PDR and insights about nodes configurations for the correct execution of the experiments can be found in [55].

## 7.5.2 Comparison of Iptables vs. IP set based implementations

To carry out the performance experiments on the SUT, we crafted IPv6 UDP packets encapsulated in outer IPv6 packets (78 byte including all headers). The outer packets have an SRH with a SID list of one SID. We repeated each test four times (note that, as described in [55], each test includes a large number of experiments and repetitions to estimate the maximum throughput).

In Figure 135, we plot the throughput (in kpps) for the SUT configured as an ingress node considering different settings. The red line represents the SUT base performance in the simple SRv6 case, i.e. it applies SR policies but does not perform any counting or coloring operations. In this case the measured average maximum throughput reached is about 995 kpps.

Then we assessed the performance loss due to the counting operations for both the *iptables* and the *IP set* solutions, varying the number of monitored flows. The SUT performs the matching operation but does not modify the packet to color it. The experiments results are reported in Figure 135 using the black lines. As expected, using the *iptables* based PF-PLM the performance degrades increasing the number of monitored SID lists (i.e. the number of *iptables* rules). The measured throughput decreased in an inversely proportional way with the number of required *iptables* rules. Instead, the *IP set* version that uses the hashset allows to achieve a throughput that is almost constant with the number of monitored SID lists. We note from Figure 135 that when we need to monitor a single flow, the *iptables* based implementation achieves a higher SUT throughput with respect to the *IP set* based one. In particular, the throughput degradation compared to the base case is 8%, while the degradation for the *IP set* based PF-PLM is 15.5%. However when 16 flows are monitored the throughput of the *iptables* based PF-PLM decreases by 42%.

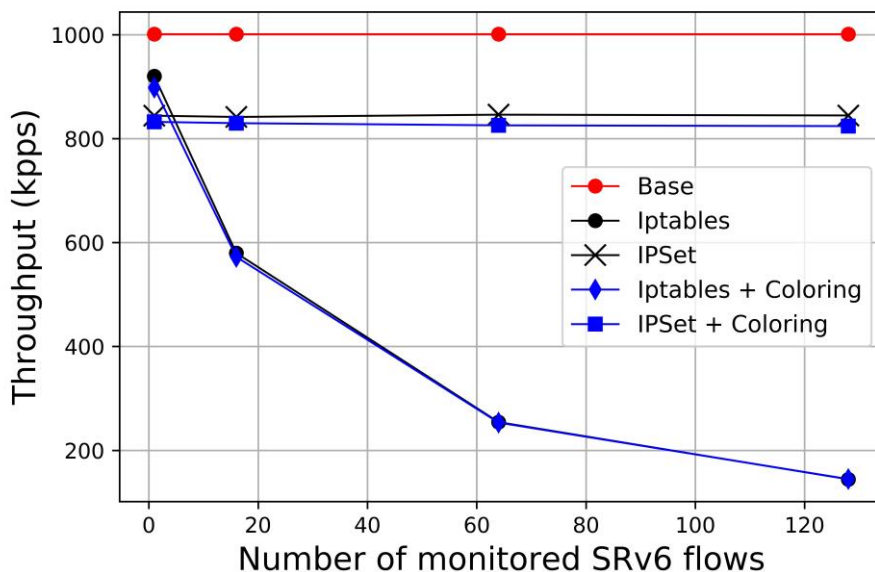


Figure 135: SUT throughput (ingress node configuration)

Finally, we evaluate the coloring cost of both solutions. In this case the degradation both in the *iptables* and in the *IP set* PF-PLM is less than 2.5%. In particular in the *iptables* case the maximum degradation is measured when a single rule is present and it reaches 10.3% with respect to the base case. The coloring loss in the *IP set* case is slightly less and the throughput degradation with respect to the base case is about 17.5%.

In Figure 136 we report the same analysis for the SUT configured as an egress node. We note that the decapsulation operation is more demanding and the total overall throughput in the base case is lower (about 940kpps). As for all the other configurations, the trend is similar to that of the ingress node with similar percentage degradation.

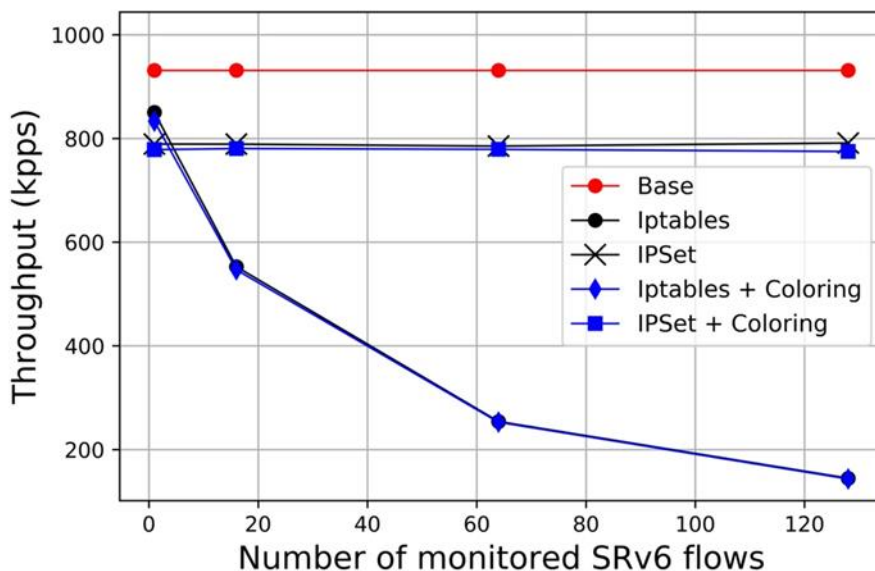


Figure 136: SUT throughput (egress node configuration)

### 7.5.3 Comparison of IP set vs. eBPF based implementations

For comparing the performance of the *IP set* with that of the eBPF, we carried out all our experiments on a new testbed instance which relies on the same hardware used in Subsection 7.5.2. We also used the same tools for generating traffic as well as for evaluating the achieved throughput considering the PDR@0.5%. The only noteworthy difference between the new testbed and the previous one concerns the Linux kernel version of the SUT

which has been upgraded from 5.4 to 5.6. We decided to update the Linux kernel to release 5.6 because, at the time of writing, it represents the stable and the latest release.

Moving to the Linux kernel 5.6 resulted in a positive effect on the overall performance of the networking stack. In fact, in the new experiments the base performance in the SRv6 encap and decap is respectively around 1100 kpps and 1000 kpps. Due to this fact, we carried out the experiments for *IP set* in the new instance of the testbed so that we can correctly and fairly compare the results with those obtained from the experiments on eBPF.

In Figure 137, we plot the throughput (in kpps) vs. the number of monitored flows for the SUT configured as an ingress node considering different settings. The eBPF program hooked to TC allows to achieve better performance in terms of the overall throughput compared to the performance obtained with Netfilter/*IP set*. The eBPF based PF-PLM starts with a 10.5% degradation (w.r.t. the base) when there are zero flows to be monitored. The overall performance degradation reaches the average value of 13.9% for one flow to be monitored and this value remains stable regardless of how many additional flows are added. The *IP set* based PF-PLM starts with a 13.8% performance degradation for zero flows to be monitored and it reaches the stable value of 22.0% for one or more monitored flows. This is due to the low processing overhead of the TC layer compared to the non-negligible one introduced by the Netfilter layer on which are hooked many running subsystems, i.e: *iptables* and all the related tables and chains.

In Figure 138 we report the same analysis for the SUT configured as an egress node. In this scenario, the performance achieved with the eBPF program are considerably better if compared to performance of the *IP Set* module. The eBPF based PF-PLM achieves a 4.2% of performance degradation (w.r.t. the base) in case there are no flows to be monitored. The performance drops at 5.4% on average if there is one flow to be monitored and it keeps stable no matter of how many flows (to be monitored) are subsequently added. The *IP set* based PF-PLM introduces a 15.0% of degradation for zero flows and it remains stable at 17.6% for one or more flows to be monitored. The reason lies in the fact that the eBPF program is directly hooked to the eXpress Data Path (XDP) and, in our case, the network card driver supports the XDP native-mode. Native mode executes eBPF/XDP program in the networking driver early RX path. At this stage, a network packet is nothing more than a mere sequence of bytes with no associated metadata. Raw data access is very efficient, as the operations

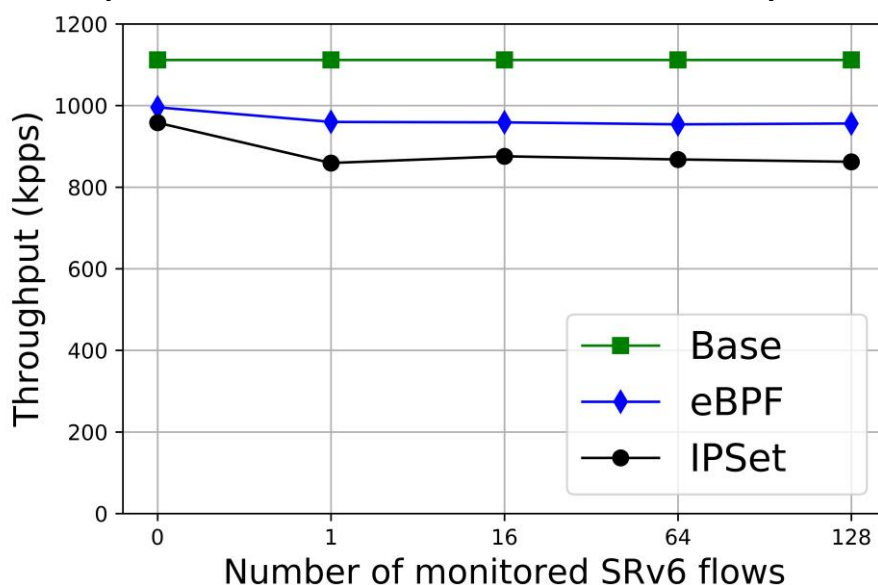
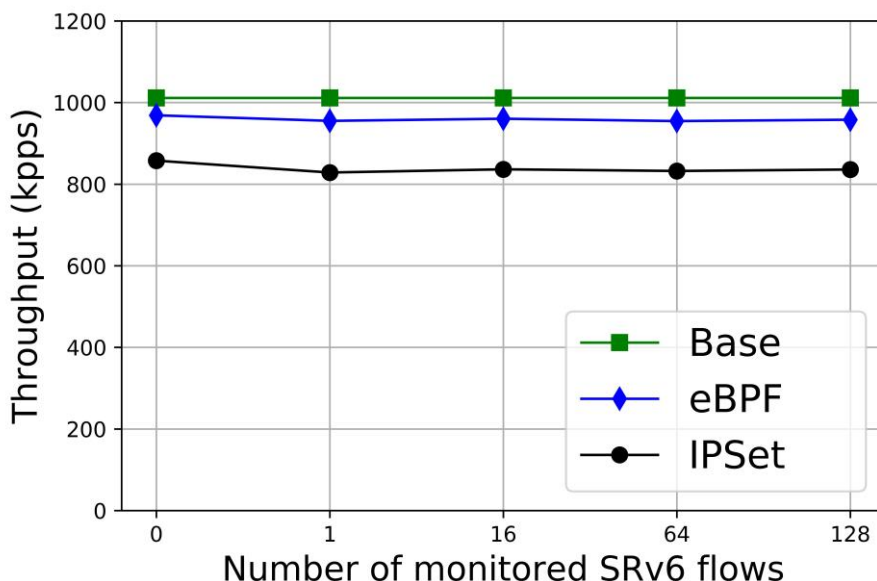


Figure 137: SUT throughput (ingress node configuration)



**Figure 138: SUT throughput (egress node configuration) needed for parsing the packet headers and for extracting the flows.**

## 7.6 Analysis of results

In this section we have first presented the status of the ongoing efforts for the standardization of the performance monitoring of SRv6 networks, focusing on loss monitoring. The proposed solutions consider the *alternate marking* method to achieve an accurate evaluation of packet loss (single packet loss granularity). The definition of the specific packet marking mechanism for loss monitoring in SRv6 networks is left open, so we have proposed a marking mechanism based on the DS (Differentiated Services) field in the IPv6 header.

We have provided an open source implementation of the proposed Per-Flow Packet Loss Monitoring (PF-PLM) solution in Linux. In particular, we have implemented three versions of (PF-PLM) in Linux, respectively based on plain *iptables*, on *IP set* and finally using eBPF with XDP and TC.

We have setup an evaluation testbed platforms, which allowed us to validate the functionality of the protocol and to evaluate some performance aspects. We were able to evaluate the cost of activating the monitoring (i.e. the degradation of the maximum forwarding throughput achievable in a node), for the three versions of the implementation. In particular, we measured the throughput degradation versus the number of monitored SRv6 flows. In a first instance of the testbed experiments, we compared the *iptables* and *IP set* based implementations. The *iptables* based PF-PLM starts with a 8% degradation for a single monitored flows but reaches 42% degradation with 16 flows and 80% degradation with 100 flows. The *IP set* based PF-PLM achieves a 15% degradation, irrespective of the number of monitored flow, hence it is scalable. In a second instance of the testbed experiments, we compared the *IP set* based implementation with the eBPF based one. The *IP set* based PF-PLM starts with a 13.8% and 15.0% performance degradation on the ingress and egress node for zero flows to be monitored. Instead, for one or more monitored flows the drop in performance reaches 22.0% on the ingress node and 17.6% on the egress node. The eBPF based PF-PLM allows us to achieve the best overall performance. When there are no flows to be monitored, the overhead introduced by the monitoring operations is approximately 10.5% and 4.2% respectively in the ingress and egress node. In case that there is at least one flow to be monitored, the drop in performance is stable around 13.9% in the ingress node and it is only 5.4% in the egress node.

Note that worse relative performance of the *IP set* based PF-PLM implementation with respect to the previous experiment depends on the fact that in the 5.6 release of the Linux kernel the performance of SRv6 encap and decap were improved.

Our ongoing work includes the improvements to the counting mechanism implementation to reduce the throughput degradation. Our target is to achieve an almost negligible cost for running the loss monitoring. We are also considering the implementation of other marking mechanisms for SRv6 and the implementation of delay monitoring in addition to loss monitoring.

## 8 Conclusions

This deliverable is divided in two parts. The first part presents the second version of the 5G EVE Portal, including an updated version of the REST API offered by the Portal backend to both the GUI elements developed by the project, and to other authorized projects by means of the public methods defined by the 5G EVE project. We also provide a new update of the service handbook, presenting a complete 5G EVE Portal user guide, which is of paramount importance for all the actors involved in 5G EVE overall testbed platform and for the successful deployment of any experiment.

The second part to the deliverable includes five contributions regarding the implementation and benchmarking of innovative data planes and radio access features. These include (i) the use of P4 abstraction model to share non-local states between the VNFs, (ii) the comparison of different approaches to support VNF, (iii) the use of Flexible Ethernet and of Preferred Path Routing, (iv) the fully cloud-native implementation of OpenAir-Interface RAN and Core, (v) the open source implementation of an efficient and accurate per-flow packet loss measurement (PF-PLM) solution based on the “alternate marking” method for SRv6 based network. All these contributions are disjoint and complementary and each of them tackles a new research approach to improve the performance of the data plane and meet the requirements for uRLLC applications.

## Acknowledgment

This project has received funding from the EU H2020 research and innovation programme under Grant Agreement No. 815074.



## Annex A – Updated example of requests received by the DCM from the ELM

This example is based on the metrics and KPIs from the VSB, CtxB and ExpB examples provided in D4.3 – Annex A, Annex E and Annex G, respectively, showing the updated transformations performed by the ELM for providing all the monitoring data to the DCM through its API, using the *publish* operation (*POST /dcm/publish/<signalling topic>*).

This case is based on the subscription phase, so the *action* field is set to “subscribe”. In case of being in the withdrawal phase, the *action* field would be set to “unsubscribe”. Both phases are defined in deliverable D3.3.

The updates included in this deliverable, comparing to the version from D4.3, are the following:

- The topic name now includes the use case identification at the beginning, before the experimentId, so that the topic name is now composed by five string chains separated by '.', instead of four. This change is also reported in D3.4, where the Topic framework proposal is updated with the new changes and updates.
- A new parameter is included within the *context* field: the *graph* type, a String that identifies the type of graph (line, pie, etc.) that Kibana must generate for that metric or KPI.
- Regarding the *context* field, note that not all the information provided in blueprints for defining the metrics and KPIs is included in the messages exchanged between the ELM and the DCM. In the case of metrics (both application and infrastructure), it is only necessary to be received the *metricId*, *name*, *metricCollectionType*, *unit*, *interval* and *graph* fields. For KPIs, it is only needed the *kpiId*, *name*, *unit*, *interval* and *graph* fields.
- There was an error in the last KPI: it is *kpi\_memory\_usage* instead of *kpi\_tracking\_response\_time* (repeated twice in the previous version).
- Note that the original application metrics defined for this example contained the metrics defined in the CtxB, but only for the case of the traffic delay generator, and not for the background traffic generator. In this updated version, the application metrics from the background traffic generator CtxB are included.

```
POST /dcm/publish/signalling.metric.application

{
  'records': [{
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.application_metric.tracked_devices',
      'context': {
        'metricId': 'tracked_devices',
        'name': 'Number of total tracked devices',
        'metricCollectionType': 'CUMULATIVE',
        'unit': 'devices',
        'interval': '5s',
        'graph': 'LINE'
      }
    }
  },
  {
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.application_metric.processed_records',
      'context': {
        'metricId': 'processed records',
        'name': 'Number of total processed records',
        'metricCollectionType': 'CUMULATIVE',
        'unit': 'records',
        'interval': '5s',
        'topic': '/app/processed records'
      }
    }
  }
],
}
```

```

{
  'value': {
    'expId': '1',
    'action': 'subscribe',
    'topic': 'a2t.1.italy_turin.application_metric.tracking_response_time',
    'context': {
      'metricId': 'tracking_response_time',
      'name': 'Max response time in milliseconds',
      'metricCollectionType': 'GAUGE',
      'unit': 'ms',
      'interval': '1s',
      'graph': 'PIE'
    }
  }
},
{
  'value': {
    'expId': '1',
    'action': 'subscribe',
    'topic': 'a2t.1.italy_turin.application_metric.tracking_memory_usage',
    'context': {
      'metricId': 'tracking_memory_usage',
      'name': 'Memory usage in %',
      'metricCollectionType': 'GAUGE',
      'unit': '%',
      'interval': '1s',
      'graph': 'PIE'
    }
  }
},
{
  'value': {
    'expId': '1',
    'action': 'subscribe',
    'topic': 'a2t.1.italy_turin.application_metric.delay_iface',
    'context': {
      'metricId': 'delay_iface',
      'name': 'Traffic delay measured between In and Out interfaces',
      'metricCollectionType': 'GAUGE',
      'unit': 'ms',
      'interval': '1s',
      'graph': 'PIE'
    }
  }
},
{
  'value': {
    'expId': '1',
    'action': 'subscribe',
    'topic': 'a2t.1.italy_turin.application_metric.traffic_rate_dst',
    'context': {
      'metricId': 'traffic_rate_dst',
      'name': 'Traffic rate measured at DST',
      'metricCollectionType': 'GAUGE',
      'unit': 'Mbps',
      'interval': '2s',
      'graph': 'PIE'
    }
  }
},
{
  'value': {
    'expId': '1',
    'action': 'subscribe',
    'topic': 'a2t.1.italy_turin.application_metric.lost_packets',
    'context': {
      'metricId': 'lost_packets',
      'name': 'Lost packets between SRC and DST',
      'metricCollectionType': 'CUMULATIVE',
      'unit': 'count',
      'interval': '5s',
      'graph': 'LINE'
    }
  }
}
}

```

```
POST /dcm/publish/signalling.metric.infrastructure

{
  'records': [{
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.infrastructure_metric.cpu_consumption',
      'context': {
        'metricId': 'cpu_consumption',
        'name': 'Measurement of CPU consumption.',
        'metricCollectionType': 'GAUGE',
        'unit': '%',
        'interval': '1s',
        'graph': 'PIE'
      }
    }
  },
  {
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.infrastructure_metric.memory_consumption',
      'context': {
        'metricId': 'memory_consumption',
        'name': 'Measurement of memory consumption.',
        'metricCollectionType': 'GAUGE',
        'unit': 'GB',
        'interval': '1s',
        'graph': 'PIE'
      }
    }
  }
]}

POST /dcm/publish/signalling.kpi

{
  'records': [{
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.kpi.kpi_tracking_response_time',
      'context': {
        'kpiId': 'kpi_tracking_response_time',
        'name': 'Response time must be lower than or equal to the objective.',
        'unit': 'ms',
        'interval': '1s',
        'graph': 'LINE'
      }
    }
  },
  {
    'value': {
      'expId': '1',
      'action': 'subscribe',
      'topic': 'a2t.1.italy_turin.kpi.kpi_memory_usage',
      'context': {
        'kpiId': 'kpi_memory_usage',
        'name': 'Memory usage must be lower than or equal to the objective.',
        'unit': 'MB',
        'interval': '1s',
        'graph': 'LINE'
      }
    }
  }
]}
}
```

## References

- [1] 5G EVE Project, “Deliverable D4.1: Experimentation tools and VNF repository”, October 2019.
- [2] 5G EVE Project, “Deliverable D4.2: First version of the experimental portal and service handbook. December 2019.
- [3] 5G EVE Project, “Deliverable D4.3: Models for vertical descriptor adaptation”. April 2020.
- [4] 5G EVE Project, “Deliverable D3.4: Second implementation of the interworking reference model”. June 2020.
- [5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. T. nd Amin Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-independent Packet Processors”, ACM SIGCOMM, 2014.
- [6] “State sharing with P4.”, <https://github.com/imanlotfimahyari/State-Sharing-p4python>
- [7] A. S. Muqaddas, G. Sviridov, P. Giaccone, and A. Bianco, “Optimal state replication in stateful data planes,” *IEEE Journal on Selected Areas in Communications*, 2020.
- [8] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi, “LODGE: Local Decisions on Global statEs in programmable data planes,” *arXiv preprint arXiv:2001.07670*, 2020, <http://arxiv.org/abs/2001.07670>
- [9] “pub sub.p4.”, [https://github.com/imanlotfimahyari/State-Sharing-p4python/blob/master/pub-sub/pubsub/register/pub\\_sub.p4](https://github.com/imanlotfimahyari/State-Sharing-p4python/blob/master/pub-sub/pubsub/register/pub_sub.p4)
- [10] “MININET An instant virtual network on your laptop (or other PC),” <http://mininet.org/>.
- [11] “OVSK Open Virtual Switch,” <https://www.openvswitch.org/>.
- [12] “BMV2 Behavioral Model Version 2,” <https://github.com/p4lang/behavioral-model>.
- [13] “Build SDN agilely.”, <https://osrg.github.io/ryu/>
- [14] M. Condoluci and T. Mahmoodi, “Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges”, *Computer Networks*, vol. 146, pp. 65–84, Dec. 2018
- [15] 3GPP, “NG-RAN: Architecture description,” TS 38.401, v15.7.0, Mar.2020
- [16] 3GPP, “System architecture for the 5G System,” TS 23.501, Mar. 2019.
- [17] T. Taleb et al., “On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration”, *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, Sep. 2017.
- [18] D. Bega, A. Banchs, M. Gramaglia, X. Perez, and P. Rost, “CARES: Computation-Aware Scheduling in Virtualized Radio Access Networks”, *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 7993–8006, Dec. 2018
- [19] I. Gomez-Miguel, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, “SrsLTE: An Open-Source Platform for LTE Evolution and Experimentation,” in *ACM WiNTECH 2016*.
- [20] J. A. Ayala-Romero et al., “Demo: Vrain proof-of-concept – a deeplearning approach for virtualized ran resource control,” *ACM MobiCom 2019*
- [21] C. Marquez et al., “How Should I Slice My Network? A Multi-Service Empirical Evaluation of Resource Sharing Efficiency,” *ACM MobiCom 2018*
- [22] U. Chunduri, A. Clemm, R. Li, “Preferred Path Routing - A Next-Generation Routing Framework beyond Segment Routing”, *IEEE Globecom 2018*

- [23] S. Previdi et al., “Segment Routing Architecture,” IETF RFC 8402, Jul. 2018. [Online]. Available: <https://tools.ietf.org/html/rfc8402/>
- [24] C. Filsfils et al., “The Segment Routing Architecture,” Global Communications Conference (GLOBECOM), 2015 IEEE, pp. 1–6, 2015.
- [25] C. Filsfils (ed.) et al., “IPv6 Segment Routing Header (SRH),” Internet Engineering Task Force, Internet-Draft draft-ietf-6man-segmentrouting-header, Oct. 2019, work in Progress. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header>
- [26] C. Filsfils et al., “SRv6 Network Programming,” Internet Engineering Task Force, Internet-Draft draft-ietf-spring-srv6-network-programming, Jan. 2020, work in Progress. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-spring-srv6-network-programming>
- [27] P. Ventre et al., “Segment routing: A comprehensive survey of research activities, standardization efforts and implementation results,” arXiv preprint arXiv:1904.03471, 2019.
- [28] S. Matsushima et al., “SRv6 Implementation and Deployment Status,” Internet Engineering Task Force, InternetDraft draft-matsushima-spring-srv6-deployment-status, May 2020, work in Progress. [Online]. Available: <https://tools.ietf.org/html/draft-matsushima-spring-srv6-deployment-status>
- [29] G. Fioccola, Ed., “Alternate-Marking Method for Passive and Hybrid Performance Monitoring ,” IETF RFC 8321, Jan. 2018. [Online]. Available: <https://tools.ietf.org/html/rfc8321>
- [30] S. Shalunov, B. Teitelbaum, et. al, “A One-way Active Measurement Protocol (OWAMP),” IETF RFC 4656, Sep. 2006. [Online]. Available: <https://tools.ietf.org/html/rfc4656>
- [31] K. Hedayat, R. Krzanowski, et al., “A Two-Way Active Measurement Protocol (TWAMP),” IETF RFC 5357, Sep. 2006. [Online]. Available: <https://tools.ietf.org/html/rfc5357>
- [32] D. Frost and S. Bryant, “Packet Loss and Delay Measurement for MPLS Networks,” IETF RFC 6374, Sep. 2011. [Online]. Available: <https://tools.ietf.org/html/rfc6374>
- [33] D. Mills, J. Martin, “Network Time Protocol Version 4: Protocol and Algorithms Specification,” IETF RFC 5905, Jun. 2010. [Online]. Available: <https://tools.ietf.org/html/rfc5905>
- [34] “1588-2008 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,” IEEE, Mar. 2008.
- [35] S. Bryant, S. Sivabalan and S. Soni, “UDP Return Path for Packet Loss and Delay Measurement for MPLS Networks,” IETF RFC 7876, Jul. 2016. [Online]. Available: <https://tools.ietf.org/html/rfc7876>
- [36] R. Gandhi, C. Filsfils, D. Voyer, M. Chen, and B. Janssens, “Performance Measurement Using TWAMP Light for Segment Routing Networks,” Internet Engineering Task Force, Internet-Draft draft-gandhi-springtwamp-srpm-05, Dec. 2019, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-gandhi-spring-twamp-srpm-05>
- [37] R. Gandhi, C. Filsfils, D. Voyer, S. Salsano, and M. Chen, “Performance Measurement Using UDP Path for Segment Routing Networks,” Internet Engineering Task Force, Internet-Draft draft-gandhi-spring-rfc6374-srpmudp-03, Nov. 2019, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-gandhi-spring-rfc6374-srpm-udp-03>
- [38] G. Mirsky, G. Jun, H. Nydell, and R. Foote, “Simple Two-way Active Measurement Protocol,” Internet Engineering Task Force, Internet-Draft draft-ietf-ippm-stamp-10, Oct. 2019, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-ietf-ippm-stamp-10>

- [39] “SRv6-pm - SRv6 Performance Monitoring in Linux,” 2020. [Online]. Available: <https://net-group.github.io/srv6-pm/>
- [40] D. Lebrun and O. Bonaventure, “Implementing IPv6 Segment Routing in the Linux Kernel,” in Proceedings of the Applied Networking Research Workshop. ACM, 2017, pp. 35–41.
- [41] F. Cladet et al., “Service Programming with Segment Routing,” Internet Engineering Task Force, Internet-Draft draft-xuclad-spring-sr-service-programming, Oct. 2018, work in Progress. [Online]. Available: <https://tools.ietf.org/html/draft-xuclad-spring-sr-service-programming>
- [42] A. Abdelsalam et al., “SRPerf: a Performance Evaluation Framework for IPv6 Segment Routing,” arXiv preprint arXiv:2001.06182, 2020.
- [43] “IPset.” [Online]. Available: <http://ipset.netfilter.org>
- [44] “Xtables-addons - additional extension to Xtables packet filter.” [Online]. Available: <http://xtables-addons.sourceforge.net>
- [45] S. McCanne and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture,” in USENIX winter, vol. 46, 1993.
- [46] “A JIT for packet filters.” [Online]. Available: <https://lwn.net/Articles/437981/>
- [47] “Linux Socket Filtering aka Berkeley Packet Filter (BPF).” [Online]. Available: <https://www.kernel.org/doc/Documentation/networking/filter.txt>
- [48] S. M. et al., “Creating Complex Network Services with eBPF: Experience and Lessons Learned,” in IEEE International Conference on High Performance Switching and Routing (HPSR2018). IEEE, 2018.
- [49] P. Loreti, A. Mayer, P. Lungaroni, S. Salsano, R. Gandhi, and C. Filsfils, “Implementation of accurate per-flow packet loss monitoring in segment routing over ipv6 networks,” in 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR). IEEE, 2020, pp.
- [50] “Scapy - Packet crafting for Python2 and Python3.” [Online]. Available: <https://scapy.net/>
- [51] S. Bradner and J. McQuaid, “Benchmarking Methodology for Network Interconnect Devices,” Internet Requests for Comments, RFC Editor, RFC 2544, March 1999. [Online]. Available: <https://tools.ietf.org/html/rfc2544>
- [52] Robert Ricci, Eric Eide, and the CloudLab Team, “Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications,” ; login:: the magazine of USENIX & SAGE, vol. 39, no. 6, pp. 36–38, 2014.
- [53] “TRex realistic traffic generator.” [Online]. Available: <https://trex-tgn.cisco.com/>
- [54] “DPDK.” [Online]. Available: <https://www.dpdk.org/>
- [55] A. Abdelsalam et al., “Performance of IPv6 Segment Routing in Linux Kernel,” in 1st Workshop on Segment Routing and Service Function Chaining (SR+SFC 2018) at CNSM 2018, Rome, Italy, 2018.
- [56] Small Cell Forum, 5G FAPI PHY API Specification 222.10.01, 2019.
- [57] Small-Cell Forum, nFAPI and FAPI Specifications, SCF082.09.05, 2017.
- [58] Zdarsky, F., Knopp, R., “Build Your Own Private 5G Network on Kubernetes”. Retrieved from Kubecon + CloudNativeCon Americas 2019: <https://kccncna19.sched.com/event/UaWZ/build-your-own-private-5g-network-on-kubernetes-frank-zdarsky-red-hat-raymond-knopp-eurecom>, Nov.20 2019.
- [59] Charts. Retrieved from helm.sh: <https://helm.sh/docs/topics/charts/>

- [60] Multicloud-k8s Plugin-Service API. Retrieved from wiki.onap.org: <https://wiki.onap.org/display/DW/MultiCloud+K8s-Plugin-service+API>
- [61] Kapadia, A. Integrating ONAP with a 5G Cloud Native Network. Retrieved from Linux Foundation Networking: <https://www.lfnetworking.org/resources/lfn-webinar-series/>, May 19, 2020.
- [62] openairCN. Retrieved from <https://github.com/OPENAIRINTERFACE/openair-cn>.
- [63] openairinterface-k8s. Retrieved from <https://github.com/OPENAIRINTERFACE/openairinterface-k8s>.
- [64] OPNFV VCO3, Kubecon 5G Cloud-Native Demonstration. Retrieved from <https://youtu.be/IL4nxb-mUIX8>, Nov. 21, 2019.