H2020 5G Dive Project
Grant No. 859881

D3.3 KPI and Performance Evaluation of 5G-DIVE Platform in Vertical Field Trials

Abstract

This deliverable of D3.3 is the third and final deliverable of WP3. The goal of this deliverable is to provide the final validation results of different 5G-DIVE use cases defined in D1.1, i.e., Digital Twin (DT), Zero Defect Manufacturing (ZDM) and Massive Machine-Type-of-Communication (mMTC) use cases for Industry 4.0 (I4.0) trial, and drone fleet navigation (ADSUC1) and Intelligent Image Processing for Drones (ADSUC2) use cases for Autonomous Drone Scouting (ADS) trial. Experiments of each use case have been performed for implementation validation including scalability analysis and system reliability. And the experimental results are evaluated considering the 5G-DIVE platform, 5G connectivity and edge and fog computing. Furthermore, the updated information of I4.0 trial site in 5TONIC and I4.0 use case integration are also provided in this deliverable.

Document properties

Document number	D3.3
Document title	KPI and performance evaluation of 5G-DIVE platform in vertical field trials
Document responsible	ITRI
Document editor	Samer Talat (ITRI)
Editorial team	NCTU: Muhammad Febrian Ardiansyah, Timothy William ADLINK: Gabriele Baldoni, Ivan Paez ITRI: Andee Lin, Samer Talat UC3M: Laura Caruso, Milan Groshev EAB: Chenguang Lu, Gyanesh Patra ULUND: Chao Zhang RISE: Saptarshi Hazra, Luca Mottola TELCA: Javier Sacido, Matteo Pergolesi, Aitor Zabala IDCC: Ibrahim Hemadeh, Filipe Conceicao
Target dissemination level	Public
Status of the document	Final
Version	1.0

Production properties

Reviewers	Antonio De la Oliva, Bengt Ahlgren, Ibrahim Hemadeh, Filipe Conceicao, Ivan Paez, Milan Groshev, Muhammad Febrian Ardiansyah , Timothy William, Matteo Pergolesi, Samer Talat, Andee Lin, Chenguang Lu, Shally Lu.

Document history

Revision	Date	Issued by	Description
1.0	2021-12-28	Samer Talat	Final version

Disclaimer

Contents

List of Tables 5

List of Figures 6

List of Acronyms 10

Executive Summary 14

1. Introduction 15

2. 5G Solution Validation 17

2.1. 5G SA solution for I4.0 trials in 5TONIC 17

2.2. 5G-NSA solution for ADS trials in MIRC 21

3. DEEP Platform Validation 24

3.1. BASS Validation results 24

3.1.1. Multi-region orchestration feature validation 24

3.1.2. Service management feature validation 25

3.1.3. Active Monitoring feature validation 26

3.1.4. SLA Enforcement feature validation 30

3.1.5. Scalability measurements 31

3.1.6. Availability and Reliability 35

3.2. DASS Validation results 36

3.2.1. Scalability measurements 36

3.2.2. Performance measurements 37

3.3. IESS Validation results 41

3.3.1. Model training feature validation 41

3.3.2. Inference app packaging and deployment feature validation 41

3.3.3. IESS Catalogue feature validation 42

3.3.4. Offer AutoML service for easy AI adoption 42

3.3.5. Scalability 43

3.3.6. Availability and Reliability 45

4. Final Integration 47

4.1. Trial setup premises and hardware resources 47

4.2. Configuration for DEEP platform and use-cases 50

5. Final validation results of I4.0 use cases 52

5.1. I4.0-UC1: Digital Twin 52

5.1.1. DEEP Integration Validation 52
5.1.2. End-to-end profiling 57
5.1.3. Experimental Results 60
5.1.4. Scalability 67
5.2. I4.0-UC2: ZDM 72
5.2.1. DEEP Integration Validation 72
5.2.2. Experimental Results 73
5.3. I4.0-UC3: massive MTC 83
5.3.1. DEEP Integration Validation 85
5.3.2. Experimental Results 88
5.3.3. Scalability 100
6. Final validation results of ADS use cases 105
6.1. Experimental setup 105
6.2. ADS-UC1: Drone Fleet Navigation 106
6.2.1. Mission Scenario and Flow 108
6.2.2. Experimental Results 109
6.2.3. Scalability 113
6.3. ADS-UC2: Intelligent Image Processing for Drones 116
6.3.1. Mission Scenario and Flow 116
6.3.2. Experimental Results 118
6.3.3. DEEP Integration Validation 122
6.3.4. Scalability 124
7. Conclusions 126
8. Annex A- VSD in YAML format for mMTC 128
9. References 129

List of Tables

Table 2-1 Throughput measurement results 20
Table 2-2 Various SpeedTest measurement results 23
Table 3-1 Region configured in 5Tonic trials 24
Table 3-2 CPU usage results for 10 sequential deployments on the BASS 31
Table 3-3 CPU usage results for parallel deployments on the BASS 32
Table 3-4 Deployment time results for 10 sequential deployments on the BASS 33
Table 3-5 Deployment time results for parallel deployments on the BASS 34
Table 3-6 Mesh routing at scale test results 36
Table 3-7 CPU usage results for multiple requests on the IESS 44
Table 3-8 Elapsed time for multiple requests arriving at the IESS 45
Table 4-1 List of servers available in 5TONIC trial site used for the deployment 49
Table 4-2: Resource specification for the trial setup 51
Table 5-1 Digital Twin modules, deployment location and BASS orchestration Driver 53
Table 5-2 Ping results 58
Table 5-3 Inference times of the movement prediction module in the Raspberry pi 62
Table 5-4 Hardware performance comparison 63
Table 5-5 Network and Application layer measurement 75
Table 5-6 Yolo engine telemetry collection example data point 78
Table 5-7 Parameters for different message patterns 94
Table 5-8 Impact of collection frequency on application performance 94
Table 5-9 Comparison results for the Pod failure without redundancy scenario 97
Table 5-10 Measurement results for POD redundancy scenario 98
Table 5-11 Measurement results for node redundancy scenario 99
Table 6-1 DCAS RTT latency measurements with 7 meter spacing 110
Table 6-2 DCAS RTT latency measurements with 15 meter spacing 110
Table 6-3: EagleEYE key modules processing latency 119
Table 6-4 EagleStitch key modules processing latency 120
Table 6-5 Drone data processor system performance metrics 121
Table 7-1: Final achievement per use case 126

List of Figures

Figure 2-1 5G SA trial system installed in 5TONIC........ 17
Figure 2-2 Two types of UEs used 17
Figure 2-3 5G SA trial system design........ 18
Figure 2-4 Network Topology of the 5G SA trial system 18
Figure 2-5 Screenshot of the SpeedTest measurement........ 19
Figure 2-6 Measurement locations at 5TONIC........ 20
Figure 2-7 The overall 5G NSA structure........ 21
Figure 2-8 Internal structure of 5G NSA EPC 22
Figure 2-9 Internal structure of 5G NSA iMEC........ 22
Figure 2-10 5G NSA NSA locations of measurement........ 23
Figure 3-1 BASS Web view of the regions for the final trials........ 25
Figure 3-2 BASS web view of multiple vertical services 26
Figure 3-3 INFLUXDB metrics visualization for mMTC use case........ 27
Figure 3-4 Metrics visualization from the BASS' web user interface........ 28
Figure 3-5 Logs visualization from the BASS’ user interface........ 29
Figure 3-6 Example of SLA definition in a VSD........ 30
Figure 3-7 Trend visualization of the maximum CPU usage........ 32
Figure 3-8 Trend visualization of the maximum CPU usage for parallel deployments........ 33
Figure 3-9 Deployment times for parallel deployments........ 34
Figure 3-10 Uptime monitor for the BASS........ 35
Figure 3-11 Peer-to-peer communication scenario........ 37
Figure 3-12 a) Eclipse Zenoh API messages/seconds in log scale, b) Eclipse Zenoh API bits/seconds in log scale........ 38
Figure 3-13 Eclipse Zenoh API latency on P2P communication........ 38
Figure 3-14 Brokered communication scenario........ 39
Figure 3-15 a) Eclipse Zenoh API messages/seconds in log scale, b) Eclipse Zenoh API bits/seconds in log scale........ 40
Figure 3-16 Eclipse Zenoh API latency on brokered communication 40
Figure 3-17 IESS catalogue view in BASS User Interface 42
Figure 3-18 Component descriptor for AutoML with IESS........ 43

Figure 3-19 Trend visualization for the maximum CPU usage 44
Figure 3-20 Elapsed time for multiple requests arriving at the IESS 45
Figure 3-21 Uptime monitor for the IESS 46
Figure 4-1 Plan of the 5TONIC Trial site 48
Figure 4-2 Trial setup at 5TONIC premises 50
Figure 5-1 Robot drivers section in the Digital Twin VSD 53
Figure 5-2 BASS Web-view after successful deployment and instantiation 54
Figure 5-3 Movement prediction section in the VSD 54
Figure 5-4 BASS Dataset catalogue web-view 55
Figure 5-5 BASS Web-view during IESS training 56
Figure 5-6 BASS web-view upon successful deployment and instantiation with the movement prediction 56
Figure 5-7 TCP downlink throughput 57
Figure 5-8 TCP uplink throughput 58
Figure 5-9 Downlink latency 59
Figure 5-10 Uplink latency 59
Figure 5-11 Comparison of movement prediction algorithms (training set). 61
Figure 5-12 Movement predictions for different actions and different algorithms (5 movements). 61
Figure 5-13 Movement predictions for different actions and different algorithms (50 movements). ... 61
Figure 5-14 Performance of movement prediction when applied to the physical robot 62
Figure 5-15 Average cumulative reward during training 64
Figure 5-16 Virtual environment for the training of the algorithm representing the robot workspace 65
Figure 5-17 Overall workflow of the SLA enforcer 66
Figure 5-18 Learning behaviour of SLA enforcer algorithm for precision tasks (500 ms) 67
Figure 5-19 Learning behaviour of SLA enforcer algorithm for screening tasks (1000ms) 67
Figure 5-20 CPU and RAM usage for the Digital twin modules when deployed on the robot 68
Figure 5-21: Testbed setup 69
Figure 5-22: Edge node overall CPU usage, Digital Twin control loop, update and command time and command loss rate 71
Figure 5-23. ZDM use case VSD file 73
Figure 5-24. Application-level measurement software 75

Figure 5-25 ZDM video stream initial throughput assessment 76
Figure 5-26 Defective object on displayed on the edge screen 77
Figure 5-27 Training and predicted object detection confidence values 79
Figure 5-28 future available wi-fi throughput 80
Figure 5-29 FUTURE AVAILABLE 5G THROUGHPUT 81
Figure 5-30 Example output of ATSSS decision making 82
Figure 5-31 achieved ATSSS xApp throughput for different splitting rules 82
Figure 5-32 5TONIC trial setup of massive MTC use case 84
Figure 5-33 mMTC trial setup deployment in 5TONIC 85
Figure 5-34 BASS VSD for the mMTC use case 86
Figure 5-35 BASS web view of mMTC use case with details on its components 87
Figure 5-36 mMTC Kubernetes cluster deployed using BASS 87
Figure 5-37 Influxdb monitoring dashboard for mMTC use case 88
Figure 5-38 Implementation of Experimental Setup for experiment A 89
Figure 5-39 CDF of RTT for asynchronous requests 90
Figure 5-40: Result of application performance per node for asynchronous requests 91
Figure 5-41 CDF of RTT for scheduled requests 92
Figure 5-42 Application performance of different nodes with scheduled requests 92
Figure 5-43 CDF of RTT with 5G connectivity 93
Figure 5-44 Application performance per node with 5G connectivity 93
Figure 5-45 ROC curve for RF Fingerprinting 96
Figure 5-46 Results of application performance per node for pod failure scenario 98
Figure 5-47 Results of application performance per node for POD redundancy scenario 99
Figure 5-48 Experimental setup with emulated test nodes 100
Figure 5-49 CPU utilization with respect to the number of test nodes 101
Figure 5-50 Impact of balancing the load across two RAN microservices 102
Figure 5-51 CPU utilization for PHY and Contiki under different traffic loads 103
Figure 6-1 ADS Network connectivity mapping 105
Figure 6-2 ADS drone connectivity mapping 106
Figure 6-3 ADS Utilized drone 106
Figure 6-4: iDrOS experimental setup 107

Figure 6-5 DCAS experimental setup 108
Figure 6-6: DCAS validation setup 108
Figure 6-7 ADS-UC1 mission timing sequence 109
Figure 6-8: DCAS latency measurement setup 110
Figure 6-9 IDRoS execution times when migrating components 111
Figure 6-10 IDRoS migration performance 112
Figure 6-11 Scalability testing with 20 emulated drones 113
Figure 6-12 Scalability testing with 50 emulated drones 114
Figure 6-13 Control webpage of 20 emulated drones 115
Figure 6-14 Overview of ADS-UC2 system 116
Figure 6-15 ADS-UC2 mission scenario and flow 116
Figure 6-16 ADS-UC2 mission timing sequence 117
Figure 6-17 Overview of EagleEYE key modules 118
Figure 6-18 EagleEYE output 119
Figure 6-19 Overview of EagleStitch key modules 120
Figure 6-20 EagleStitch output 120
Figure 6-21 Visualization output of ADS-UC2 during two hours stress test 122
Figure 6-22 Stress test setup at ASKEY's lab 122
Figure 6-23 EagleEYE System Deployment with BASS 123
Figure 6-24 EagleEYE scalable architecture for multiple drone inputs 124
Figure 6-25 EagleEYE scalability test with 6 drones Input 125

List of Acronyms

3GPP	3rd Generation Partnership Project
4G	4th Generation
5G	5th Generation
5G NR	5th Generation New Radio
5G-DIVE	eDge Intelligence for Vertical Experimentation
AC	Alternating Current
ACK	Acknowledgement
ADS	Autonomous Drone Scout
AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
APPs	Applications
AWS	Amazon Web Services
BASS	Business Automation Support Stratum
CDF	Cumulative Distribution Function
CNN	convolutional neural networks
COVID-19	Coronavirus Disease 2019
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CRAN	Cloud Radio Access Network
D1.1	Deliverable 1.1
D2.3	Deliverable 2.3
D3.1	Deliverable 3.1
D3.2	Deliverable 3.2
DASS	Data Analytics Support Stratum
DC	Data Centre
DCAS	Drone Collision Avoidance System
DEEP	5G-DIVE Elastic Edge Platform
DL	Downlink
DNS	Domain Name System
DoD	Dual Object Detection
DT	Digital Twin
E2E	Exchange-to-Exchange
EAB	Ericsson AB
EagleEYE	Aerial Edge-enabled Disaster Relief Response System
EagleStitch	2D Panorama Stitching System
EDC	Edge Data Centre
EFS	Edge and Fog System
eMMB	Enhanced Mobile Broadband
eNB	Evolved Node Base Station
EPC	Evolved Packet Core
EuCNC	European Conference on Networks and Communications

E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
FIM	Fog Infrastructure Manager
FOrcE	Fog Orchestration Manager
Gbps	Gigabytes per second
GGC	Green grass Core
GHz	Gigahertz
gNB	5G NR base stations
GPS	Global Positioning System
GPU	Graphics processing unit
GUTI	Globally Unique Temporary Identifier
H2020	Horizon 2020
HD	High Definition
HTTP	Hypertext Transfer Protocol
HW	Hardware
I4.0	Industry 4.0
ID	Identity
IDCC	Interdigital Europe
IDrOS	Internet Drone Operating System
IE	Intelligence Engine
IEEE	Institute of Electrical and Electronics Engineers
IESS	Intelligence Engine Support Stratum
III	Institute for Information Industry
IIPFD	Intelligent Image Processing for Drones
IMDEA	Madrid Institute for Advanced Studies
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
IP	Internet Protocol
ITRI	Industrial Technology Research Institute
JSON	JavaScript Object Notation
K8S	Kubernetes
KPI	Key Performance Indicator
LED	Light Emitting Diode
LoRa	Long Range
LTE	Long Term Evolution
LTS	Long Term Support
LWM2M	Lightweight M2M standard
MAC	Medium Access Control
mAP	mean Average Precision
Mbps	Megabits per second
MiniPC	Minicomputer
MIRC	Microelectronics and Information Systems Research Centre
ML	Machine Learning

MME	Mobility Management Entity
mMTC	Massive Machine Type of Communication
MQTT	Message Queuing Telemetry Transport
MTC	Machine Type Communications
NBI	Northbound interface
NCTU	National Chiao Tung University
NDI	Network Device Interface
NR	New Radio
NSA	Non-standalone
OCS	Orchestration and Control System
OPTUNS	SDN-based optical-tunnel-network system
PER	Packet Error Rate
PHY	Physical (Layer)
PiH	Person in need of Help
PoC	Proof of Concept
PTZ	Pan, Tilt and Zoom
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RAT	Radio Access Technology
RF	Radio Frequency
ROS	Robot Operating System
RSRP	Reference Signal Received Power
RSSI	Received Signal Strength Indicator
RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
SA	Standalone
SCTP	Stream Control Transmission Protocol
SFP	Small Form-factor Pluggable
Sgw	Serving Gateway
SINR	Signal to interference plus noise radio
SLA	Service Level Agreement
SQL	Structured Query Language
TBD	To Be Determined
TCP	Transmission Control Protocol
TED	Test Dataset
TELCA	Telcaria S.A
TLS	Transport Layer Security
TW	Trained Weight
UAV	Unmanned Aerial Vehicle

UC	Use Case
UC3M	University Carlos III of Madrid
UDP	User Datagram Protocol
UE	User Equipment
UI	User Interface
UL	Uplink
ULUND	Lund University
URLLC	Ultra-reliable low-latency communication
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity
WP	Work Package
ZDM	Zero Defect Manufacturing
ZMQ	ZeroMQ

Executive Summary

This deliverable of D3.3 is the final deliverable of WP3 after D3.2 (D3.1: Definition and setup of vertical trial sites, 2020). The goal of this deliverable is to provide the final validation results of different 5G-DIVE use cases defined in D1.1 (D1.1: 5G-DIVE architecture and detailed analysis of vertical use cases, 2020), i.e., Digital Twin (DT), Zero Defect Manufacturing (ZDM) and Massive Machine-Type-of-Communication (mMTC) use cases for Industry 4.0 (I4.0) trial, and Drone Collision Avoidance System (DCAS) and Intelligent Image Processing for Drones (IIPFD) use cases for Autonomous Drone Scouting (ADS) trial. The main achievements of D3.3 are: (1) Final experiments of each use case have been performed for use-case implementation validation. The experimental results are evaluated against the technical requirements defined in D1.1, which indicates that the developments of the use cases are all on track. (2) The final use case integration is provided, presenting the basic ideas of how all use cases can be integrated under one common DEEP platform (5G-DIVE Elastic Edge Platform). The following provides a summary of the contents in Sections 2-6, respectively.

In Section 2, the final 5G solution validation results are presented. In particular, 5G-SA solution is mainly used for I4.0 uses cases and 5G-NSA solution is used for ADS use cases.

In Section 3, The main features are (1) BASS is developed orchestrator drivers to support different orchestrator frameworks (e.g., K8s and FogO5) which are used by different use cases; (2) IESS provides a common IESS catalog for storing use-case specific ML/AI models especially for DT; (3) most of the use cases adopted a common DASS component using Eclipse Zenoh developed in 5G-DIVE. In this way, a common DEEP platform can serve multiple vertical use cases simultaneously. In addition, validation results have been shown in this section.

In Section 4, the I4.0 and ADS trials are planned in Taiwan. However, this plan gets affected by the COVID-19 pandemic situation. Therefore, we provide an update regarding the trial site information for both I4.0 and ADS trials. In particular, more details are provided on the adopted alternative plan for I4.0 trial (5TONIC lab in Spain). Then, we present the final use-case integration regarding how all 5G-DIVE use cases will be integrated under one common DEEP platform.

In Section 5, three use cases have been validated for the I4.0 trial. The results indicate that the development of three use cases delivered the aimed targets. DT focused on the development of additional AI/ML features and their integration in the DEEP platform. In particular, movement prediction, obstacle avoidance and SLA Enforcer. Then, ZDM use case focused on a new object detection engine, integration with AWS and integration with 5G network. In addition, mMTC use case focused on application layer features, RF fingerprinting, orchestration and automation features. All the use case above also provided DEEP integration validation, long-term validation and scalability analysis.

In Section 6, two use cases have been validated for the ADS trial. The results indicate that the development of two use cases delivered the aimed targets. ADSUC1 focused on adopting Drone Collison Avoidance System (DCAS) into the edge for drone fleet navigation and Internet Drone Operating System (IDrOS). Then, ADSUC2 focused on 5G-NSA validation, EagleEYE and Egale stitching. In addition, DEEP integration validation, long-term validation and scalability analysis are provided. Finally, Section 7 concludes the deliverable.

1. Introduction

One of the key objectives of 5G-DIVE WP3 is to perform field trials for pilots over the E2E in European and Taiwanese testbeds. The two verticals are Industry 4.0 (I4.0) and Autonomous Drone Scouting (ADS) have worked extensively even under COVID-19 restriction to validate the solution. Also, WP3 had integrated the novel 5G-DIVE DEEP strata (developed in WP2) into the 5G-CORAL baseline architecture. The deployment of the 5G-DIVE platform occurred in all trial sites, including software and hardware components for the aforementioned pilots. This deliverable addressed the long and short validation for 3 tires of 5G-DIVE solution. The tires include 5G connectivity, Edge computing, and AI. For 5G connectivity, a comprehensive set of validation featuring high-throughput and low-latency demanding applications at the vicinity of the end-user and in real-world environments is possible using 5G-SA and 5G-NSA solutions. More details on 5G solution validation are presented in Section 2. Edge computing and AI have been reported individually for each use case.

On the other hand, it is mandatory to orchestrate and manage the pilot through DEEP platform. Hence, in Section 3, we validate DEEP platform. This covers three parts: BASS, DASS, IESS. For BASS, resource management, automates the lifecycle management, user-friendly UI and active monitoring features have been elaborated. In addition, BASS performance has also been evaluated in terms of scalability, availability and reliability. For DASS, data preprocessing, storage, as well as data dispatching features have been elaborated and validated. In addition, the DASS has been evaluated in terms of scalability and performance. The scalability test was a mesh routing at scale test and was performed at Zenoh-net level. The performance tests were carried out in peer-to-peer and in brokered communication mode, in both the Zenoh API, and Zenoh-net layer. For IESS, it offers features to facilitate the adoption of AI and ML functionalities. Besides, IESS performance has also been evaluated in terms of scalability, availability and reliability.

Section 4 presented the final integration where an update regarding more information on the trial sites of I4.0 trials in Spain and ADS trials in Taiwan is provided. Also, the final use-case integration is presented regarding how all 5G-DIVE use cases in trials are integrated with one common DEEP platform.

Section 5, the final validation results of the three I4.0 use cases are presented. For Digital Twin (DT) use case part, focused on the development of additional AI/ML features for and their integration in the DEEP platform (i.e. movement prediction, obstacle avoidance and SLA Enforcer). It also provided 5G-SA profiling network measurements. For Zero Defect Manufacturing (ZDM) use case, a new object detection engine has been integrated and the ZDM use case has been integrated with AWS Wavelength. Moreover, the various ZDM experiments are used to evaluate 5G connectivity, fog and edge computing, DEEP components. For Massive Machine-Type-of-Communication (mMTC) use case, final validation for the added orchestration features using K8s for automation and auto-scaling is performed. Besides, intelligent RF fingerprinting is developed, integrated into the tested and validated.

Section 6 provides a description of the integration of 5G-DIVE solution as well as the complete end-to-end deployment of 5G-enabled edge infrastructure for Autonomous Drone scout (ADS) has been described. In particular, ADS is used for a disaster relief response system into two use cases. The first ADS use case (ADS-UC1) is drone fleet navigation. The second ADS use case (ADS-UC2) is intelligent image processing for drones. In ADS-UC1, the drone fleet can support several applications for relief efforts including DCAS. While in ADS-UC2, it supports Eagle EYE, and EagleStitch. The system (i.e. ADSUC1 and ADSUC2) has shown the scalability and availability required for such public safety mission. In addition, the 5G-DIVE DEEP platform (e.g.: BASS, DASS, IESS) utilization toward enhancement of the overall performance has been elaborated. For example, it provided high throughput data publishing framework via the DASS, automatic training of AI model via the IESS, as well as automatic application deployment and lifecycle management via the BASS.

Finally, Section 7 concludes the deliverable with the key achievements for each use case.

2. 5G Solution Validation

In this section, we present 5G SA solution used in I4.0 field trials as elaborated in Section 2.1. Also, we present 5G NSA solution used in ADS field trials as elaborated in Section 2.2.

2.1. 5G SA solution for I4.0 trials in 5TONIC

A 5G SA trial system is deployed at 5TONIC, the trial site for I4.0 use cases of 5G-DIVE. The 5G SA system is designed based on Ericsson Radio Dot System, where 5G DOT is deployed on a cable ladder and the rest of the nodes are installed in one half-size rack (referred to as flightrack with a minimal footprint which is easy to transport), as shown in Figure 2-1. For UEs, we have two ASKEY HPUE RTL0330 UEs and two other test UEs based on RaspberryPi HAT (Raspberry Pi HAT), as shown in Figure 2-2. The UEs are based on Qualcomm SDX55 chipset.

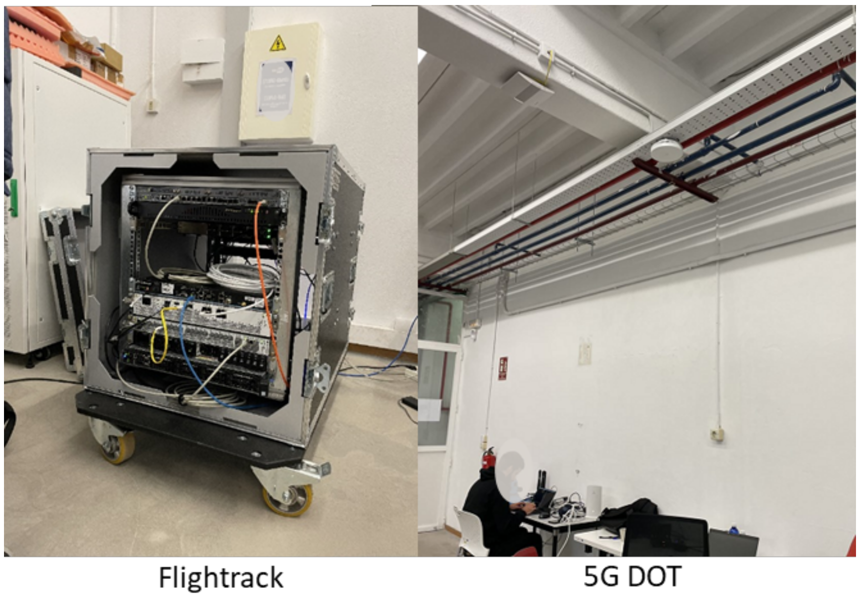


FIGURE 2-1 5G SA trial system installed in 5TONIC

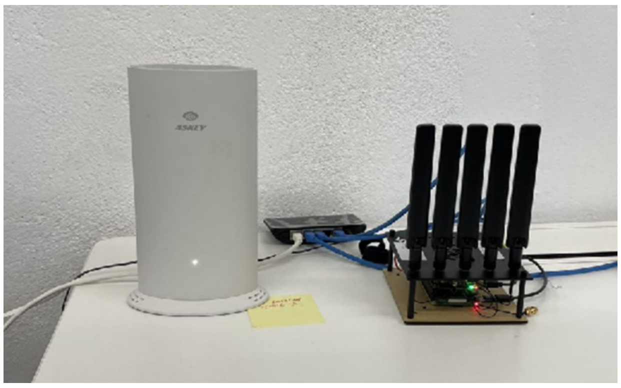


FIGURE 2-2 Two types of UEs used
(left: ASKEY RTL0330, right: RaspberryPi HAT)

Figure 2-3 shows the system design of the flightrack which houses an IRU 8846, a Baseband BB6630, UPF server, app server, a switch and a firewall. The 5G connectivity supports band n79 and 100 MHz bandwidth with 4T4R, i.e. 4 transmit and 4 receive antennas. A local GNSS sync source provides the

SYNC signal to the BB6630 using an Ethernet cable. The Control Plane of 5GC is located remotely in Sweden and connected to 5G RAN a.k.a flightrack over the Internet via an IP Sec tunnel. The User Plane Function (UFP) of 5GC, i.e. UPF server, is deployed locally on the flightrack.

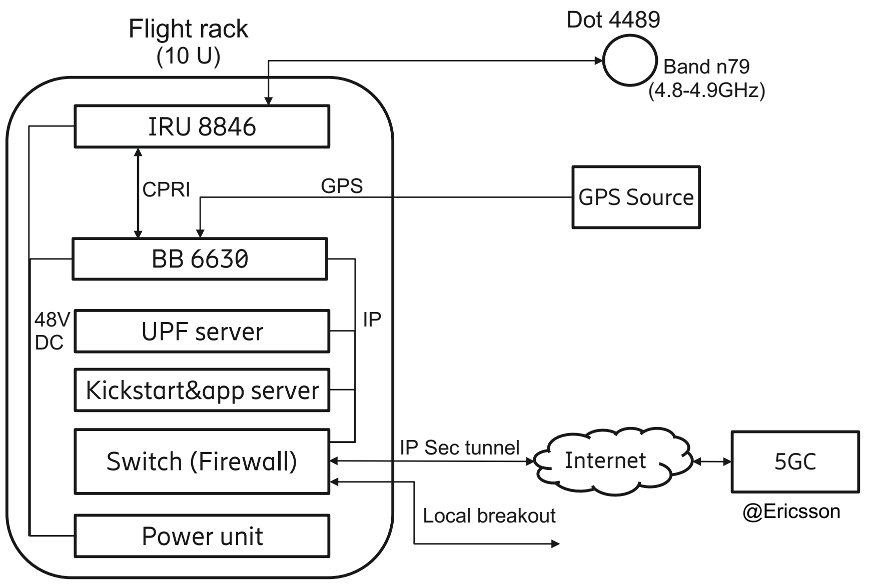


FIGURE 2-3 5G SA trial system design

The trial network setup at 5TONIC has 3 main components.

- 5G RAN and UPF at the trial site at 5TONIC, Madrid, Spain.
- 5GC at a remote location at Kista, Sweden.
- Edge servers at the 5TONIC data centre in the server room next to the trial site, Madrid, Spain.

These three components are established in separate physical locations and belong to different private networks. Hence it was necessary to create efficient connectivity among them. The network topology and connectivity are illustrated in Figure 2-4.

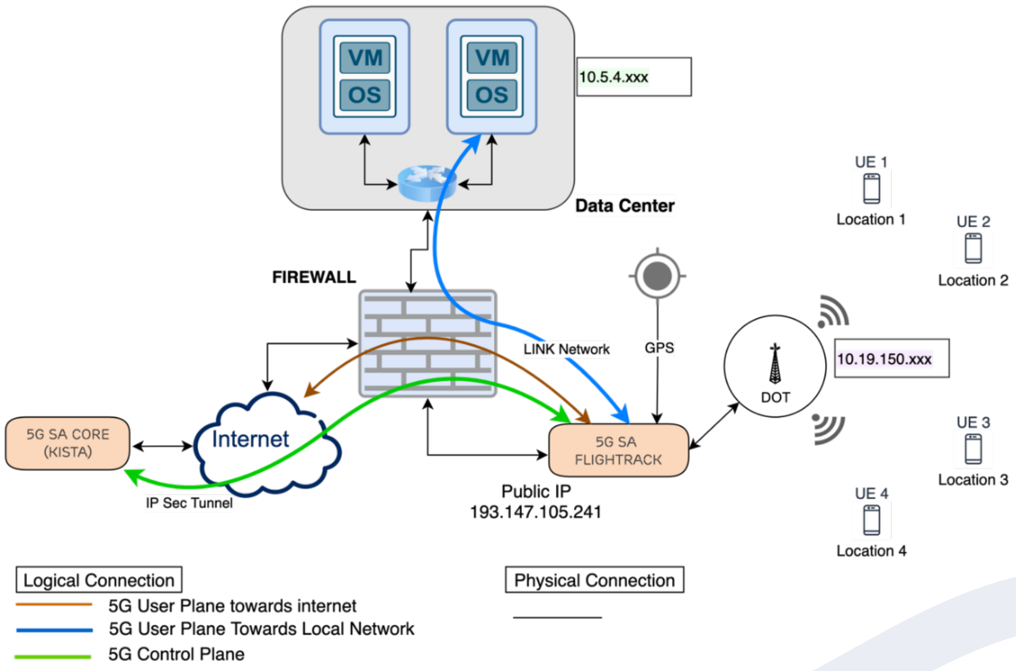


FIGURE 2-4 Network Topology of the 5G SA trial system

Connectivity between 5G RAN and 5GC

5G RAN is connected to the 5GC over the Internet via an IP SEC tunnel. We achieve this by configuring a public IP address at the 5G RAN firewall and connecting to the 5G CORE. This allows us to provide a 5G control plane from a remote site making the connectivity setup flexible and cloud friendly. In addition to that, we can manage and configure the 5G RAN setup remotely over the same tunnel connection securely.

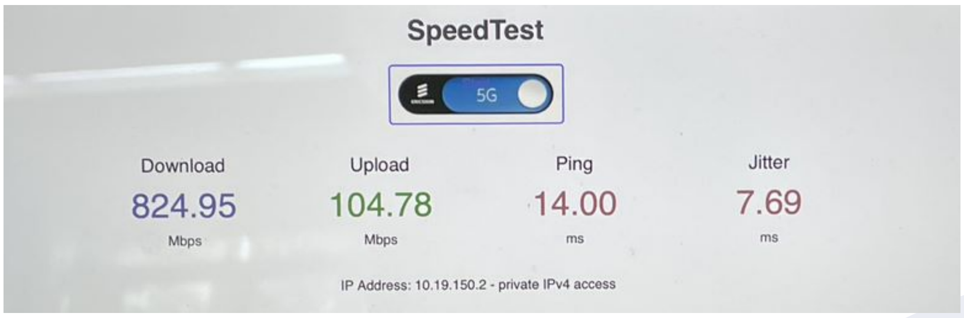
Connectivity between 5G RAN and Edge servers

The EDGE servers necessary for the I4.0 use cases are behind the firewall of 5TONIC lab and reachable over the Internet using a secure VPN connection. These servers belong to a local network [10.5.4.0/24] which is different than the 5G network to which the UEs [10.19.150.0/24] belong to. Although the UEs have access to the Internet, being on a different network they can't reach the EDGE servers directly. The use of VPN at the UEs to establish a connection with EDGE server is not efficient in terms of the potential performance and latency impairments. It was important for UEs to have a reliable and low latency connection towards the EDGE servers. Towards this, we created a Link Network between the 5G SA setup (via switch and firewall on the flightrack) and 5TONIC data center over their firewall to establish a close loop connection between 5G UEs [10.19.150.0/24] and Edge Servers (VMs) [10.5.4.0/24] networks. With this new Link Network, the UEs have reliable and low latency connections to the EDGE servers.

Connectivity to Internet

As described before, UPF is deployed in the flightrack locally. Therefore, the user plane data are locally connected to the Internet via UPF. With an IP Sec tunnel towards remote 5GC and a local Link Network with 5TONIC data center, the 5G SA trial setup provides all necessary connectivity for I4.0 use cases. On top of that, we can remotely manage and configure the 5G SA setup as needed.

To verify the performance of the setup, we list some measurement results done at 5TONIC. These results were done with the SpeedTest application preinstalled in the application server, which is usually used for performance verification. Figure 2-5 shows a screenshot of one SpeedTest measurement.


SpeedTest

5G

Download	Upload	Ping	Jitter
824.95	104.78	14.00	7.69
Mbps	Mbps	ms	ms

IP Address: 10.19.150.2 - private IPv4 access

FIGURE 2-5 Screenshot of the SpeedTest measurement

Figure 2-6 shows the locations of 5 measurements done in 5TONIC. For each location, the UE was placed on the table where I4.0 use case trials would take place. And Table 2-1 lists the measurement results. It shows that more than 100 Mbps in UL and more than 750 Mbps in DL are achieved consistently. The average round-trip time (RTT) is up to 20 ms while the jitter remains under 14 ms. These results validated that the system is up & running properly at 5TONIC and ready for I4.0 use case trials.

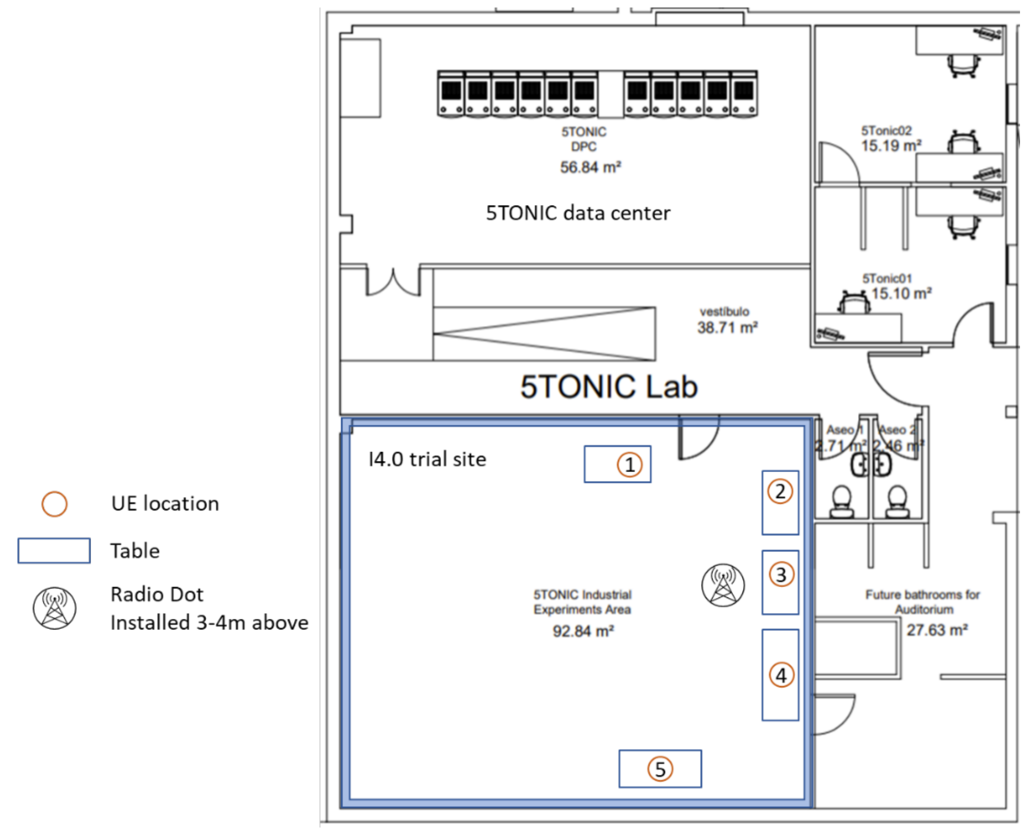


FIGURE 2-6 Measurement locations at 5TONIC

TABLE 2-1 Throughput measurement results

UE Location	Uplink throughput (Mbps)	Downlink throughput (Mbps)	Latency (RTT) (ms)	Jitter (ms)
Location 1	112.85	800.95	12	7
Location 2	112.23	866.11	20	13.27
Location 3	111.86	872.36	14	6.49
Location 4	107.48	825.96	16	10.63
Location 5	106.30	757.54	12	5.81

2.2. 5G-NSA solution for ADS trials in MIRC

The 5G NSA mobile network at NCTU site is built for ADS use cases. This 5G NSA network consists of the Radio Access Network and the Core network. The eNB-gNB combo base station made by Askey Computer supports both 4G and 5G radios. They follow 3GPP EN-DC Option 3X procedures for a UE (a drone) to switch its connectivity to 5G when the signal is strong enough and otherwise fall back to 4G The overall architecture is shown in Figure 2-7. In Figure 2-7, the green topology of lines depicts the cabling. The text near a line describes the 3GPP protocol interfaces. The other lines without text have been traffic local breakout, and thus those packets in the lines are of regular IP.

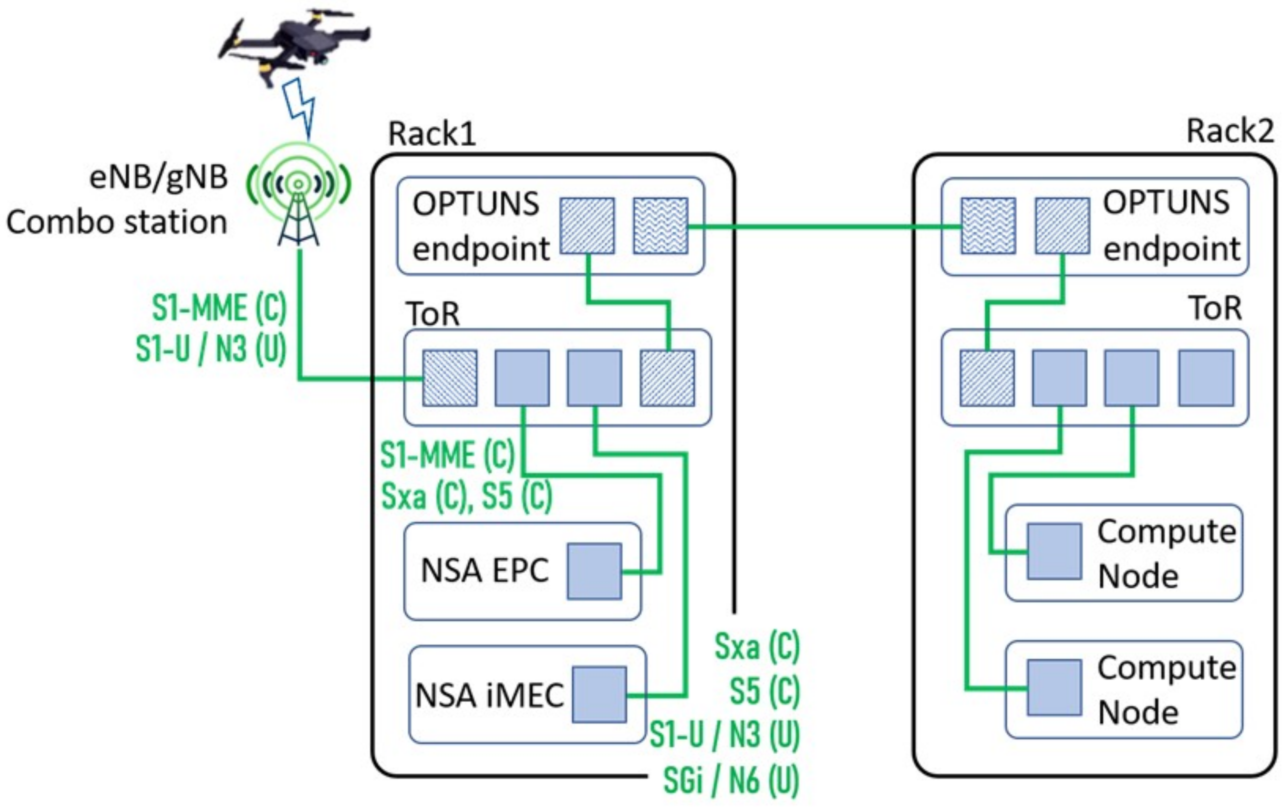


FIGURE 2-7 The overall 5G NSA structure

The fully-virtualized core network is developed by III and ITRI as shown in Figure 2-8 and Figure 2-9. III upgraded the former 4G Core to enable 5G signalling. Meanwhile, ITRI provides the NSA-enabled iMEC, which runs the specialized serving gateway to perform local breakout of UE traffic to those virtual servers, such as EagleEye and EagleStitch, on compute nodes. iMEC also provides a GUI interface for operators to deploy or to remove a virtual server.

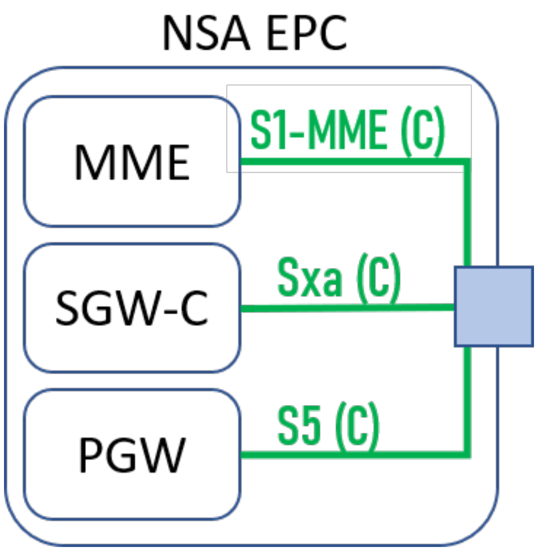


FIGURE 2-8 Internal structure of 5G NSA EPC

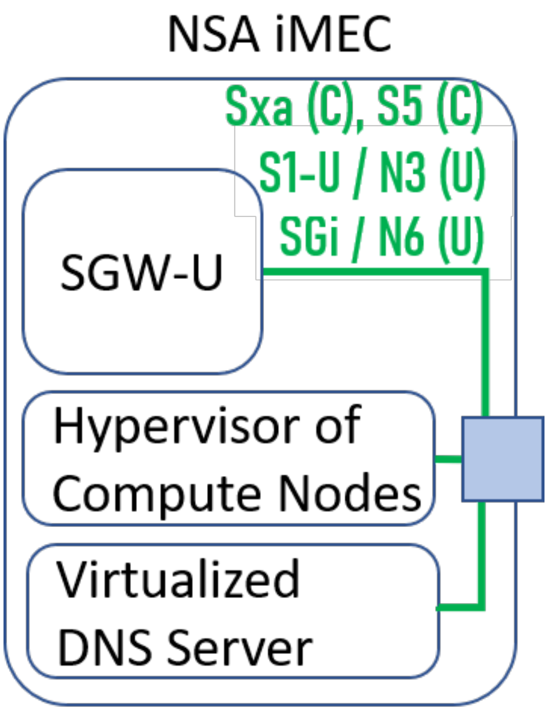


FIGURE 2-9 Internal structure of 5G NSA iMEC

Connectivity between NSA RAN and NSA EPC

An eNB-gNB combo base station connects to EPC over a local Ethernet network. After booting up, it registers its existence to MME at NSA EPC. During run time, it forwards the controlling signals between a UE and MME, as well as delivers data between UE and serving gateway with the ability to do local breakout. During run time, the base station continuously measures the radio strength and inform MME to upgrade or fall back data path to gNB or eNB radio for UEs accordingly.

Connectivity between NSA RAN and iMEC with Virtualized Drone Servers

No matter eNB or gNB that a UE is connected to, the UE traffic reaches the serving gateway, which runs on iMEC. The serving gateway performs local breakout, decapsulate GTP-U header, and then forwards the traffic to localized application servers that also run on iMEC. The networking performance of this NSA mobile network has been evaluated at NCTU trial site. The measuring positions are shown in Figure 2-10, and the results are in Table 2-2.

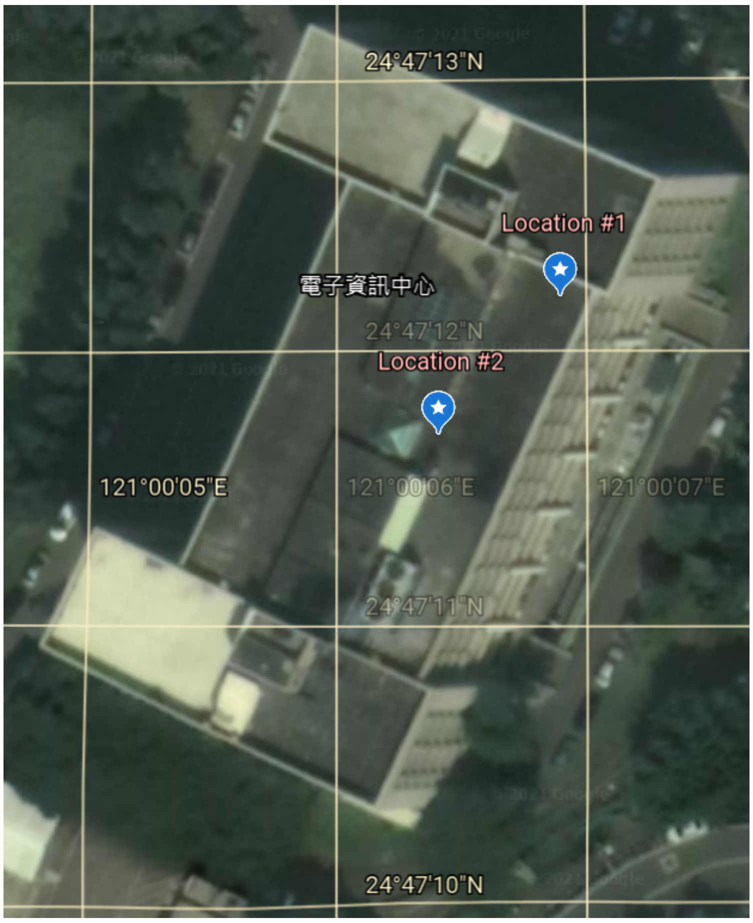


FIGURE 2-10 5G NSA NSA locations of measurement

TABLE 2-2 Various SpeedTest measurement results

UE Location	Uplink throughput (Mbps)	Downlink throughput (Mbps)	Latency (RTT) (ms)	Jitter (ms)
Location 1	99.1	177.0	39.512	9.308
Location 2	126.0	166.1	51.735	13.615

3. DEEP Platform Validation

One of the main challenges of the 5G-DIVE project is the design and development of the DEEP platform, an intelligent software system for the unified access to heterogeneous and remote clusters of resources and the integration of multiple use cases. The final design is presented in D1.3 (D1.3 , 2021), while details about the implementation are reported in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021).
The functionalities and performance of the platform have been validated in multiple experiments and benchmarking activities. Since the DEEP aims at providing advanced features to vertical use-cases, the project effort focused on multiple iterative integration cycles in order to get quick feedback about proposed features and their usability. The integration campaign has culminated in the setup of a field trial testbed at 5TONIC premises in Spain. Here, the I4.0 use cases have been all deployed on the DEEP platform that manages and orchestrates a common pool of computing, networking, and storage resources. Due to travel and technical restrictions, ADS use-cases have been integrated with a far remote testbed configuration where the DEEP platform was deployed in Spain and the Vertical Premises were located in Hsinchu City, Taiwan.
The next sections report separately the validation of the three main components of the DEEP platform, the Business Analysis Support Stratum (BASS), the Data Analysis Support Stratum (DASS), and the Intelligence Engine Support Stratum (IESS).

3.1. BASS Validation results

The BASS' functional and non-functional features have been validated against all of the use cases of the project during a preliminary integration phase and finally during the field trials. The central component of the DEEP platform takes care of managing the resources in vertical premises (i.e., regions), automates the lifecycle management of the use cases, and provides advanced features like a user-friendly UI and active monitoring to ease the interaction of verticals with the infrastructure and vertical services. The BASS performance has also been evaluated in terms of scalability, availability and reliability.

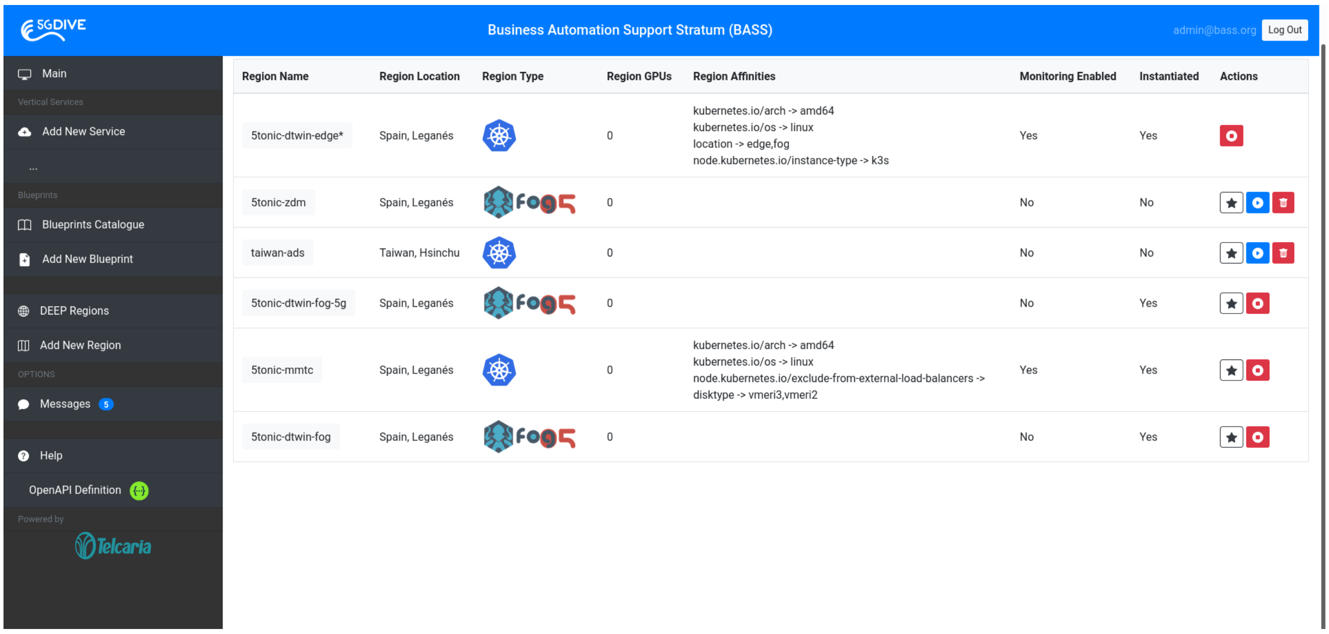
3.1.1. Multi-region orchestration feature validation

The BASS provides the unified management of multiple clusters of computing, networking, and storage resources. A cluster or aggregation of clusters owned by a single vertical is named Vertical Region and it can be local or remote with respect to the location of the BASS instance that manages it.
For the field trials at 5TONIC premises, we emulated a scenario with three vertical regions, deployed locally in the data centre but configured as they were remote (see Section 4). Plus, we included a real remote region located in Hsinchu City, Taiwan. Table 3-1 provides more information about the configuration of each region.

TABLE 3-1 Region configured in 5Tonic trials

Region name	Orchestration technology	Orchestrated resources
5tonic-dtwin-edge	K3s	1 edge node
5tonic-dtwin-fog	Fog05	2 fog nodes
5tonic-mmtc	Kubernetes	3 edge nodes
5tonic-zdm	Fog05	1 edge node, 1 fog node
NCTU-Hsinchu City, Taiwan.	K3s	1 edge node

As shown in the previous table, each region is implemented with a different orchestration platform. The BASS provides a common, unified interface to manage both Fog05 and Kubernetes (plus compatible flavours like K3s) based regions. The software is designed with a plugin-based pattern in order to add support for more in the future.



Region Name	Region Location	Region Type	Region GPUs	Region Affinities	Monitoring Enabled	Instantiated	Actions
5tonic-dtwin-edge*	Spain, Leganés		0	kubernetes.io/arch -> amd64 kubernetes.io/os -> linux location -> edge,fog node.kubernetes.io/instance-type -> k3s	Yes	Yes	
5tonic-zdm	Spain, Leganés	Fog05	0		No	No	
taiwan-ads	Taiwan, Hsinchu		0		No	No	
5tonic-dtwin-fog-5g	Spain, Leganés	Fog05	0		No	Yes	
5tonic-mmtc	Spain, Leganés		0	kubernetes.io/arch -> amd64 kubernetes.io/os -> linux node.kubernetes.io/exclude-from-external-load-balancers -> disktype -> vmeri3,vmeri2	Yes	Yes	
5tonic-dtwin-fog	Spain, Leganés	Fog05	0		No	Yes	

FIGURE 3-1 BASS Web view of the regions for the final trials

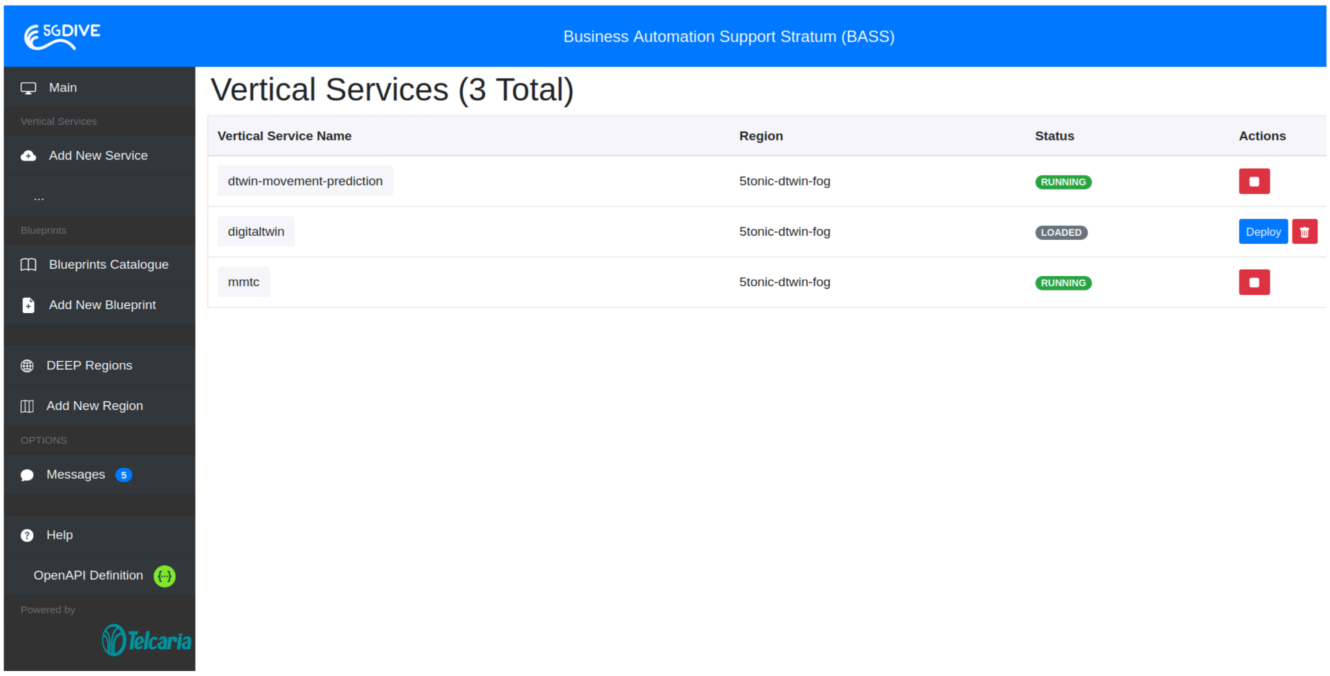
During the trials at 5TONIC premises, we validated the correct functionality of the multi-region management region. Figure 3-1 shows a screenshot of the BASS user interface showing the managed regions.

3.1.2. Service management feature validation

The BASS provides several tools in order to define a Vertical Service and to manage its lifecycle. The Vertical Service Descriptor (VSD) is a data model to define services on the BASS, describing components, connections, and any other information for the deployment. The descriptor is applicable to all the orchestration platforms supported by the BASS, as shown in Section 3.1.1, providing a common language for the definition of services. Similarly, the BASS defines a unified lifecycle for Vertical Services that are applicable to all the orchestration platforms supported. The management of the lifecycle is facilitated by a simple Web User Interface.

To demonstrate how the BASS facilitates the definition and management of a Vertical Service, with respect to its direct implementation on an orchestration platform, we can compare the descriptors. We use lines of code (LOC) as a raw measurement of the complexity and we select the mMTC use-case (see Section 5.3) as an example. The use case is deployed on a Kubernetes compatible orchestration platform (K3S). Kubernetes deployment files are expressed in YAML, while the BASS' VSD is expressed in JSON. Since YAML is a superset of JSON, the conversion from JSON to YAML is possible without any information loss. The mMTC VSD is converted to YAML in order to perform a meaningful comparison. Comparing the VSD and Kubernetes deployment files for the mMTC use case, the first thing we can notice is that the BASS descriptor is contained in a single file. On the contrary, Kubernetes needs multiple files in order to fully describe the deployment: six YAML files are needed in order to completely describe the deployment of mMTC use-case in Kubernetes. Actually, thanks to YAML

features, the different files can be merged into a single one but this is not a best practice as it generates a more complicated file harder to manage for the user. Concerning the complexity in terms of LOC, the Kubernetes deployment files sum up to 193 lines while the BASS' VSD file only contains 44 lines, a reduction of about 77% (the VSD, in YAML format, is provided in Section 8). The BASS achieves this significant result thanks to its augmented intelligence that abstracts the vertical service definition from Kubernetes technical details, infrastructure details, and provides safe defaults for common requirements from verticals collected during the project.
During the trials at 5TONIC premises, all the use-cases were correctly deployed and managed by the BASS. The verticals were able to create, deploy, stop, and delete several versions of their services. Figure 3-2 provides a screenshot of the Web User Interface during the trials, showing several Vertical Services managed with the BASS at the same time.



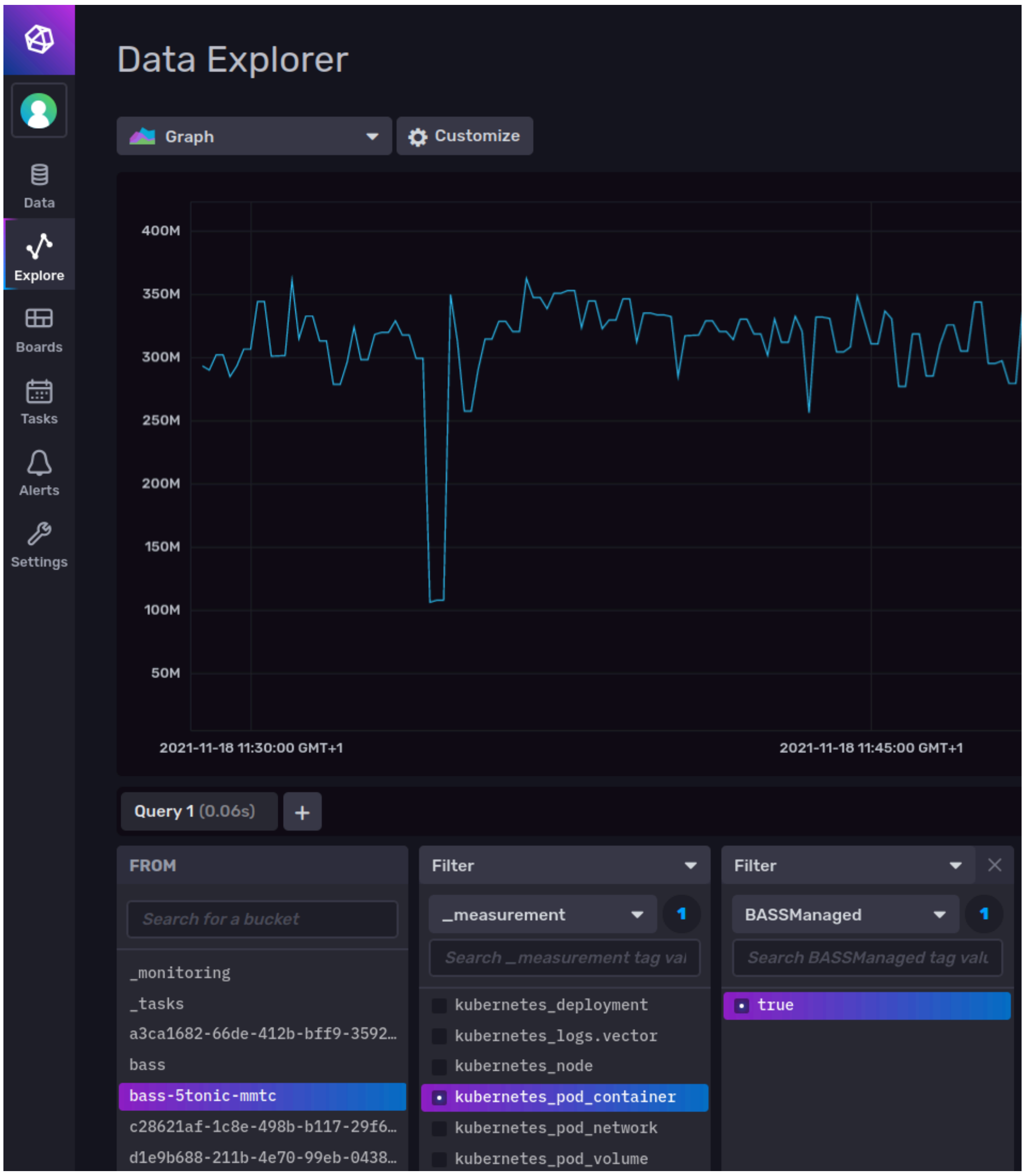
Vertical Services (3 Total)

Vertical Service Name	Region	Status	Actions
dtwin-movement-prediction	5tonic-dtwin-fog	RUNNING	
digitaltwin	5tonic-dtwin-fog	LOADED	Deploy
mmtc	5tonic-dtwin-fog	RUNNING	

FIGURE 3-2 BASS web view of multiple vertical services

3.1.3. Active Monitoring feature validation

The BASS offers to the verticals a monitoring solution working out-of-the-box. For Vertical Services deployed on resources orchestrated by Kubernetes, it automatically enables the monitoring of per-component CPU usage, memory usage, network usage, and disk usage. Furthermore, it provides active probes templates to integrate custom metrics from the Vertical Service into the monitoring system. The storage and query engine are done with InfluxDB (InfluxDB, n.d.), while the probes are deployed as Telegraf plugins (Telegraf documentation, 2021), integrated as sidecar containers.
During the trials at 5TONIC premises, we validated the monitoring features of the BASS for all the use-cases deployed on regions orchestrated by Kubernetes. Figure 3-3 shows a screenshot of the metrics collected inside InfluxDB for the mMTC use case.



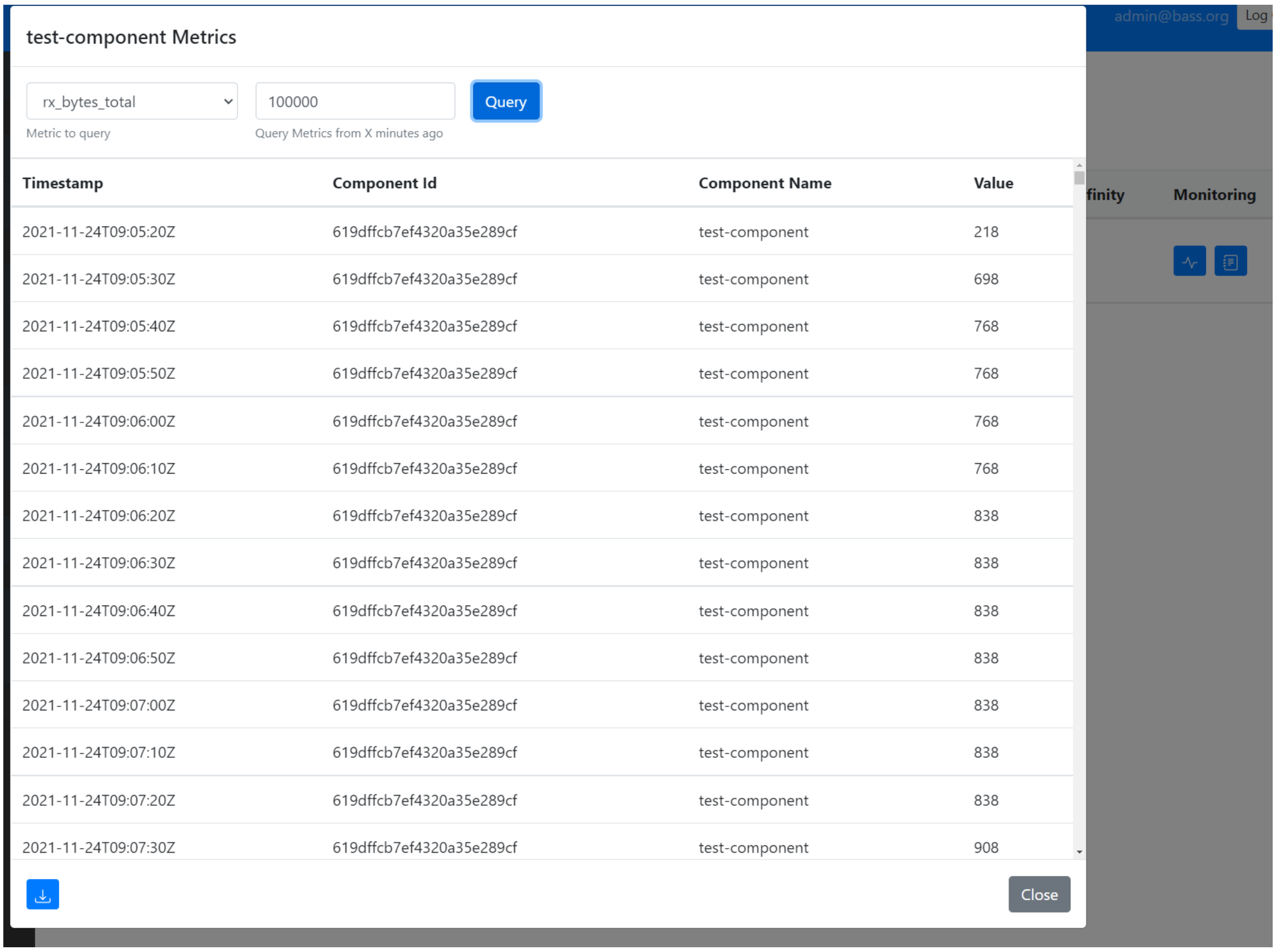
FROM: _monitoring, _tasks, a3ca1682-66de-412b-bff9-3592…, bass, bass-5tonic-mmtc (selected), c28621af-1c8e-498b-b117-29f6…, d1e9b688-211b-4e70-99eb-0438…

Filter: _measurement (1): kubernetes_deployment, kubernetes_logs.vector, kubernetes_node, kubernetes_pod_container (selected), kubernetes_pod_network, kubernetes_pod_volume

Filter: BASSManaged (1): true (selected)

FIGURE 3-3 INFLUXDB metrics visualization for mMTC use case

A basic visualization of metrics is also available directly from the BASS Web User Interface. Figure 3-4 shows an example visualization of a metric measuring the total received bytes on the interface of a component.



test-component Metrics

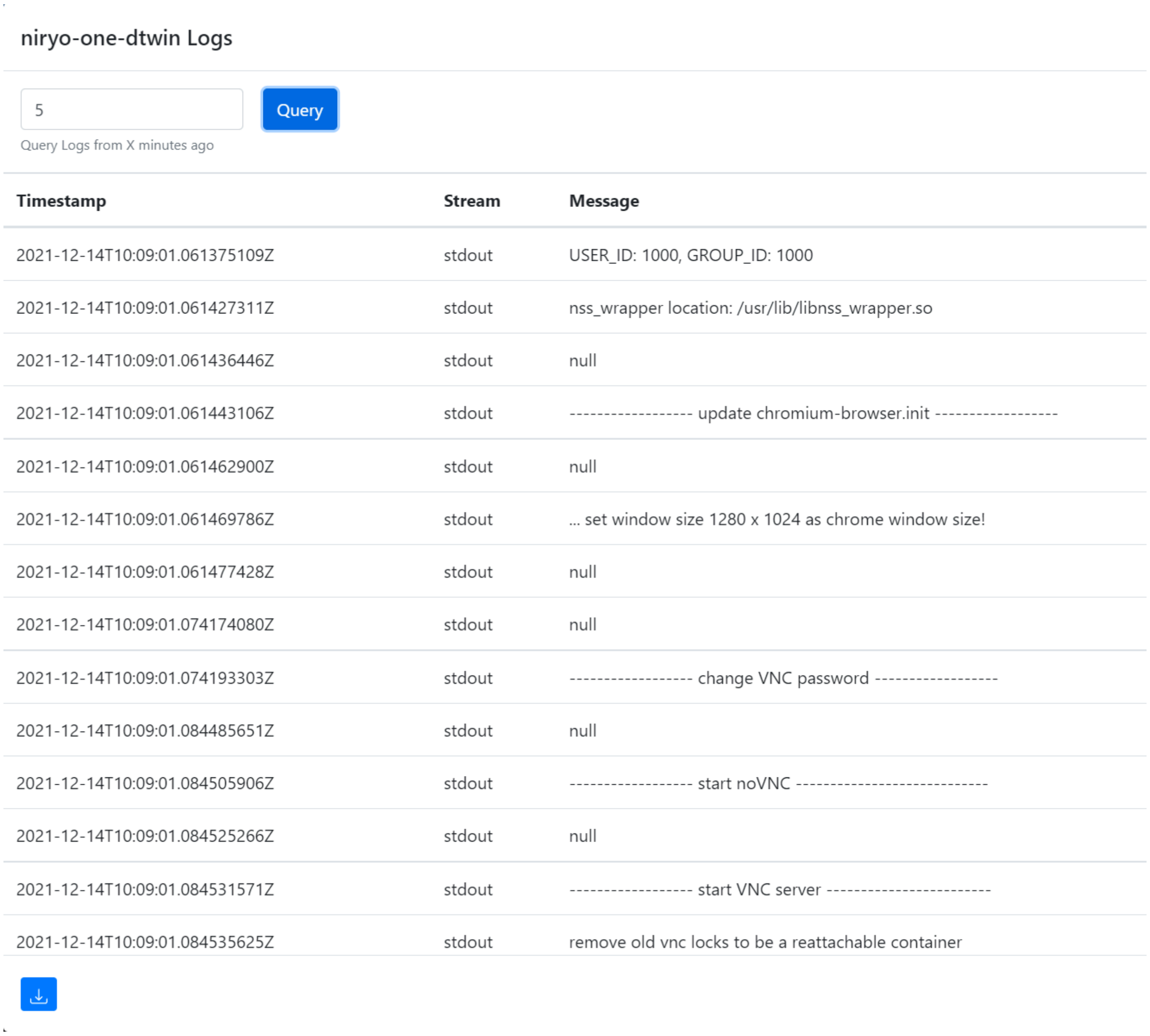
Metric to query: rx_bytes_total

Query Metrics from X minutes ago: 100000

Timestamp	Component Id	Component Name	Value
2021-11-24T09:05:20Z	619dffcb7ef4320a35e289cf	test-component	218
2021-11-24T09:05:30Z	619dffcb7ef4320a35e289cf	test-component	698
2021-11-24T09:05:40Z	619dffcb7ef4320a35e289cf	test-component	768
2021-11-24T09:05:50Z	619dffcb7ef4320a35e289cf	test-component	768
2021-11-24T09:06:00Z	619dffcb7ef4320a35e289cf	test-component	768
2021-11-24T09:06:10Z	619dffcb7ef4320a35e289cf	test-component	768
2021-11-24T09:06:20Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:06:30Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:06:40Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:06:50Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:07:00Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:07:10Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:07:20Z	619dffcb7ef4320a35e289cf	test-component	838
2021-11-24T09:07:30Z	619dffcb7ef4320a35e289cf	test-component	908

FIGURE 3-4 Metrics visualization from the BASS' web user interface

The BASS also embeds the visualization of logs for the managed components, as shown in Figure 3-5.



Timestamp	Stream	Message
2021-12-14T10:09:01.061375109Z	stdout	USER_ID: 1000, GROUP_ID: 1000
2021-12-14T10:09:01.061427311Z	stdout	nss_wrapper location: /usr/lib/libnss_wrapper.so
2021-12-14T10:09:01.061436446Z	stdout	null
2021-12-14T10:09:01.061443106Z	stdout	------------------ update chromium-browser.init ------------------
2021-12-14T10:09:01.061462900Z	stdout	null
2021-12-14T10:09:01.061469786Z	stdout	... set window size 1280 x 1024 as chrome window size!
2021-12-14T10:09:01.061477428Z	stdout	null
2021-12-14T10:09:01.074174080Z	stdout	null
2021-12-14T10:09:01.074193303Z	stdout	------------------ change VNC password ------------------
2021-12-14T10:09:01.084485651Z	stdout	null
2021-12-14T10:09:01.084505906Z	stdout	------------------ start noVNC ----------------------------
2021-12-14T10:09:01.084525266Z	stdout	null
2021-12-14T10:09:01.084531571Z	stdout	------------------ start VNC server ------------------------
2021-12-14T10:09:01.084535625Z	stdout	remove old vnc locks to be a reattachable container

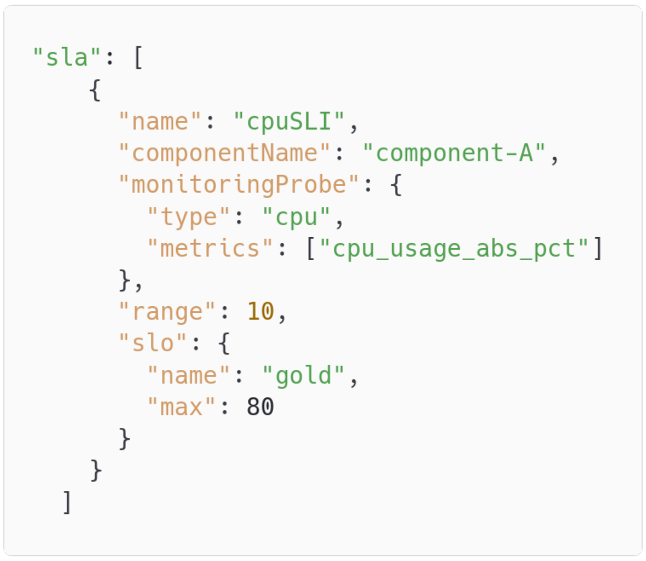
FIGURE 3-5 Logs visualization from the BASS' user interface

The automatic activation of resource monitoring and logs visualization was greatly appreciated by use case owners since it is extremely useful for the detection of anomalies, the identification of performance issues, and the assessment of the service health. Monitoring a Vertical Service requires the implementation or setup of several tools to collect, store, query, and visualize the metrics. The unified monitoring platform provided by the BASS already implements all of these things out-of-the-box, with sane defaults for the configuration and the application of best practices in order to avoid common mistakes. From the measurements during the trials, it has been observed that 30 to 40 MB of monitoring data are generated for a single component daily. Considering that a Vertical Service is usually composed of three or four components, we can estimate about 150 MB of monitoring data per service per day. This amount of data per service generates a storage pressure that can be easily handled in modern data centres, where storage solutions are designed for Big Data. Furthermore, the monitoring system allows for the configuration of data retention parameters with the possibility to aggregate or delete old data, so to save storage resources.

Furthermore, the active monitoring feature acts as a source of truth for the status of the resources managed by the BASS necessary for the SLA management feature. SLA enforcement engines get data from the monitoring system to check if any SLA violation is happening.

3.1.4. SLA Enforcement feature validation

The BASS allows defining a Service Level Agreement (SLA) for each Vertical Service. The SLA information is included in the VSD in the form of more atomic elements, such as Service Level Indicators (SLI) and Service Level Objectives (SLO). More details about the SLA model can be found in (D1.3 , 2021). Figure 3-6 shows an example of SLA definition in a VSD.



```
"sla": [
    {
      "name": "cpuSLI",
      "componentName": "component-A",
      "monitoringProbe": {
        "type": "cpu",
        "metrics": ["cpu_usage_abs_pct"]
      },
      "range": 10,
      "slo": {
        "name": "gold",
        "max": 80
      }
    }
  ]
```

FIGURE 3-6 Example of SLA definition in a VSD

The SLI defines the component and the metric to be monitored, as well as the range of time for sampling. The structure also contains an SLO, which defines the thresholds for the metrics. In the example in Figure 3-6, the maximum acceptable value for the CPU usage of "Component-A" is 80%.
Aside from the SLA definition, the BASS also provides methods to get SLI values, validate SLOs are respected, and execute corrective actions in case of SLA violations. The corrective actions allow for vertical (i.e., setting CPU and memory limits per component) and horizontal (i.e., increasing the number of replicas of a component) scaling.
All the methods are exposed by the BASS through an HTTP interface so that external clients can operate on the SLA. The latter are called SLA enforces and they implement algorithms in order to check the SLA of a Vertical Service for violations and request actions in order to correct it. Thanks to the generic interface, the enforcement logic can be trivial or more advanced, depending on the level of sophistication desired by the specific use case. An advanced example of an SLA enforcer is presented in Section 5.1.1.2 for the Digital Twin use case.
The features summarized above have been thoroughly validated in the BASS unit and integration test during the second year. They constitute the SLA Enforcement Framework, presented in more detail in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021).

3.1.5. Scalability measurements

The scalability of the BASS is defined as the trend of resource consumption under a constantly increasing amount of load. We also include in this definition the performance trend, measured as the elapsed time needed to deploy a service. In the context of the BASS, the load is intended as the number of Vertical Services currently deployed (i.e., active or in running state). Since each Vertical Service is composed of multiple components, using complex services may lead to different trends in the measurements. Hence, we selected a representative service for performing the scalability tests which is a copy of the Digital Twin service (see Section 5.1). During the tests, we deploy several replicas of the Vertical Service and collect the metrics aforementioned in order to observe a trend. For specifications about the hardware configuration for the experiments, please see Table 4-2.
To observe the effect of multiple Vertical Services on the CPU consumption of the BASS two series of tests have been executed. In the first, an increasing number of Vertical Services are deployed on the system sequentially. Starting from an initial state in which only one service is present, we deploy new services one by one up to a total of 10 services running at the same time. Table 3-2 reports a summary of the collected results.

TABLE 3-2 CPU usage results for 10 sequential deployments on the BASS

Vertical Services	Components (total)	Mean (%)	Standard deviation (%)	Minimum (%)	Maximum (%)
0	0	0,0753	0,0272	0,0501	0,2008
1	7	0,2521	0,1369	0,1256	0,5643
2	14	0,2448	0,1389	0,1758	0,5529
3	21	0,2134	0,1321	0,1885	0,5028
4	28	0,2951	0,1029	0,2137	0,5028
5	35	0,2773	0,1689	0,2261	0,6416
6	42	0,2882	0,1621	0,2514	0,6399
7	49	0,3275	0,1939	0,2768	0,7543
8	56	0,3517	0,1266	0,3016	0,6159
9	63	0,3656	0,1532	0,3270	0,7044
10	70	0,3898	0,1610	0,3639	0,8169

The first thing we can notice from the results in Table 3-2 is how little is the BASS consuming in general. The mean CPU utilization stays always below one percent (computed on all the available cores of the node where the BASS is deployed). In the raw data, we noticed a spike in the CPU consumption while deploying a service that later decreases to a lower and more stable value. Hence it is important to look at the "Maximum (%)"to evaluate the intensity of the spike. The reader can observe that the higher value recorded is 0,8169 %, corresponding to the last test case. We can confirm that the BASS has a very low impact in terms of resource utilization: a fair price considering the advanced features that it provides. Anyway, the Maximum column hints at an increasing trend in the CPU consumption in correspondence with more services deployed on the platform. Despite working with very low numbers, probably affected by measurement noise due to the operating system and the Java Virtual Machine, Figure 3-7 provides a plot of the maximum values.

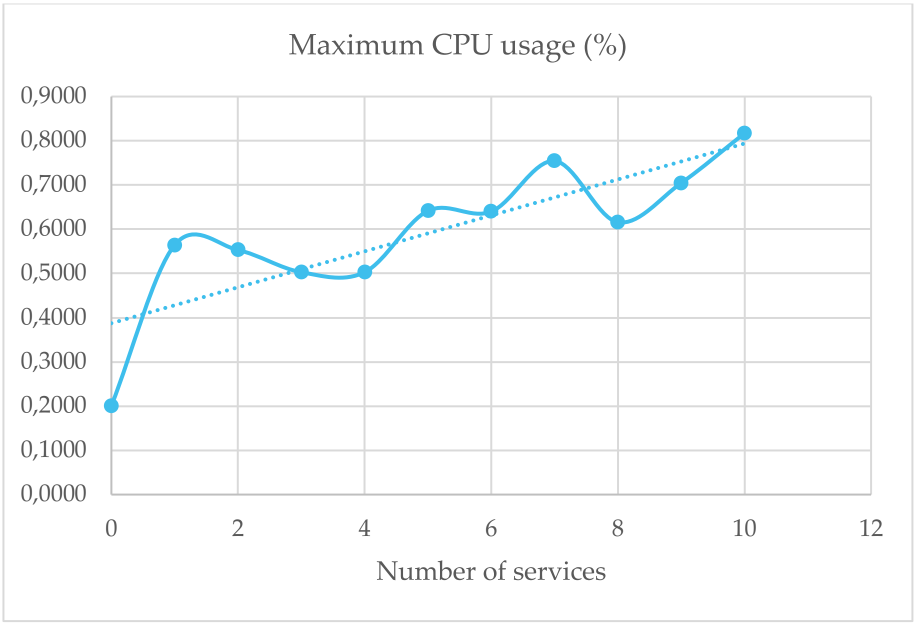


FIGURE 3-7 Trend visualization of the maximum CPU usage

As shown in the figure, the trend in CPU consumption is linear. We can conclude that the BASS scales well with respect to an increasing number of vertical services deployed on the platform.
The second series of tests has the goal to observe the effect of parallel deployments on the CPU consumption of the BASS. In this case, we send multiple deployment requests at the same time to a single instance of the BASS, starting from one and stepping up to 10. Table 3-3 reports a summary of the collected results.

TABLE 3-3 CPU usage results for parallel deployments on the BASS

Vertical Services	Components (total)	Mean (%)	Standard deviation (%)	Minimum (%)	Maximum (%)
1	7	0,2521	0,1369	0,1256	0,5643
2	14	0,2015	0,3833	0,1257	1,0565
4	28	0,3014	0,3720	0,1130	1,3581
6	42	0,3140	0,4069	0,1131	1,4845
8	56	0,3779	0,3500	0,1128	1,5436
10	70	0,3893	0,3353	0,1633	1,6198

We can observe how the mean CPU usage stays negligible, aligned with the results in Table 3-3. Anyway, looking at the “Maximum (%)” column, we can see higher spiking values even if they manage to stay under 2% of the total available CPU in the node. We can conclude that multiple deployment requests at the same time have some appreciable effect on the CPU consumption of the BASS but without constituting a risk for its normal operation. Figure 3-8 shows the trend visualization for maximum values in the case of parallel deployments. The trend appears to reach a plateau of around 1,60% of CPU utilization and so does not pose any issue for the scalability of the BASS.

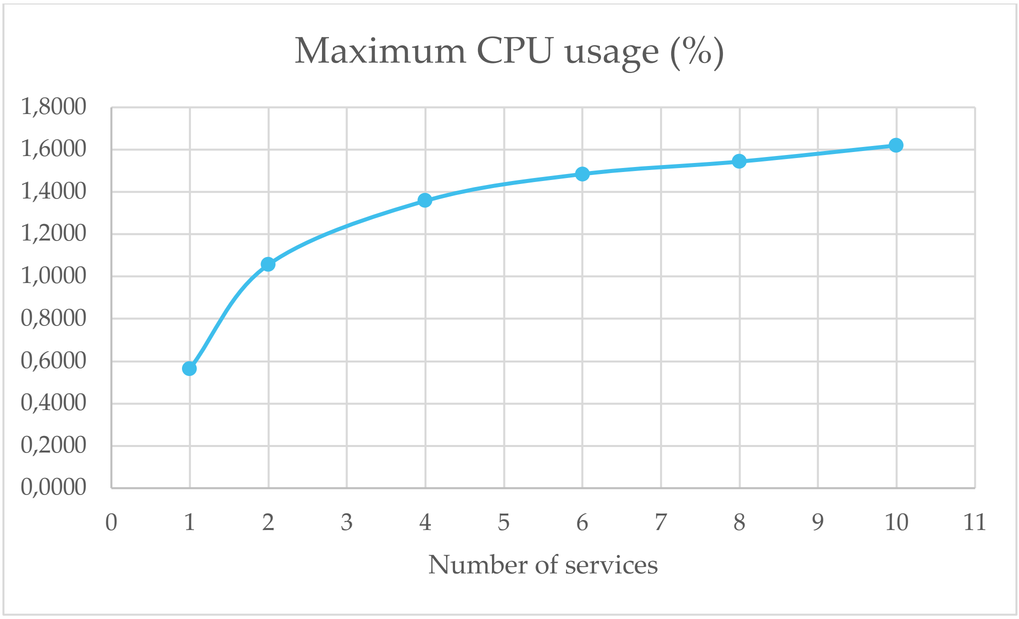


FIGURE 3-8 Trend visualization of the maximum CPU usage for parallel deployments

The two series of tests, in sequential and in parallel order, have been also used to measure the deployment time. Table 3-4 reports the measured deployment times for a series of ten sequential deployments on the BASS.

TABLE 3-4 Deployment time results for 10 sequential deployments on the BASS

Vertical Services	Components (total)	Deployment time (s)
1	7	56,13
2	14	33,08
3	21	28,36
4	28	33,10
5	35	30,37
6	42	29,54
7	49	28,78
8	56	30,60
9	63	29,37
10	70	31,61

Mean (s)	Standard deviation (s)	Minimum (s)	Maximum (s)
30,48	8,25	28,36	56,13

The first deployment taking a longer time is an expected result, since the BASS needs to transfer the container images to its local registry via the network. The subsequent deployments show a quite stable behaviour, with little variations in the elapsed time for the service to be completely running.
The results for deployment time in the series of tests with parallel deployments are reported in Table 3-5.

TABLE 3-5 Deployment time results for parallel deployments on the BASS

Vertical Services	Components (total)	Mean (s)	Standard deviation (s)	Minimum (s)	Maximum (s)
1	7	30,4800	0	30,4800	30,4800
2	14	31,4995	4,1019	28,5990	34,4000
4	28	49,6045	18,5656	32,9090	66,5160
6	42	64,2430	17,2051	32,8550	70,0110
8	56	69,9720	32,6310	33,6110	124,4660
10	70	71,5195	39,9318	38,3950	135,3720

The arrival of multiple requests at the same time affects significantly the deployment time. As we can see by observing the "Minimum (s)" and "Maximum (s)" columns, the BASS manages to parallelize 2 deployments requested at the same time and later on it starts queueing the requests. The plot of the maximum values, shown in Figure 3-9, confirms this observation.

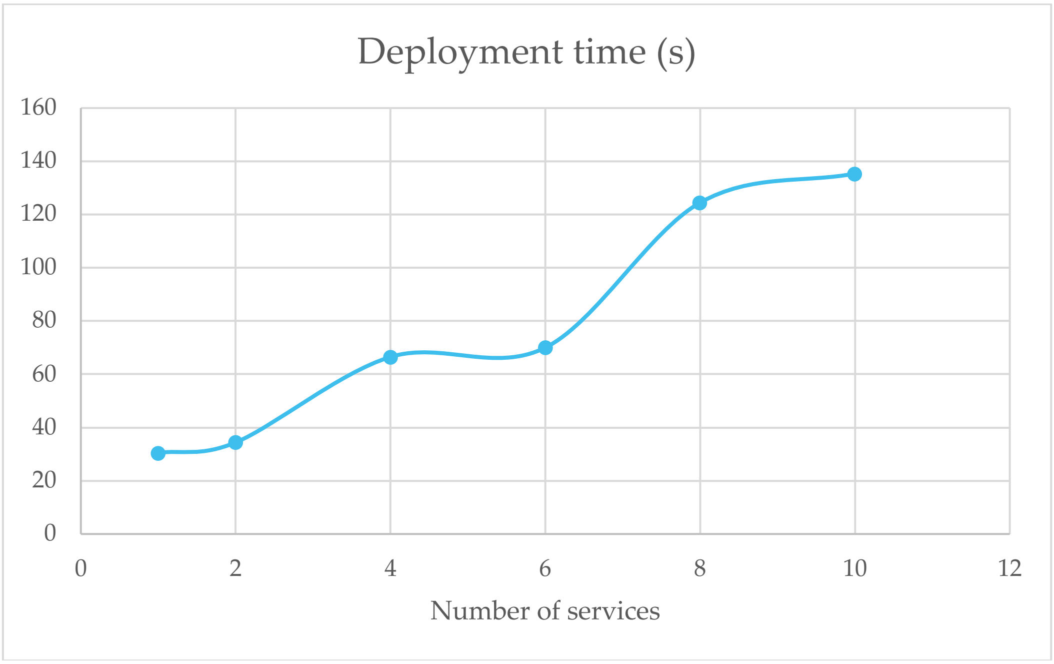
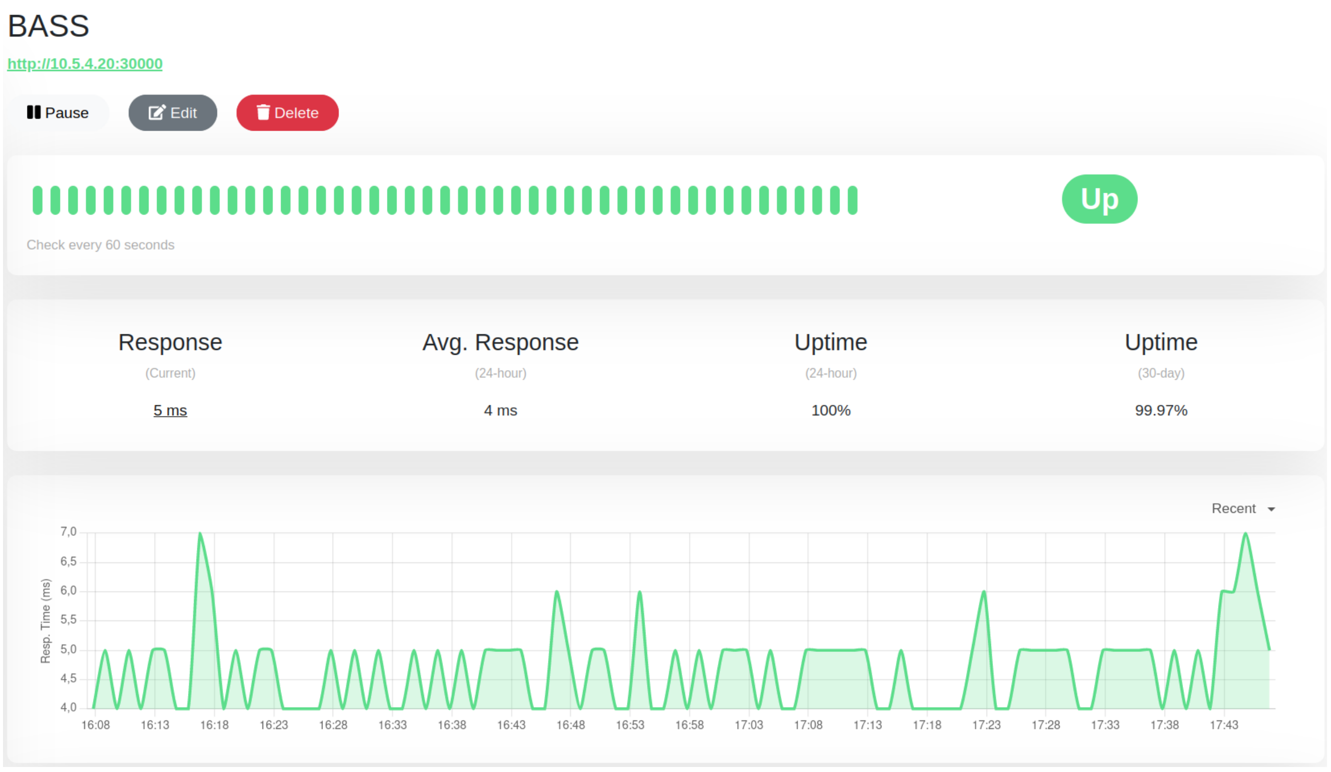


FIGURE 3-9 Deployment times for parallel deployments

The cause for the increase in steps of two is unknown. Further investigation and tests on different hardware configurations are required. While the discussed behaviour may look alarming for the scalability of the system, it is also true that the probability of receiving parallel requests for deployments decreases with respect to their amount. It is quite unlikely for ten deployment requests to arrive at the same time, meaning the same second. Despite the performance degradation, the BASS demonstrated to react without errors to unusual load. Further tests, with more realistic arrival distributions (e.g., Poisson) can provide a more representative behaviour of a production-like scenario.
Finally, for what concerns memory consumption, we state that it remained constant throughout all the tests we performed. Being the BASS developed in Java, we monitored Heap and non-Heap memory. The only, small, variations we observed were due to the intervention of the garbage collector.

3.1.6. Availability and Reliability

Availability and reliability are KPIs that aim at validating the stability of the BASS and they are measured on a relatively long period of time of one month. Availability is defined as the probability that the BASS is operating properly when it is requested for use. We use a self-hosted uptime monitoring service, Kuma Uptime [5], in order to periodically probe the BASS Northbound Interface. During the long-term monitoring period, we measured availability of 99.97% for the BASS. The result is acceptable for a setup in an experimental environment. Figure 3-10 shows the results visualized in the uptime monitoring service user interface.



Response (Current)	Avg. Response (24-hour)	Uptime (24-hour)	Uptime (30-day)
5 ms	4 ms	100%	99.97%

FIGURE 3-10 Uptime monitor for the BASS

Anyway, this availability measurement does not give any information about the capacity of the BASS to correctly perform its functions, e.g., deploy a Vertical Service. Hence, a small software tool has been developed to deploy and subsequently stop a Vertical Service on the BASS, while validating that the deployment was successful. The routine is executed twice a day. The collected results show a success rate of 100%.
Reliability is defined as the probability that the BASS will produce correct outputs up to some given time. To measure reliability, we use again the aforementioned uptime monitoring service that measures also the response time for requests to the BASS' Northbound Interface. Setting a threshold of 100 ms for the response time (about the limit for having the user feel that the system is reacting instantaneously (Nielsen, 1993)) we measured reliability of 100% for the BASS.

3.2. DASS Validation results

The DASS functional and non-functional features have been validated for most of the use cases of the project during the field trials. The use cases have adopted it to perform data preprocessing, storage, as well as data dispatching, which are the main functionalities of the DASS.

For the DASS to support a wide heterogeneity of scenarios, networks, and devices, we adopt a two-level protocol design the Zenoh API and the Zenoh-net API. The data pre-processing and the data storage components are implemented by the Zenoh layer. The Zenoh layer is a higher-level API providing the same abstractions as the Zenoh-net API in a simpler and more data-centric oriented manner as well as providing all the building blocks to create distributed storage. The Zenoh layer is aware of the data content and can apply content-based filtering and transcoding.

The Zenoh-net layer is a network-oriented API providing the key primitives to allow pub/sub (push) communications as well as query/reply (pull) communications. The Zenoh-net layer focuses on data transportation and is agnostic about data content nor storing data. The Zenoh-net layer implements the data-dispatching functionality.

Besides the functional validation of the DASS in the context of the use cases, the DASS has been evaluated in terms of scalability and performance. The scalability test was a mesh routing at scale test and was performed at the Zenoh-net level. The performance tests were carried out in peer-to-peer and in brokered communication mode, in both the Zenoh API, and Zenoh-net layer.

3.2.1. Scalability measurements

Regarding scalability, we performed a mesh routing at scale test at the Zenoh-net layer. The Zenoh-net layer was tested on three scenarios: initial setup, router failure, and router insertion. During each of these scenarios, it was collected the data regarding the number of link-state message, link-state bandwidth (Kb), link-state alignment time (ms), network trees computation time (ms), routes computation time (ms) and total alignment time (ms). The total set of mentioned parameters were calculated for a network graph of 50 nodes, and a graph of 100 nodes. The results are presented in the following Table 3-6:

TABLE 3-6 Mesh routing at scale test results

	Graph 50 nodes, 99 edges			Graph 100 nodes, 121 edges		
	Initial setup	Router failure	Router insertion	Initial setup	Router failure	Router insertion
Linkstate msgs	13388	567	1841	13544	659	1383
Linkstate bandwidth (Kb)	918	20	91	1527	23	103
Linkstate alignment time (ms)	948	222	34	3864	251	66
Trees computation time (ms)	102	100	92	111	122	112
Routes computation time (ms)	44	78	81	71	596	630
Total alignment time (ms)	1094	400	207	4046	969	808

3.2.2. Performance measurements

Regarding performance, during the last year of the project, there have been some design improvement in the DASS implementation with regards to isolating the async code in specific parts of the code, especially the one interacting with the network, and moving some other parts to the standard sync library. As a result, Eclipse Zenoh has now a very balanced mix of sync and async code that takes the best of both worlds. This design improvement allowed us to drastically reduce the stack size of some critical async futures which immediately reflected in a performance boost as described below in terms of throughput and latency in two different scenarios: peer-to-peer and brokered communication.

Peer-to-peer communication: throughput

The peer-to-peer communication scenario consists of two Zenoh applications that can directly communicate with each other without the need for any infrastructure component through a 100GbE connection, see Figure 3-11.

Peer
100GbE
Peer

FIGURE 3-11 Peer-to-peer communication scenario

Figure 3-12 A) presents in the y-axis the number of messages per second, and in the x-axis, the payload of the messages ranging from 8 bytes to 1 GB. The results are shown in a log scale, in terms of the number of messages per second. As we can see, Zenoh-net API delivers more than 3.5M msg/s with an 8 bytes payload. At the same time, Zenoh API delivers 2M msg/s with the same 8 bytes payload.
Figure 3-12 B) presents in the y-axis the bits per second, and in the x-axis, the payload of the messages ranging from 8 bytes to 1 GB. The results are shown in log scale, in terms of throughput (bit/s) delivered at API level. We also report the throughput obtained with iperf on the same 100GbE connection as reference baseline: 60 Gb/s. As it can be seen, a 100 Mb/s connection is already saturated by Zenoh-net and Zenoh with a payload as little as 8 bytes. A 1 Gb/s connection is then saturated with a payload of 32 and 64 bytes for Zenoh-net and Zenoh, respectively. A payload of 512 and 1024 bytes is then sufficient for Zenoh-net and Zenoh to saturate a 10 Gb/s connection. Finally, payloads larger than 128 KB suffice to saturate a 40 Gb/s connection., this is a good indicator because it shows that the data transmission protocol makes use of most of the existing bandwidth in the infrastructure, even with small payloads.

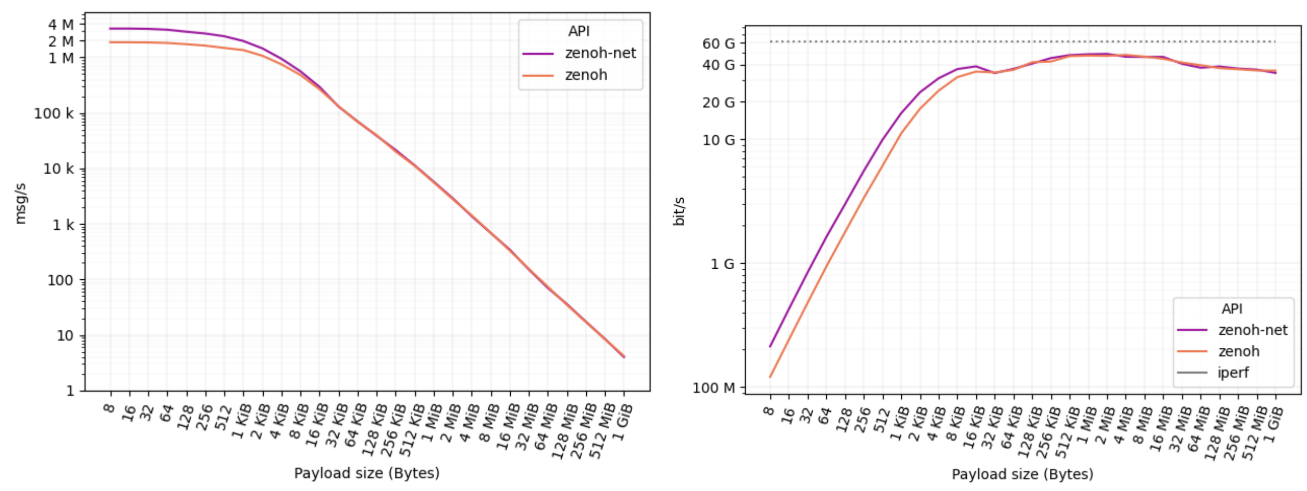


FIGURE 3-12 a) Eclipse Zenoh API messages/seconds in log scale, b) Eclipse Zenoh API bits/seconds in log scale

Peer-to-peer communication: latency

Latency is the time it takes for data to pass from one point on a network to another. As you can see from Figure 3-13, in the y-axis we have the time in micro-seconds (μs), and in the x-axis the number of messages per second, as the number of messages per second increases, latency decreases. This brings us to the conclusion that latency depends heavily on the load of the system because when messages are sent at a low rate, the processes are more likely to be descheduled by the operating system. This operation adds additional latency since the processes need to be rescheduled when messages are sent and received. This is true for both Zenoh and the classical ping, which is reported as a reference baseline for latency.

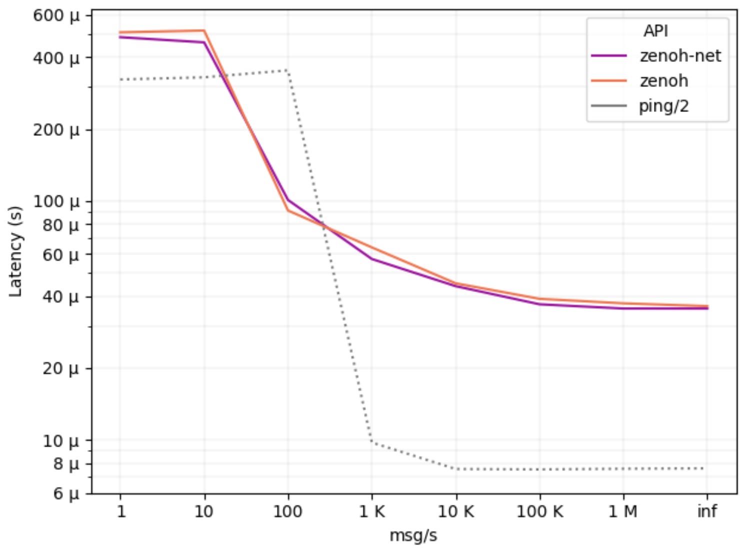


FIGURE 3-13 Eclipse Zenoh API latency on P2P communication

The x-axis of Figure 3-13 shows the number of messages that we configured to be sent in one second, from 1 to 1 million and beyond. The **inf** case represents the scenario where messages are sent back-to-back as fast as possible. In such a backlogged scenario, we can see that the latency is as little as 35 μsec for both Zenoh-net and Zenoh APIs. The payload size was constant for this test, and it was 64 bytes, the same as standard ICMP.

Brokered communication scenario: throughput

The brokered communications scenario happens when a Zenoh instance performs as a message broker and serves for message validation, transformation, and routing. It mediates communication among applications, minimizing the mutual awareness that participants should have of each other to be able to exchange messages, effectively implementing decoupling. In this test, one workstation runs the publisher, a second one runs the Zenoh router and a third one runs the subscriber. All workstations are connected through a 100GbE connection, see Figure 3-14.

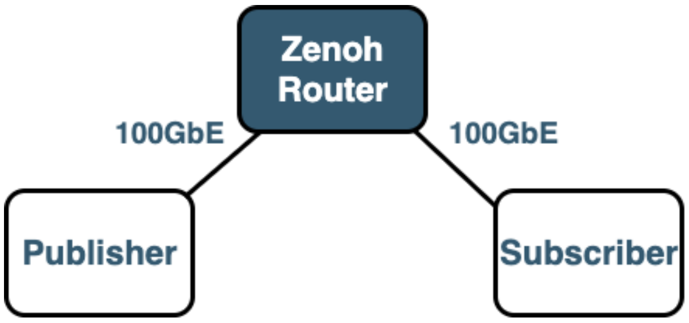


FIGURE 3-14 Brokered communication scenario

Figure 3-15 A) presents in the y-axis the number of messages per second, and in the x-axis, we have the payload of the messages ranging from 8 bytes to 1 GB. The results are shown in the log scale. We can observe that Zenoh-net API delivers 3M msg/s with an 8 bytes payload. At the same time, Zenoh API delivers 1.8M msg/s. Figure 3-15 B presents in log scale, the same results in terms of throughput (bit/s) delivered at API level, in the y-axis the bits per second, and in the x-axis, the payload of the messages ranging from 8 bytes to 1 GB. As it can be noticed from the results, a 100 Mb/s connection is still saturated by Zenoh-net and Zenoh with a payload as little as 8 bytes. A 1 Gb/s connection is then saturated with a payload of 64 bytes for Zenoh-net and Zenoh. A payload of 1024 bytes is then sufficient for both Zenoh-net and Zenoh to saturate a 10 Gb/s connection. Finally, larger payloads are forwarded at 20-30 Gb/s.

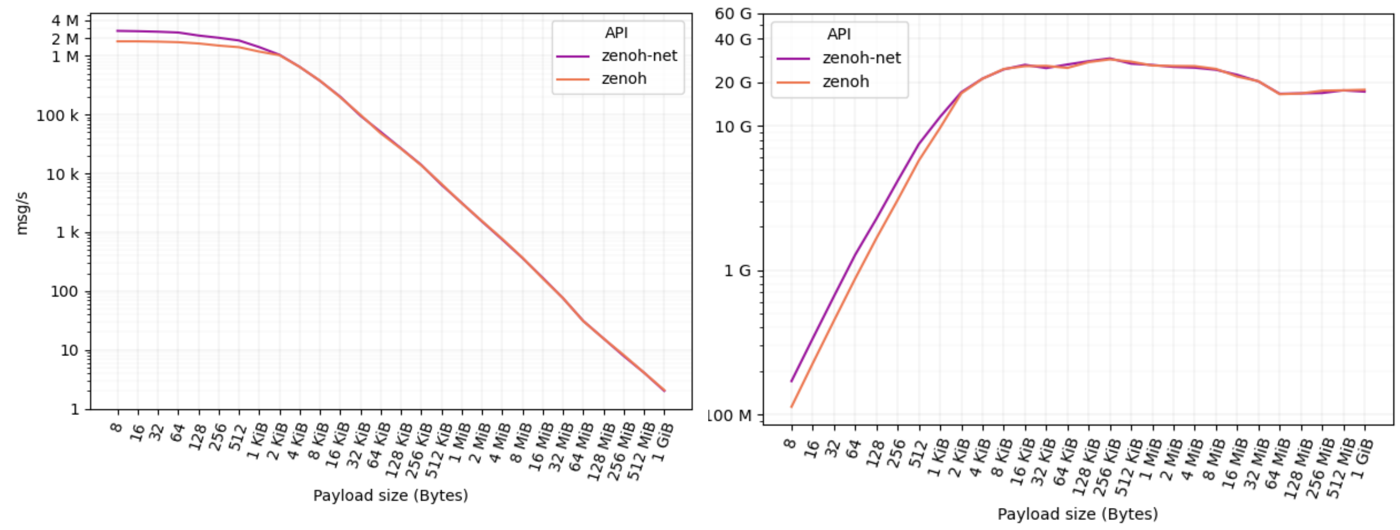


FIGURE 3-15 a) Eclipse Zenoh API messages/seconds in log scale, b) Eclipse Zenoh API bits/seconds in log scale

Brokered communication: latency

In the brokered communication test, we considered two clients that communicate with each other through a Zenoh router as an intermediate node. In the brokered test, as you can see in Figure 3-16, latency is double than the peer-to-peer test: 70 µs. This is due to the fact that an additional network hop, i.e. the router, has been introduced between the two clients. Nevertheless, it can be noticed that the router does not add any noticeable latency to the overall communication, being the latency is driven mainly by the number of hops. The message payload size remains constant at 64 bytes.

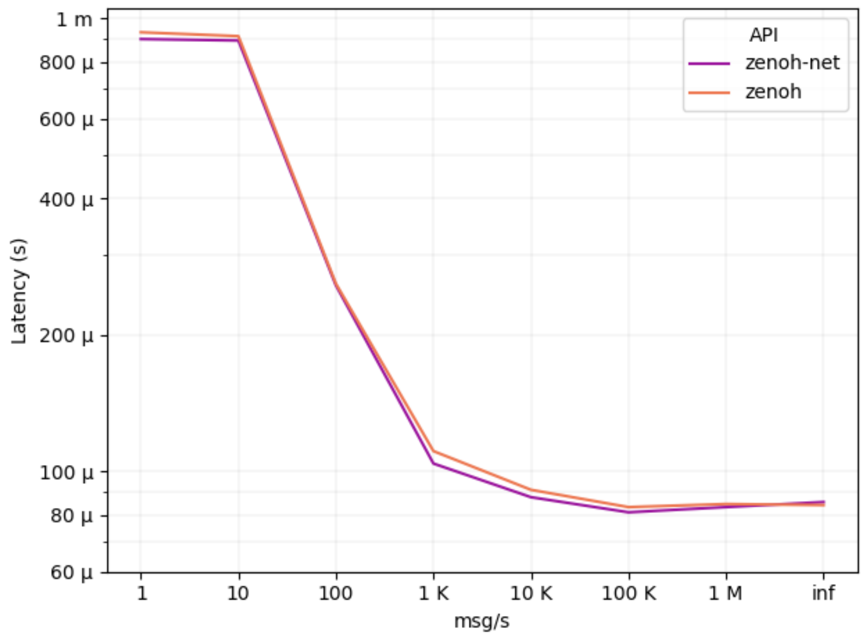


FIGURE 3-16 Eclipse Zenoh API latency on brokered communication

To summarize, recent work makes Zenoh capable to deliver over 3.5M msg/s for small messages, over 45 Gb/s for large messages, and latency as little as 35 μsec in the peer-to-peer scenario. These experiments are relevant as all the use cases require high throughput and lowering latency is an important part of building a good user experience.

3.3. IESS Validation results

The IESS functional and non-functional features have been validated for most of the use cases of the project during the field trials. The IESS offers features to facilitate the adoption of AI and ML functionalities, like automatic management of the model training operations, automatic packaging and deployment of the inference applications, and storage of pre-trained models in a catalogue. The general IESS performance has also been evaluated in terms of scalability, availability and reliability.

3.3.1. Model training feature validation

The IESS automatically defines a model training pipeline that includes resource discovery and allocation, deployment of containers for the training, storage of trained models and other software artifacts in a catalogue. The IESS does not actually manage any resources, but it interacts with the BASS that is the component in the DEEP platform responsible for resource management and orchestration.
During the trials at 5TONIC premises, the training feature was validated with the Digital Twin use case (see Section 5.1). The vertical provided a container image including the code of the algorithm performing the training (based on statsmodel (statsmodel, n.d.)) and the dataset. The training task is declared as an AI Component in the use case's Vertical Service Descriptor. The BASS recognizes these kinds of components and it forwards all of them to the IESS. Figure 5-5 shows the training of the movement prediction intelligence engine in the BASS Web User Interface. The other components of the service are kept in a WAIT TRAINING state while the training of the model is in progress. Indeed, the BASS waits for an IESS notification before proceeding with the deployment of the Vertical Service.

3.3.2. Inference app packaging and deployment feature validation

A usual problem in machine learning is how to serve prediction results from a trained model. In fact, trained models can be exported to different binary formats but to be used in order to obtain predictions, they need to be imported into other applications through the development of additional code. The IESS provides an automated packaging procedure that, given a trained model as input, generates a web application with a well-defined and well-documented HTTP REST interface to request and obtain predictions. The aforementioned application is then deployed by the BASS as a regular component of the Vertical Service, so that other components have connectivity with the HTTP interface and can all request and obtain predictions.
During the trials at 5TONIC premises, the packaging and deployment feature for intelligence engines was validated for the Digital Twin use case. After the training phase is complete (see Figure [above]), the component used for training is stopped and removed. The IESS packages the Movement Prediction trained model into a container image and uploads it into the catalog. When the image is available, the BASS finally starts the deployment of the whole Vertical Service. Figure 5-6 shows the Digital Twin vertical service successfully deployed with the Movement Prediction intelligence engine. Being the intelligence engine a regular component of the vertical service, it benefits of all the features available in the BASS, like optimized placement in the cloud, edge, or fog depending on where the predictions are requested or monitoring as discussed in Section 3.1.3.

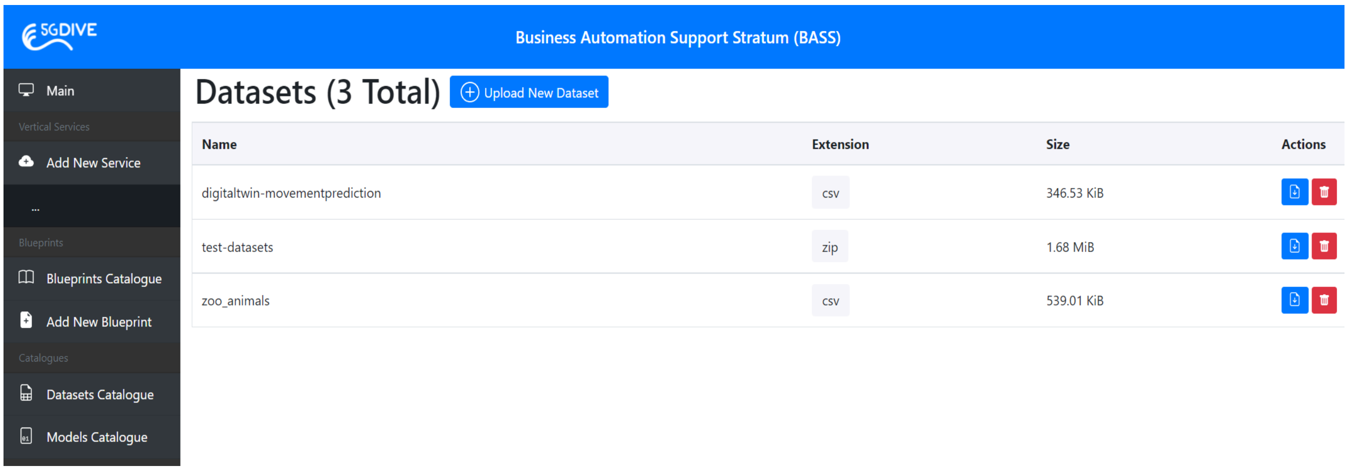
3.3.3. IESS Catalogue feature validation

The IESS catalogue can store metadata about supported Machine Learning platforms and algorithms, binary artifacts like trained models, and container images for inference applications. The catalogue has already been validated in the previous Sections 3.3.1 and 3.3.2) since it acts as a supporting component for all the other operations.

Another use for the IESS catalog is the possibility for the Vertical to manually upload a pre-trained model. This can be motivated by the requirement of specific hardware for the model training (i.e., GPU) that is not available in the regions managed by the BASS, or by a preference for an offline training procedure in order for data scientists to manually verify the accuracy of the model, or finally by the preference for a training in a cloud environment with lots of computing resources.

If the pre-trained model is uploaded to the IESS catalogue, at the moment of deploying the service, the IESS retrieves the model, stores it in a static volume, and mounts the volume into the component that declares the requirement for the model. With this procedure, the Vertical does not need to include the model into its component and has the possibility to use a different model for each deployment of the use case.

Finally, the IESS catalogue can store also datasets for the training and testing of models. They can be retrieved automatically at deployment time. Figure 3-17 shows the web view for the IESS Datasets Catalogue, integrated in the BASS User Interface. The vertical can review the available datasets and upload new ones. The web view for the IESS Models Catalogue is similar.



Datasets (3 Total)

Name	Extension	Size	Actions
digitaltwin-movementprediction	csv	346.53 KiB	
test-datasets	zip	1.68 MiB	
zoo_animals	csv	539.01 KiB	

FIGURE 3-17 IESS catalogue view in BASS User Interface

3.3.4. Offer AutoML service for easy AI adoption

Apart from taking care of the training-packaging-deployment pipeline, the IESS also offers AutoML services. With this feature, the Vertical does not need to select or develop a machine learning algorithm in order to produce a trained model for its problem. The only thing needed is a declarative description of the problem, in the form of a component in the VSD as shown in Figure 3-18.

```
{
  "AI": "true",
  "name": "animal-classifier",
  "aiengine": "automl",
  "aitype": "classification",
  "dataset": "<url/to/dataset>",
  "columnPredict": "class_type"
}
```

FIGURE 3-18 Component descriptor for AutoML with IESS

The problem is described by three main fields:

- *aitype*: describing the kind of problem to solve, can be "classification" or "regression"
- *dataset*: a link where to retrieve the dataset for training
- *columnPredict*: the goal of the prediction, can be a set of classes (classification problem) or a series of real numbers (regression problem)

The IESS leverages the H2O.ai (H2o.ai, n.d.) platform for training and subsequently, the generated model follows the pipeline, passing through the packaging and deployment phases.
The AutoML feature has been tested and validated through several demos presented in webinars and workshops. It has attracted great interest from the audience. For simple problems, like Zoo Animal Classification from Kaggle (Kaggle), the trained models achieve an accuracy of around 99%. While not being a complete replacement for custom algorithms, AutoML can result very useful for inexperienced verticals allowing them to easily integrate machine learning features in their services.

3.3.5. Scalability

The scalability of the IESS is defined as the trend of resource consumption under a constantly increasing amount of load. We also include in this definition the performance trend, measured as the elapsed time needed to train and deploy an AI component. In the context of the IESS, the load is measured as the number of simultaneous requests arriving at the same time. We selected a representative intelligence engine for performing the scalability tests which is the Movement Prediction engine of the Digital Twin use case. With respect to the tests for the scalability of the BASS, we only observe the effect of parallel requests for the IESS. Indeed, while the BASS deploys services that stay running for long periods, the IESS creates jobs (i.e., the training-packaging-deployment pipeline) that terminate after their completion. For specifications about the hardware configuration for the experiments, please see Table 4-2.
During the tests, we send multiple requests to the IESS, starting from one and stepping up to 6. Table 3-7 reports a summary of the collected results for the CPU usage. During the tests, we send multiple requests to the IESS, starting from one and stepping up to 6. Table 3-7 reports a summary of the collected results for the CPU usage.

TABLE 3-7 CPU usage results for multiple requests on the IESS

Requests for intelligence engines	Mean (%)	Standard deviation (%)	Minimum (%)	Maximum (%)
1	0,0194	0,5679	0,0126	1,5855
2	0,0189	1,5231	0,0168	5,7335
4	0,0188	2,0362	0,0125	9,9859
6	0,0189	2,3920	0,0125	16,4318

On average the CPU usage stays on very low values. This is because the IESS delegates the execution of the training phase to the BASS. Indeed the BASS can manage large clusters of resources and it is worth leveraging its functions in order to allocate the proper resources for training (e.g., GPU devices) and manage the lifecycle of the relevant components. Anyway, we can see significant peak CPU usage by the IESS by observing the values reported in the "Maximum (%)" column of Table 3-7. By analyzing the raw experimental results, we identified the cause of the peak in the packaging phase. We can observe the trend for the Maximum CPU usage in Figure 3-19.

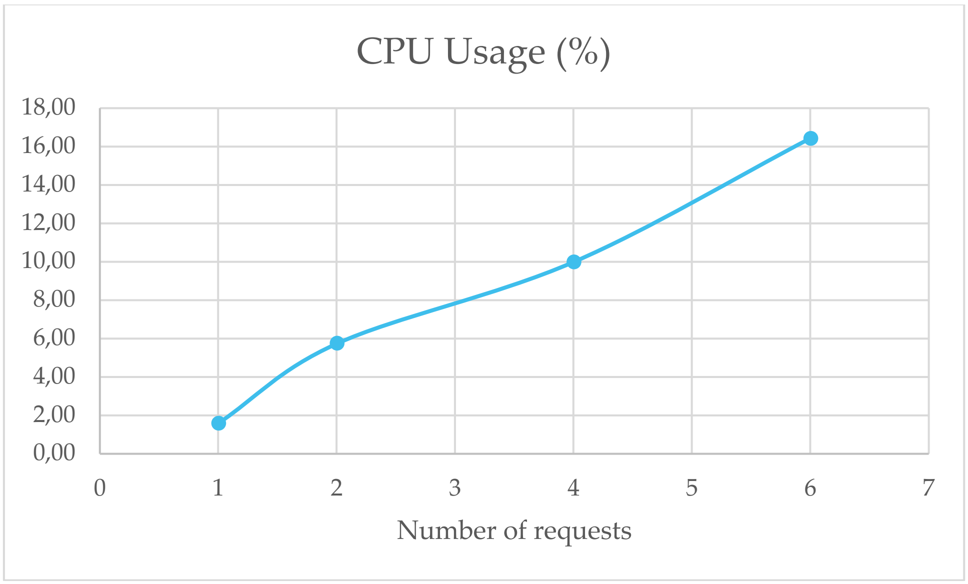


FIGURE 3-19 Trend visualization for the maximum CPU usage

Despite the significant values reported, the trend of the CPU usage is linear, suggesting acceptable scalability for the IESS. Furthermore, during the trials at 5Tonic premises, the IESS was given a limited amount of resources. On more powerful hardware, a more representative target for a production-like configuration, we expect the impact of the packaging phase to be less significant. Table 3-8 reports the total elapsed time for executing the training-packaging-deployment pipeline against an increasing number of parallel requests arriving at the IESS.

TABLE 3-8 Elapsed time for multiple requests arriving at the IESS

Requests for intelligent engines	Deployment time (s)	Deployment time (m)
1	242	4m 2s
2	297	4m 57s
4	502	8m 22s
6	627	10m 27s

The majority of the elapsed time for serving a request for an intelligent engine is spent in the training phase. This is expected as it is well known that training of machine learning models can take a long time (i.e., from minutes to hours) depending on the algorithm and the size of the dataset. Figure 3-20shows the trend for the deployment time.

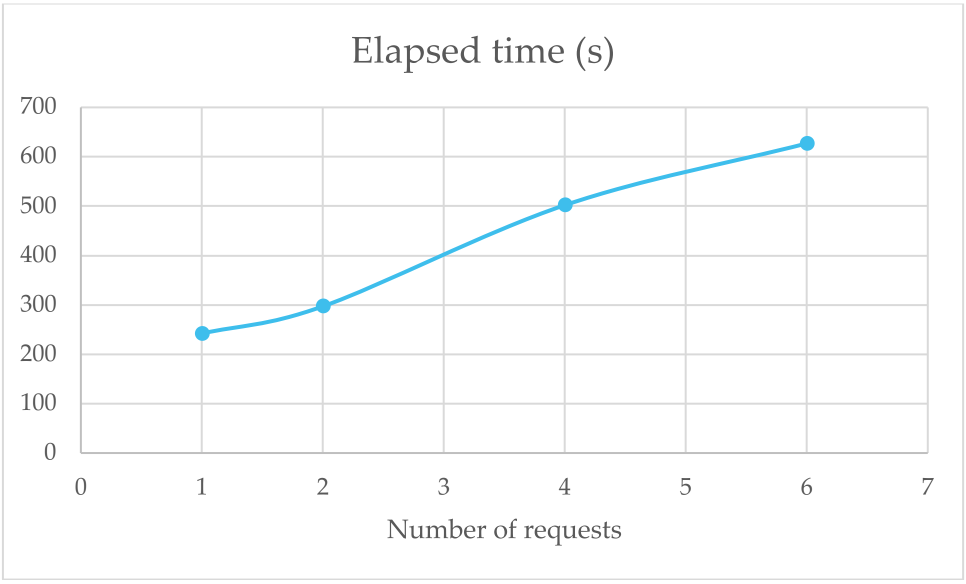


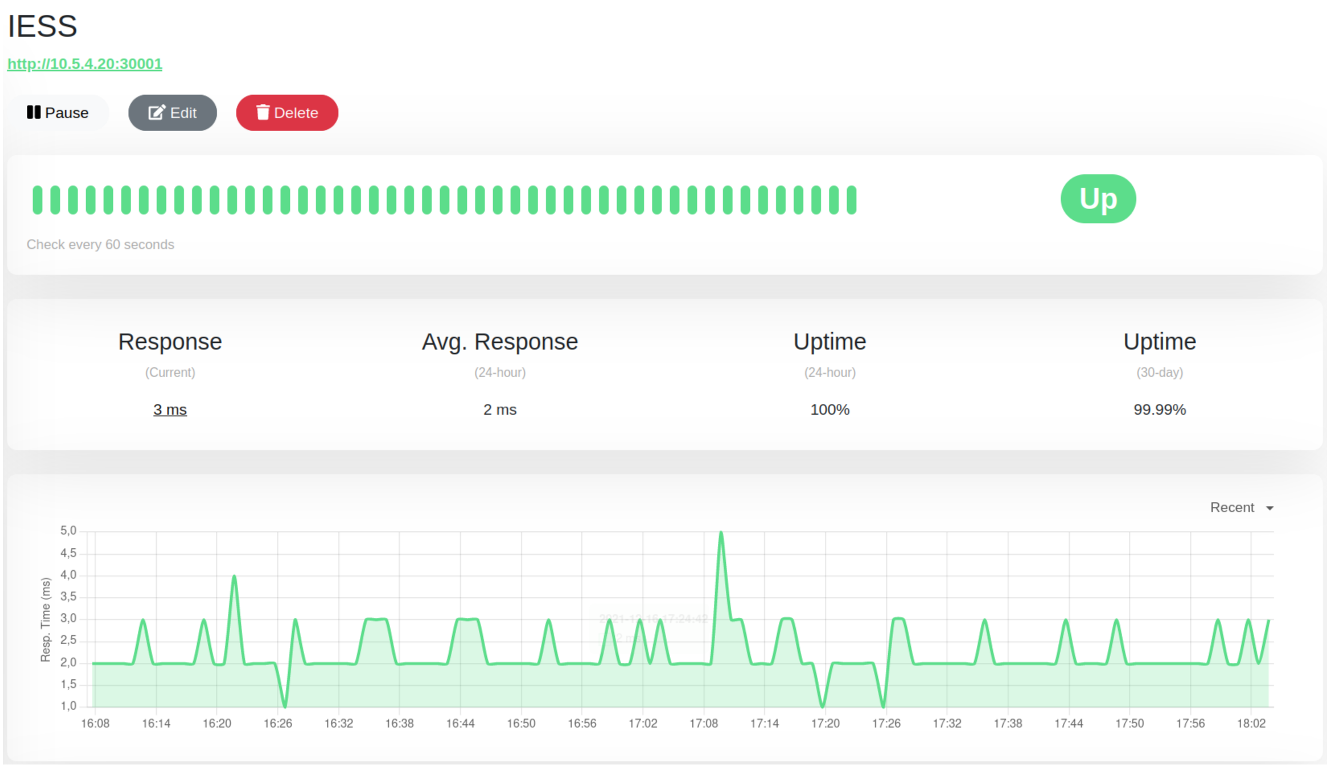
FIGURE 3-20 Elapsed time for multiple requests arriving at the IESS

The elapsed time grows almost linearly with respect to the number of requests arriving at the same time at the IESS. The experiments demonstrate that the IESS provides good scalability in terms of both resource usage and requests' execution time. Furthermore, despite the unusual load, the IESS reacted without errors and always provided a correct outcome.

Finally, for what concerns memory consumption, we state that it remained constant throughout all the tests we performed. Being the IESS developed in Java, we monitored Heap and non-Heap memory. The only, small, variations we observed were due to the intervention of the garbage collector

3.3.6. Availability and Reliability

Availability and reliability are KPIs that aim at validating the stability of the IESS and they are measured on a relatively long period of time of one month. Availability is defined as the probability that the IESS is operating properly when it is requested for use. We use a self-hosted uptime monitoring service, Kuma Uptime [5], in order to periodically probe the IESS Northbound Interface. During the long-term monitoring period, we measured availability of 99.99% for the IESS. The result is acceptable for a setup in an experimental environment. Figure 3-21 shows the results visualized in the uptime monitoring service user interface.



Response	Avg. Response	Uptime	Uptime
(Current)	(24-hour)	(24-hour)	(30-day)
3 ms	2 ms	100%	99.99%

FIGURE 3-21 Uptime monitor for the IESS

Anyway, this availability measurement does not give any information about the capacity of the IESS to correctly perform its functions, e.g., train, package, and deploy an intelligence engine. Hence, a small software tool has been developed to periodically send a request for an intelligence engine on the IESS and validate that the job terminates successfully. The routine is executed twice a day. The collected results show a success rate of 100%.
Reliability is defined as the probability that the IESS will produce correct outputs up to some given time. To measure reliability, we use again the aforementioned uptime monitoring service that measures also the response time for requests to the IESS' Northbound Interface. Setting a threshold of 100 ms for the response time (about the limit for having the user feel that the system is reacting instantaneously (Nielsen, 1993)) we measured reliability of 100% for the IESS.

4. Final Integration

This section provides, an update on the trial sites of I4.0 trials in Spain and ADS trials in Taiwan. Then the final use-case integration is presented explaining how all 5G-DIVE use cases in trials are integrated with one common DEEP platform. In the previous D3.1 (D3.1: Definition and setup of vertical trial sites, 2020)and D3.2 (D3.2, 2021), we provided the initial information regarding the trial sites of I4.0 and ADS. The ADS trial site is unchanged from the one described in D3.2 [2]. Basically, it has two locations for ADS uses cases. For ADS-UC1, the trial site is located at the football field of ITRI campus. This trial site is suitable for developing drone collision avoidance-related applications. For ADS-UC2, the trial is located at the Microelectronics and Information Systems Research Center (MIRC) building premises. The MIRC building is an 8-story building located inside of NCTU campus. This trial site is suitable for developing drone disaster relief missions.

4.1. Trial setup premises and hardware resources

In this section, the details of the final trial site of I4.0 will be elaborated. This final test trials were carried out in the 5TONIC premises. Figure 4-1 presents the plan of the 5TONIC trial site. The 5TONIC DPC, in the upper part of the image, hosts the edge side of the use cases located at 5TONIC premises. The 5TONIC Industrial Experiments area, 92 sqm big, was dedicated to hosting the fog side and hardware devices of the use cases. Here, the devices are connected to the Ericsson 5GC, with the UPF server placed in 5TONIC, through the Ericsson Radio Dot indoor antenna, which guarantees coverage for half of the room. The UPF is connected to both the Ericsson 5GC located in Sweden and the servers in the 5TONIC DPC datacentre, where the 5G-DIVE Edge infrastructure is situated.

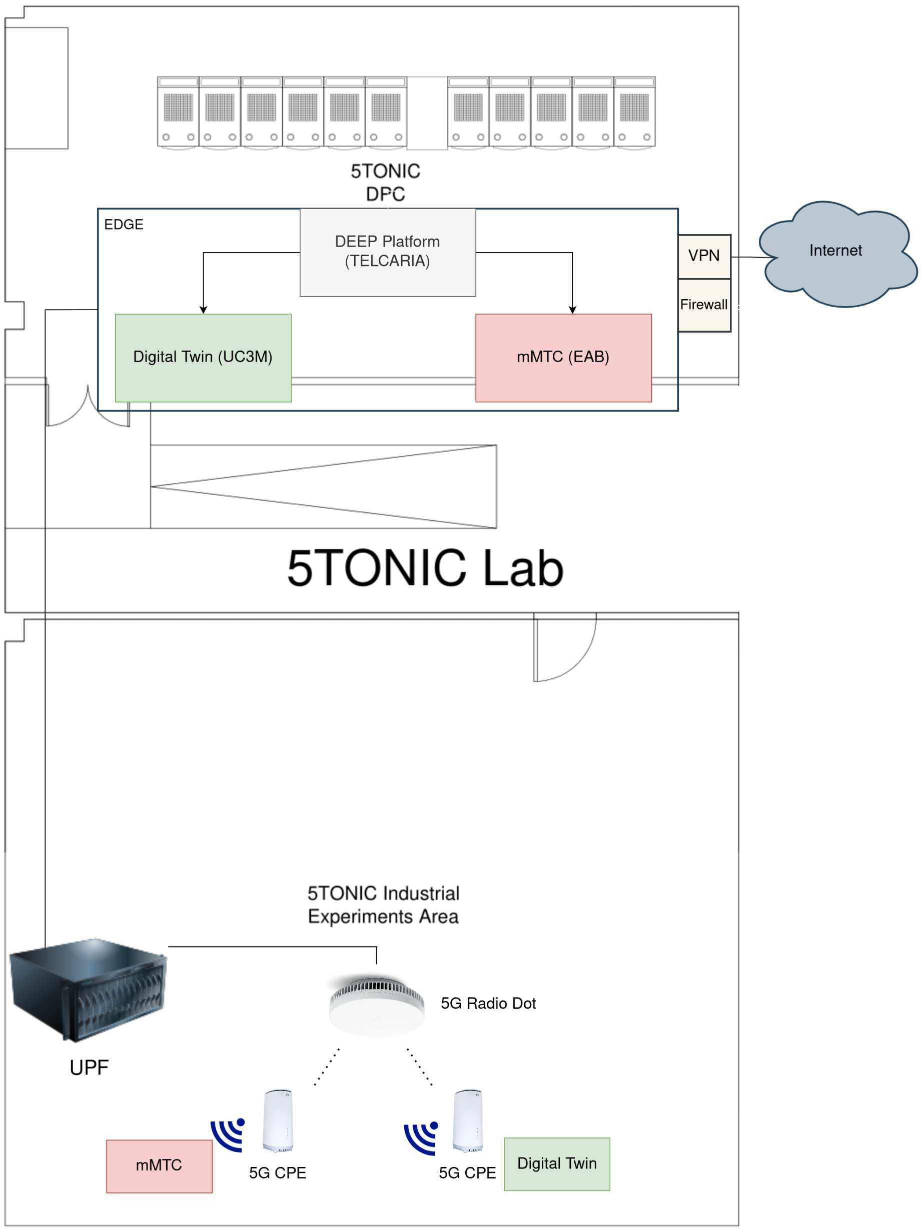


FIGURE 4-1 Plan of the 5TONIC Trial site

The following Table 4-1 presents the list of the 5TONIC servers used for the deployment of the DEEP Platform and the Industry 4.0 use cases. In particular:

TABLE 4-1 List of servers available in 5TONIC trial site used for the deployment

Component	Details
R430 v3:	Dell PowerEdge R430
	processor: Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz
	memory: 16Gb DDR4 2133 Mhz
	disc : 1TB (raid 1)
	networking: 2x 10Gb (X540-AT2), 4x1Gb (NetXtreme BCM5720) <- Ethernet
R630:	Dell PowerEdge R630
	processor: Intel Xeon E5-2620 v4 processors (16 cores and 32 threads in total)
	memory: 128 GB of RAM
	disk: 1TB
	networking: 2x X550 10Gbps
	networking: 4xI350 1Gbps network interfaces
R630 dual:	Dell PowerEdge R630
	processor: 2x Intel Xeon E5-2620 v4 processors (16 cores and 32 threads in total)
	memory: 128 GB of RAM
	disk: 1TB
	networking: 4xX520 10Gbps network interfaces
	networking: 4xI350 1Gbps network interfaces

Eventually, we had the I4.0 use cases present in 5TONIC+ADS fully integrated with the BASS (and running in the 5G CPE). So basically all containers are present at the BASS, can be instantiated and deployed using the BASS, and telemetry out of them is being received. In addition, the intelligence engines are been integrated into the IESS. So we have intelligence engines selected based on a template, automatic training, and deployment of the trained intelligent engine. For example, the DT movement prediction has been integrated following this idea.

4.2. Configuration for DEEP platform and use-cases

For the sake of the trials, hosted both at 5TONIC and remotely connected vertical premises, we created a scenario with four distinct regions (see Section 3.1.1) managed and orchestrated by a single deployment of the DEEP platform. A graphical representation of the setup is presented in Figure 4-2, with each region in a different colour.

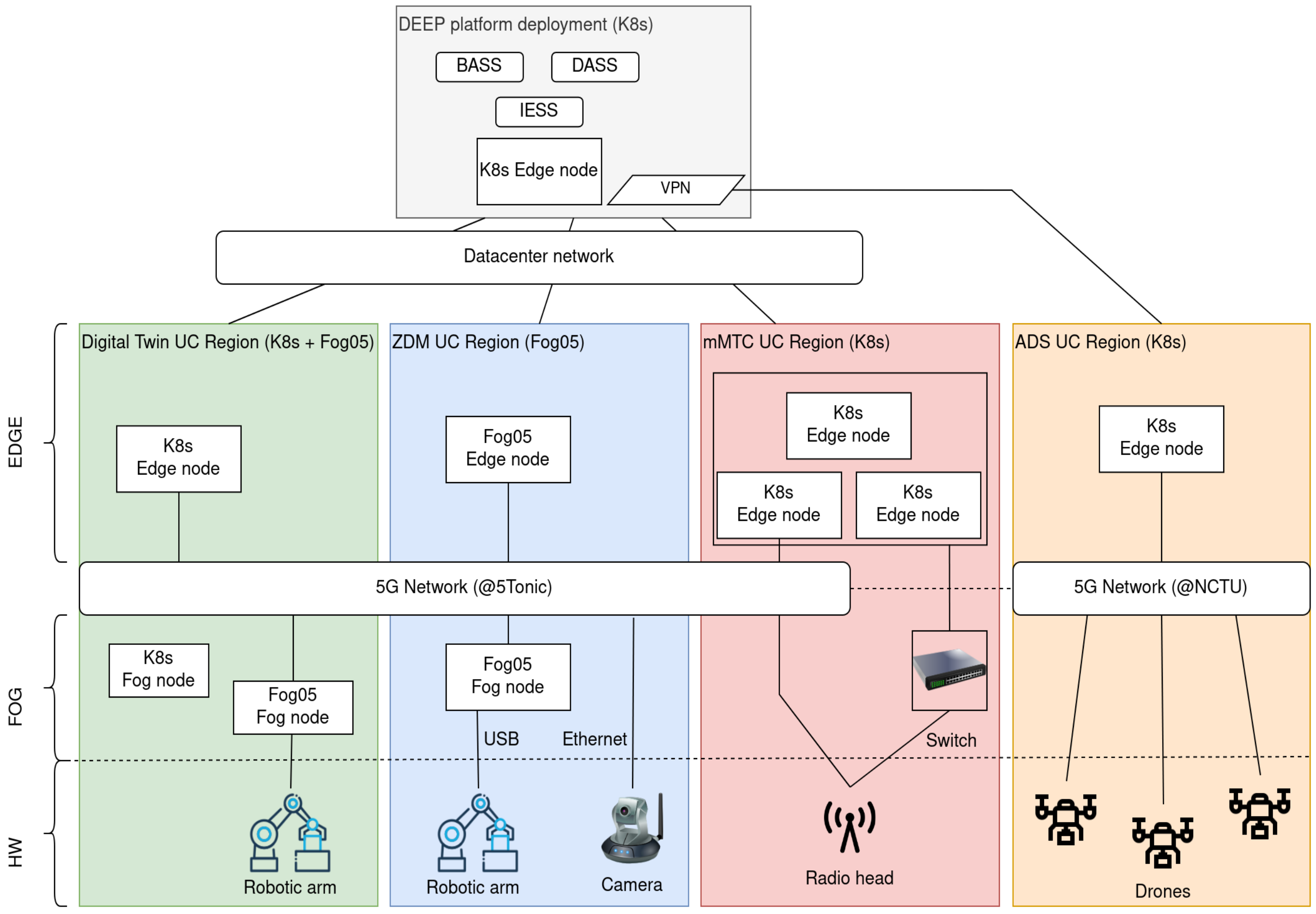


FIGURE 4-2 Trial setup at 5TONIC premises

The DEEP platform deployment, the Digital Twin UC Region Edge, the Digital Twin UC Region Fog, the ZDM UC Region, and the mMTC UC Region are deployed co-located with the resources provided by 5TONIC. The ZDM UC Region and the ADS UC Region, is are remotely located in London and Taiwan respectively and orchestrated remotely by the same DEEP instance through a VPN connection. Each region collects, defines a separated set of resources vertical owned by a single vertical premises, dedicated to hosting one or more instances of the corresponding I4.0 use case. The Digital Twin UC Region Edge and the Digital Twin UC Region Fog (green and purple inFigure 4-2) includes a Kubernetes (K8s) cluster composed of an edge node and a fog node, and a trivial Fog05 cluster composed of a single node. The ZDM UC Region (light blue inFigure 4-2) is composed of an Edge node and a Fog node, both controlled by Fog05 resources orchestrator. The mMTC UC Region (light red inFigure 4-2) includes a K8s cluster composed by three Edge nodes. The Digital Twin use case, the ZDM use case, and the mMTC use case make use of the 5G-SA network described in Section 2.1. In particular, the Fog nodes are equipped with 5G UEs and are connected to the radio interface. The ZDM UC Region (light blue in Figure 4-2) is composed of an Edge node and a Fog node, both controlled by the Fog05 resources orchestrator and it includes a 5G Network deployed at IDCC premises. Finally, the ADS UC Regioni (light orange in Figure 4-2) includes a single Kubernetes Edge Node and makes use of the 5G-NSA network solution described in Section 2.2. To implement the nodes shown in Figure 4-2, both virtual machines and hardware devices have been used. The virtual machines have been created on top of the hardware resources presented in Table 4-1. Table 4-2 reports all the nodes with their technical specification.

TABLE 4-2: Resource specification for the trial setup

Role	Device Type	CPU cores	Memory	Disk	Operating System
DEEP Platform deployment					
K8s Edge Node	KVM machine	8	12 GB	300 GB	Ubuntu 20.04
Digital Twin UC Region					
K8s Edge Node	KVM machine	4	8 GB	32 GB	Ubuntu 20.04
K8s Fog Node	NUC MiniPC	4	8 GB	32 GB	Ubuntu 18.04
Fog05 Fog Node	RaspberryPi 3	4	1 GB	32 GB	Ubuntu 18.04
ZDM UC Region					
Fog05 Edge Node	Nvidia Xavier	8	32 GB	32 GB	Ubuntu 18.04
Fog05 Fog Node	Notebook	8	8 GB	500 GB	Ubuntu 20.04
mMTC UC Region					
K8s Edge Node	KVM machine	4	16	50 GB	Ubuntu 20.04
K8s Edge Node	KVM machine	4	8	50 GB	Ubuntu 20.04
K8s Edge Node	KVM machine	4	16	50 GB	Ubuntu 20.04
ADS UC Region					
K8s Edge Node	2U Server w/ GPU (Tesla V100)	24	128	1TB	Ubuntu 18.04

5. Final validation results of I4.0 use cases

In this section, we present the final validation results of the three I4.0 use cases, regarding network and system performances, such as throughput, latency, resource utilization and robot movement accuracy, etc. In Section 4.1 the general experimental setup is provided for all three use cases. Sections 5.1, 5.2 and 5.3) present the specific testbed setup and experimental results of Digital Twin, ZDM and mMTC use cases, respectively.

5.1. I4.0-UC1: Digital Twin

The second year was focused on the development of additional AI/ML features for the Digital Twin use case and their integration in the DEEP platform, namely:

- Movement prediction
- Obstacle avoidance
- SLA Enforcer

As the date of the deliverable, the *Replay* (already showcased in the 1st year final demo) and the *Movement Prediction* features were fully integrated into the DEEP platform.

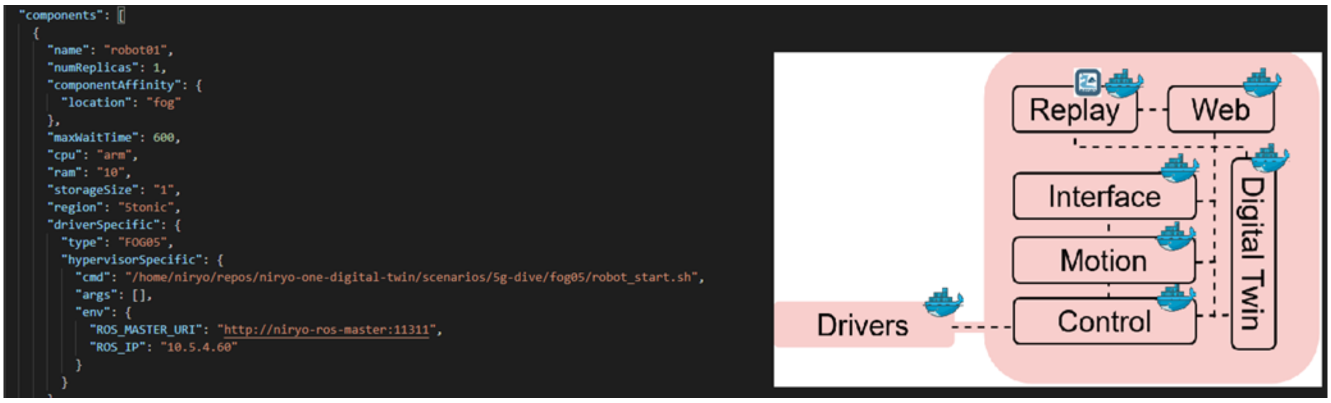
The DEEP integration validation of the Digital Twin service and the integrated AI feature is presented more in detail (Section 5.1.1). In the 2nd year, the e2e connectivity was upgraded from 5G-NSA to 5G-SA, of which we provide profiling network measurements (Section 5.1.2).
Finally, we report experimental results in Section 5.1.3 that validate and evaluate the performance of the algorithms underlying the additional features.

5.1.1. DEEP Integration Validation

In the trials, the BASS and the IESS are fully integrated with the Digital Twin service. In this section, we describe in detail the integration with the BASS and the IESS showing that the DEEP works properly with the Digital Twin Service. Please note that the DASS was successfully integrated into the Digital Twin service in the first year of the project through the Relay Feature.

5.1.1.1. BASS

The Digital Twin deployment is automated through the BASS using a combination of k3s (lightweight Kubernetes) driver for the Edge modules and Eclipse Fog05 driver for the Fog nodes. The BASS offers the benefits of deploying native applications on the physical entities at the factory floor through the Fog05 driver. This feature of the BASS is useful for Digital Twin applications because very often the physical devices (e.g. robots, production lines) are customer specific hardware that has very limited capabilities and programing interfaces that do not support virtualization. In order to deploy the Digital Twin service, the remote operator will fill in the BASS VSD that contains abstracted information about the modules that are part of the service. Figure 3-1 shows the VSD section that is related to the Robot Drivers module.



```
"components": [
  {
    "name": "robot01",
    "numReplicas": 1,
    "componentAffinity": {
      "location": "fog"
    },
    "maxWaitTime": 600,
    "cpu": "arm",
    "ram": "10",
    "storageSize": "1",
    "region": "5tonic",
    "driverSpecific": {
      "type": "FOG05",
      "hypervisorSpecific": {
        "cmd": "/home/niryo/repos/niryo-one-digital-twin/scenarios/5g-dive/fog05/robot_start.sh",
        "args": [],
        "env": {
          "ROS_MASTER_URI": "http://niryo-ros-master:11311",
          "ROS_IP": "10.5.4.60"
        }
      }
```

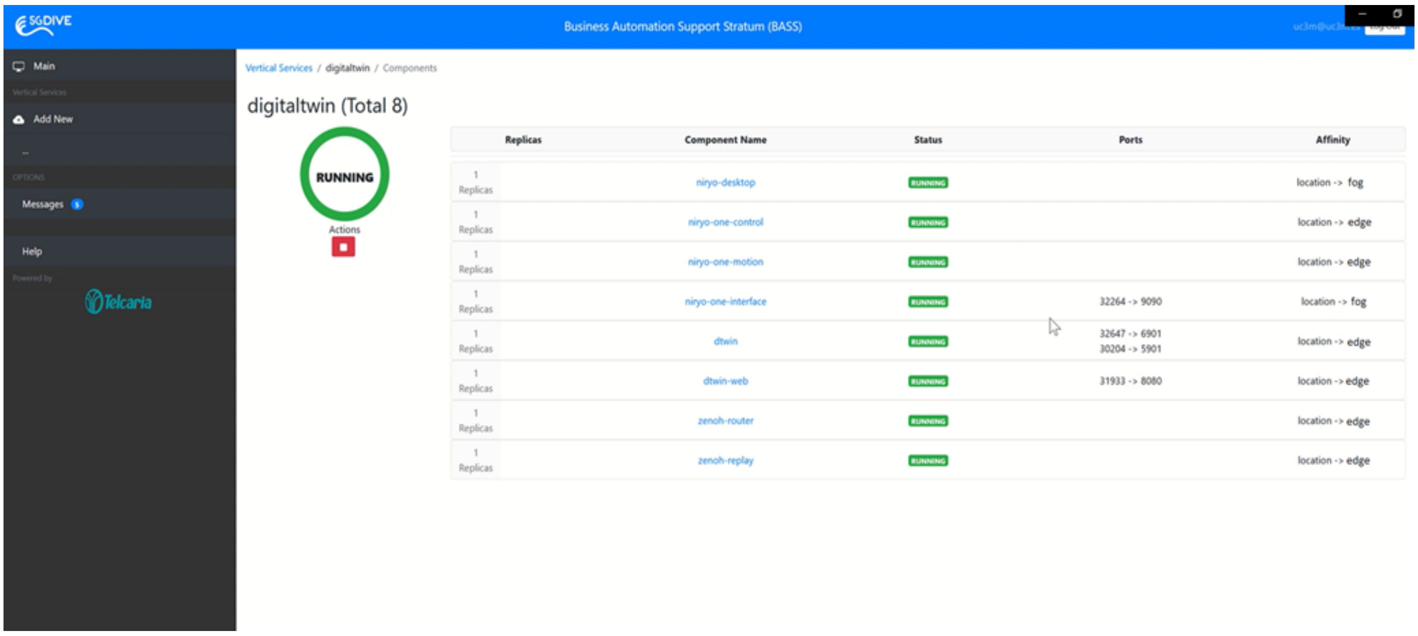
FIGURE 5-1 Robot drivers section in the Digital Twin VSD

In addition, it also shows the modules that are composing the Digital Twin service. Besides the information about the CPU architecture and the amount of ram and storage size that is required, the VSD contains information about the region and location where you want to deploy the Digital Twin service and the Driver Type. The region and location fields enable the remote operator to select the factory location where he wants to run the service and the driver type field allows the operator to select one of the available drivers in order to control the physical entity. In our example, the factory floor is 5tonic, the location is fog and the driver type is FOG05. In the hypervisorSpecific filed, instead of having hypervisor-related information, we can notice the cmd parameter that will be used by Fog05 to start the native application. It is worth mentioning that for every module that part of the Digital Twin service a section is presented in the VSD with similar information. Table 5-1 presents the modules that are part of the Base Digital Twin service with their Location and Orchestration Driver that they use.

TABLE 5-1 Digital Twin modules, deployment location and BASS orchestration Driver

Module	Location	Orchestrator Driver
Drivers	Fog	Fog05
Control	Edge	K3s
Motion	Edge	K3s
Interface	Fog	K3s
Digital Twin	Edge	K3s
Replay	Edge	K3s
Web	Edge	K3s

Once the VSD is completed and submitted through the BASS web-view, all the modules will be loaded, deployed and instantiated. Figure 5-2 shows the BASS upon successful deployment and instantiation. After this step, the Digital Twin service is up and running and the remote operator can start using the application.



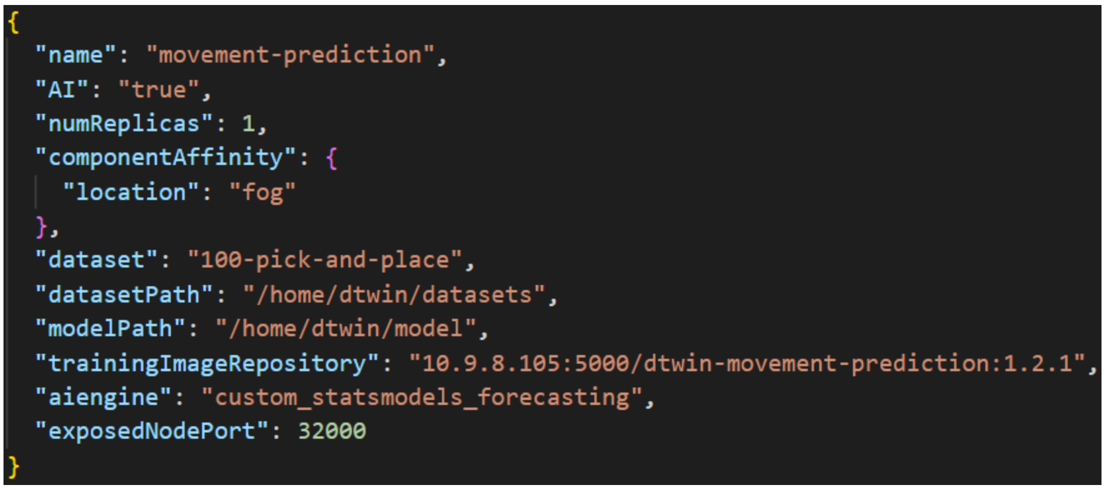
digitaltwin (Total 8)

Replicas	Component Name	Status	Ports	Affinity
1 Replicas	niryo-desktop	RUNNING		location -> fog
1 Replicas	niryo-one-control	RUNNING		location -> edge
1 Replicas	niryo-one-motion	RUNNING		location -> edge
1 Replicas	niryo-one-interface	RUNNING	32264 -> 9090	location -> fog
1 Replicas	dtwin	RUNNING	32647 -> 6901 30204 -> 5901	location -> edge
1 Replicas	dtwin-web	RUNNING	31933 -> 8080	location -> edge
1 Replicas	zenoh-router	RUNNING		location -> edge
1 Replicas	zenoh-replay	RUNNING		location -> edge

FIGURE 5-2 BASS Web-view after successful deployment and instantiation

5.1.1.2. IESS

In the second year of the project, we extended the Base Digital Twin service with Movement Prediction AI/ML feature and we integrate with the DEEP platform. The movement prediction was conceptually described in D2.3 [7], while the technical details of the implementation are presented in Section 5.1.3.1.1. The complete integration of the Movement Prediction AI/ML-based model with the DEEP was divided into three steps.

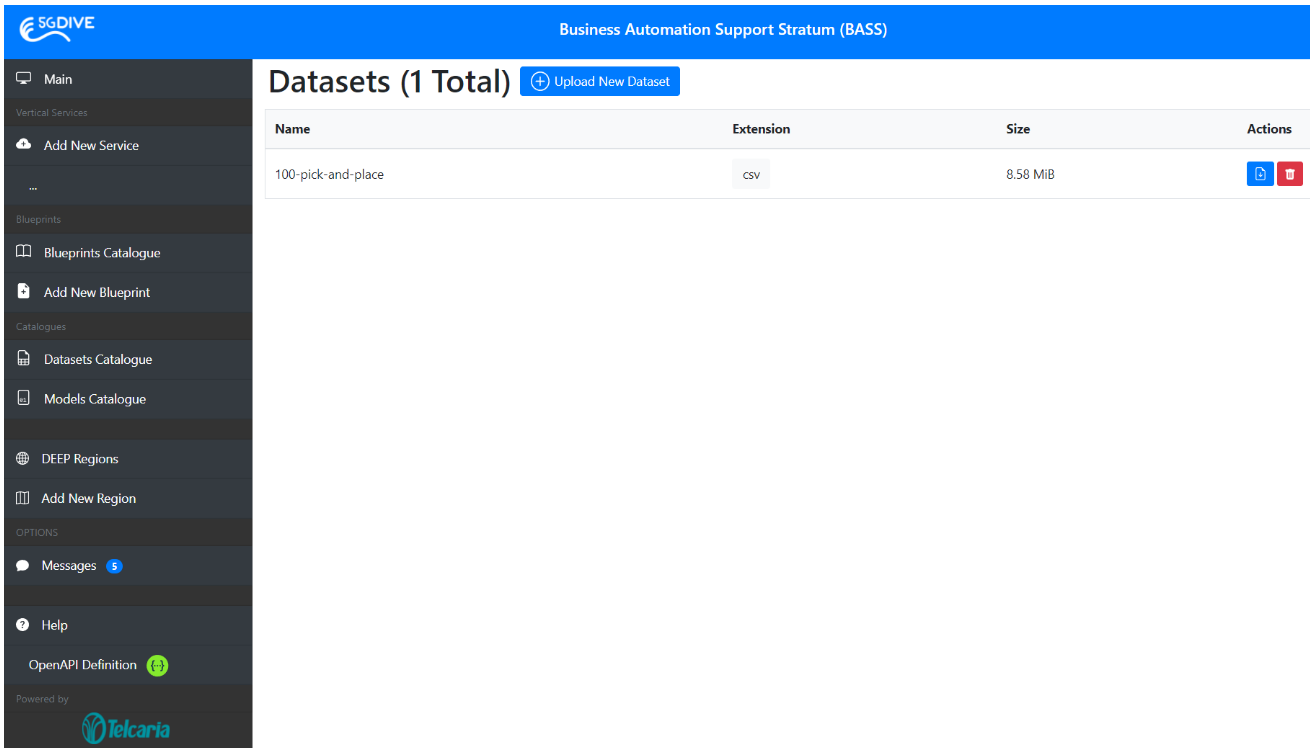


```
{
  "name": "movement-prediction",
  "AI": "true",
  "numReplicas": 1,
  "componentAffinity": {
    "location": "fog"
  },
  "dataset": "100-pick-and-place",
  "datasetPath": "/home/dtwin/datasets",
  "modelPath": "/home/dtwin/model",
  "trainingImageRepository": "10.9.8.105:5000/dtwin-movement-prediction:1.2.1",
  "aiengine": "custom_statsmodels_forecasting",
  "exposedNodePort": 32000
}
```

FIGURE 5-3 Movement prediction section in the VSD

Dataset creation and upload in the DEEP Dataset Catalogue

In order to train and use the Movement Prediction model, we created a dataset by performing pick and place actions. The pick and place actions were manually repeated 100 times by a human operator to create a dataset that contains 187108 commands. The dataset was generated in .csv format and contained the joint states of the robot manipulator under ideal network conditions. The created dataset was then uploaded in the DEEP Dataset Catalogue where the data can be visualized. Figure 5-4 shows the Dataset Catalogue view of the BASS.



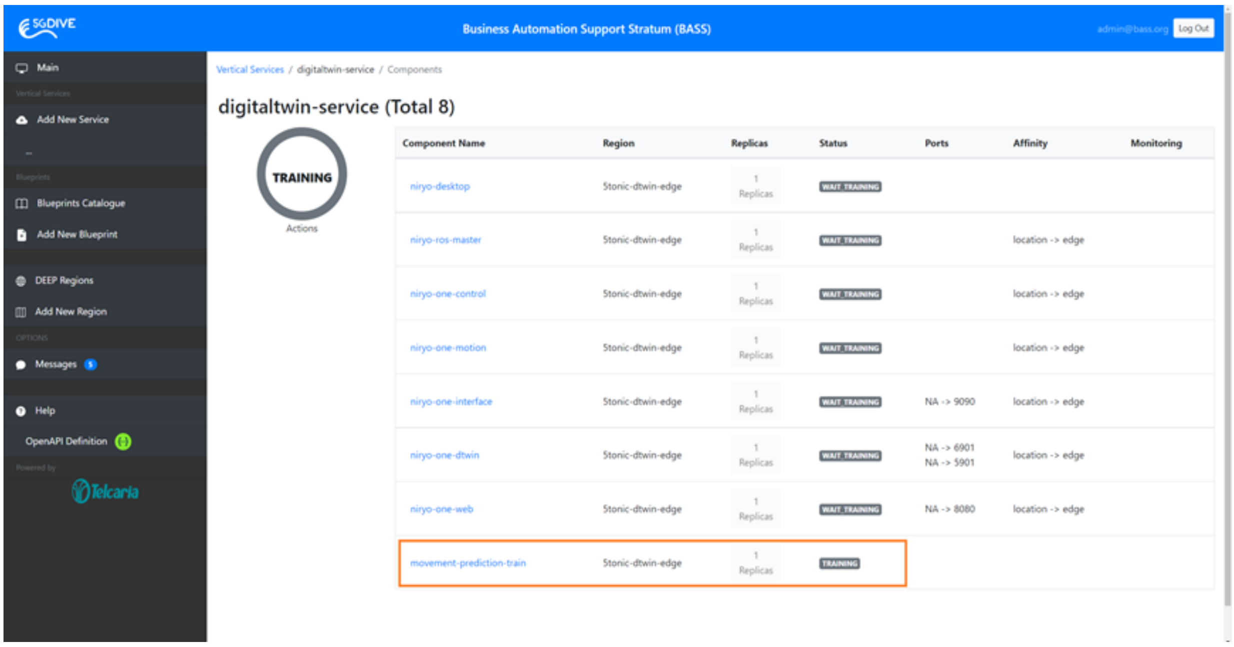
Datasets (1 Total)

Name	Extension	Size	Actions
100-pick-and-place	csv	8.58 MiB	

FIGURE 5-4 BASS Dataset catalogue web-view

IESS Training and BASS deployment

Next, the VSD was updated by adding a section related to the Movement Prediction AI/ML module (see Figure 5-3). During the service loading process, the BASS first will check if there is an AI module that needs to be trained using the IESS. It will use the dataset that is indicated in the *dataset* field of the VSD to train the model.

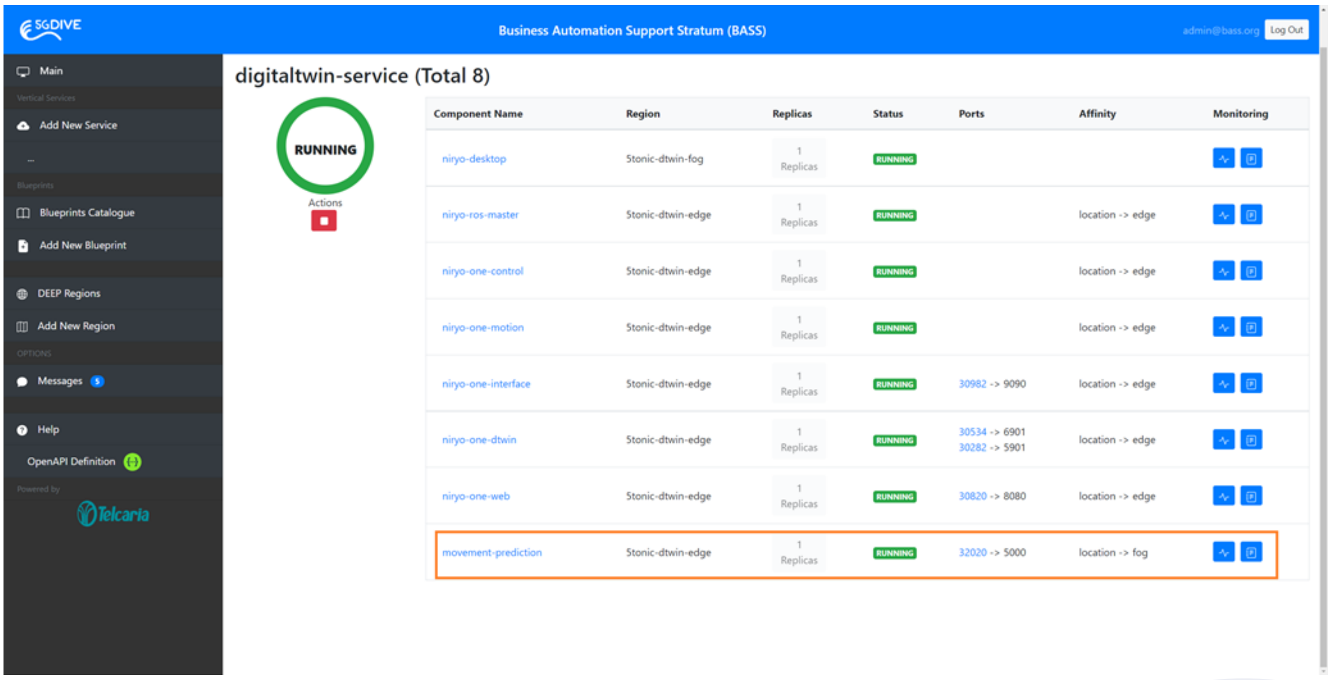


digitaltwin-service (Total 8)

Component Name	Region	Replicas	Status	Ports	Affinity	Monitoring
niryo-desktop	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING			
niryo-ros-master	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING		location -> edge	
niryo-one-control	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING		location -> edge	
niryo-one-motion	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING		location -> edge	
niryo-one-interface	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING	NA -> 9090	location -> edge	
niryo-one-dtwin	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING	NA -> 6901 NA -> 5901	location -> edge	
niryo-one-web	5tonic-dtwin-edge	1 Replicas	WAIT_TRAINING	NA -> 8080	location -> edge	
movement-prediction-train	5tonic-dtwin-edge	1 Replicas	TRAINING			

FIGURE 5-5 BASS Web-view during IESS training

The AI model type that is desirable for the Movement Prediction is specified in the *ai engine* field. Figure 5-2 shows the BASS web-view during the IESS training procedure. When the training is done, the IESS will store the trained model in the IESS catalog (using the name that is indicated in the *trainingImageRepositiry* filed) and will inform the BASS that it can proceed with the loading and deployment process. Figure 5-6 shows the BASS upon successful deployment with the Movement prediction inference application as part of the Digital Twin service. After this step, the Movement Prediction service is up and running and Digital Twin can start using the predictions.



digitaltwin-service (Total 8)

Component Name	Region	Replicas	Status	Ports	Affinity	Monitoring
niryo-desktop	5tonic-dtwin-fog	1 Replicas	RUNNING			
niryo-ros-master	5tonic-dtwin-edge	1 Replicas	RUNNING		location -> edge	
niryo-one-control	5tonic-dtwin-edge	1 Replicas	RUNNING		location -> edge	
niryo-one-motion	5tonic-dtwin-edge	1 Replicas	RUNNING		location -> edge	
niryo-one-interface	5tonic-dtwin-edge	1 Replicas	RUNNING	30982 -> 9090	location -> edge	
niryo-one-dtwin	5tonic-dtwin-edge	1 Replicas	RUNNING	30534 -> 6901 30282 -> 5901	location -> edge	
niryo-one-web	5tonic-dtwin-edge	1 Replicas	RUNNING	30820 -> 8080	location -> edge	
movement-prediction	5tonic-dtwin-edge	1 Replicas	RUNNING	32020 -> 5000	location -> fog	

FIGURE 5-6 BASS web-view upon successful deployment and instantiation with the movement prediction

Command Inference and execution

In the last step of the integration, we developed a simple client that via HTTP was consuming the predictions in real-time from the deployed Movement Prediction model. Periodically, the client will send the last 10 successfully executed commands by the robot arm to the Movement Prediction model, and the model will respond with the predicted command of the next time slot. Whenever a control command will not arrive in time, the client will execute the predicted command.

5.1.2. End-to-end profiling

Initially, we profiled the network performance of 5G-SA, to see if they were capable of sustaining the KPIs required for the correct functioning of the Digital Twin use case.
Benchmark results were carried out for the following metrics: throughput, packet loss, latency RTT, jitter, both uplink and downlink.
We used the commercial 5G CPE provided by Ericsson, ASKEY RTL0330, to connect a PC to the 5GC, and a VM server in the 5TONIC edge premise connected to the core, then run performance evaluation with iperf3, sending a UDP stream at 800 Mbit/s in downlink and 100 Mbit/s in the uplink. The results are reported in the following subparagraphs.

5.1.2.1. Network Measurements

The iperf3 experiments were run for 30 minutes, both for TCP and UDP.
For TCP maximum capacity the mean throughput was 817.05 Mbit/sec in downlink (see Figure 5-7) and 111.58 Mbit/sec in uplink (see Figure 5-8). The TCP slow start phase is visible in the first graph.

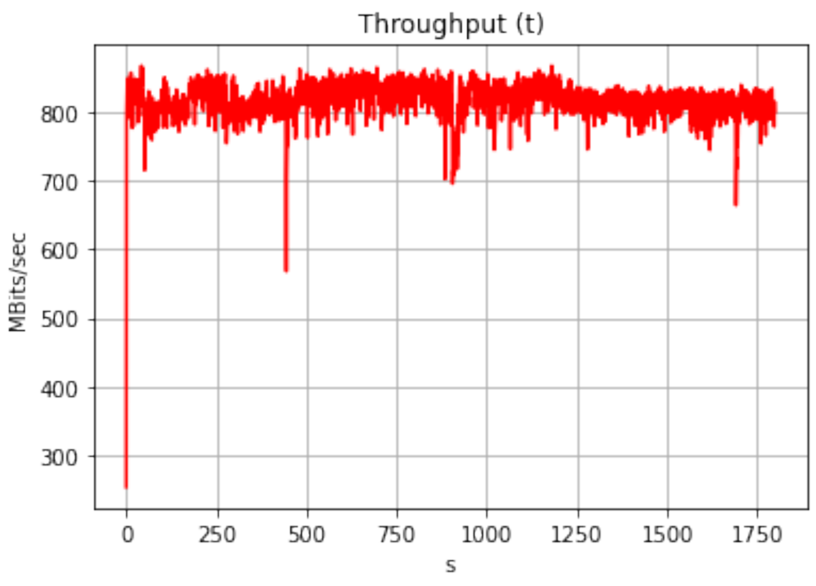


FIGURE 5-7 TCP downlink throughput

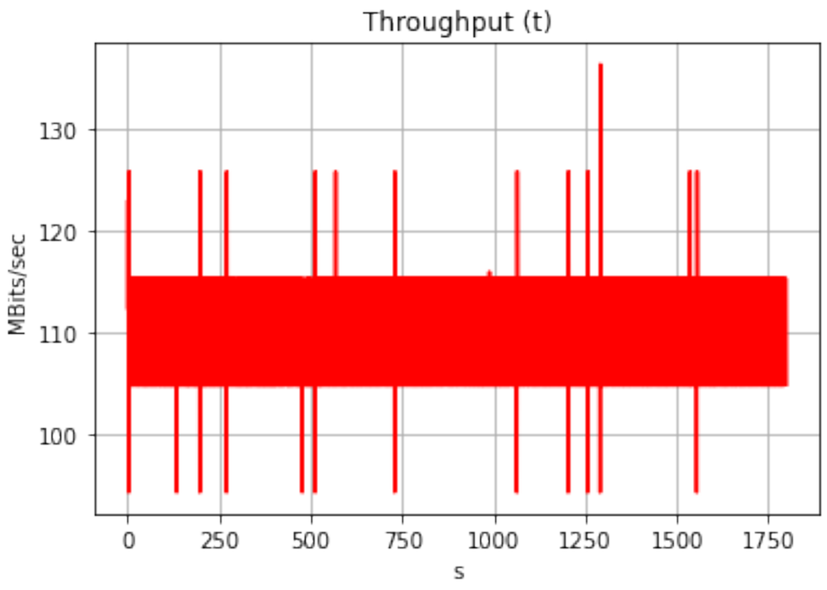


FIGURE 5-8 TCP uplink throughput

The same experiments were run with UDP using 800 Mbit/sec in downlink and 100 Mbit/sec (the TCP reference values) in uplink to measure the packet loss and obtain insights about the reliability of the wireless link. Mean packet loss in downlink was 0.027 %, while in uplink 0.0304 %, due to the links working at almost full capacity. No significant difference was found with the 5G-NSA connectivity in comparison, as the data rate boost is mainly granted by the NR technology. We could not calculate the theoretical data rate achievable due to missing parameters. As for the latency and the jitter, we report the results of the pings over 240s in Table 5-2, with the latency values in ms and the CDF of the pings in Figure 5-9 and Figure 5-10.

TABLE 5-2 Ping results

	Min (ms)	Avg (ms)	Max (ms)	Mdev	Jitter (ms)
Uplink (RTT)	13.082	19.149	59.272	3.741	3
Downlink (RTT)	7.547	16.584	28.095	4.036	4.4

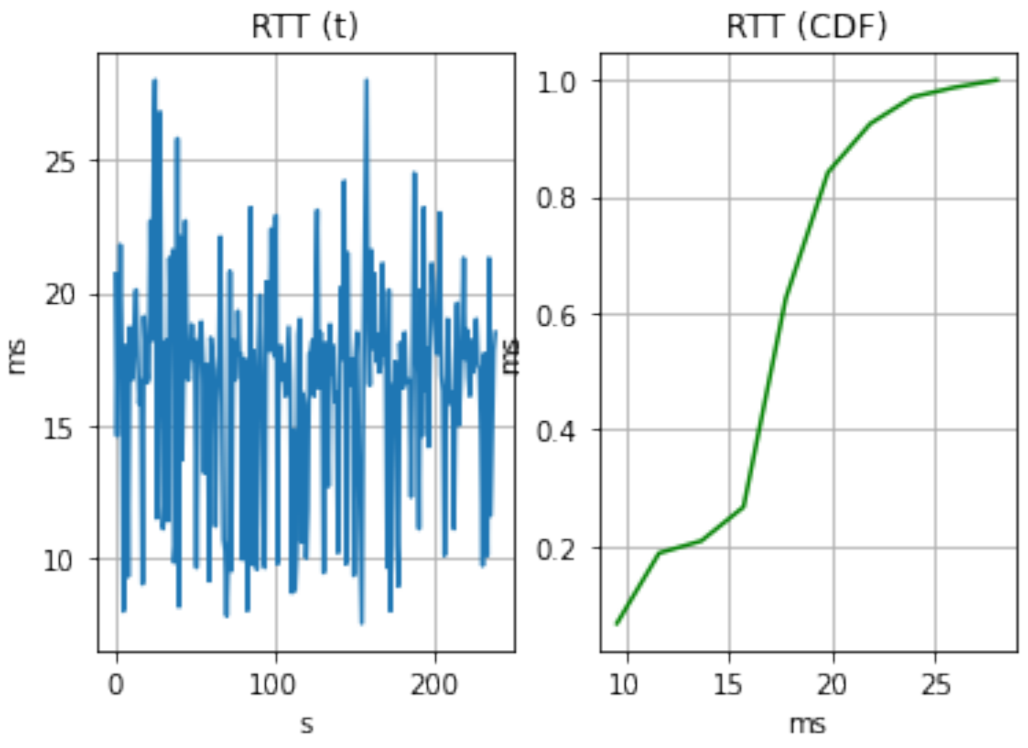


FIGURE 5-9 Downlink latency

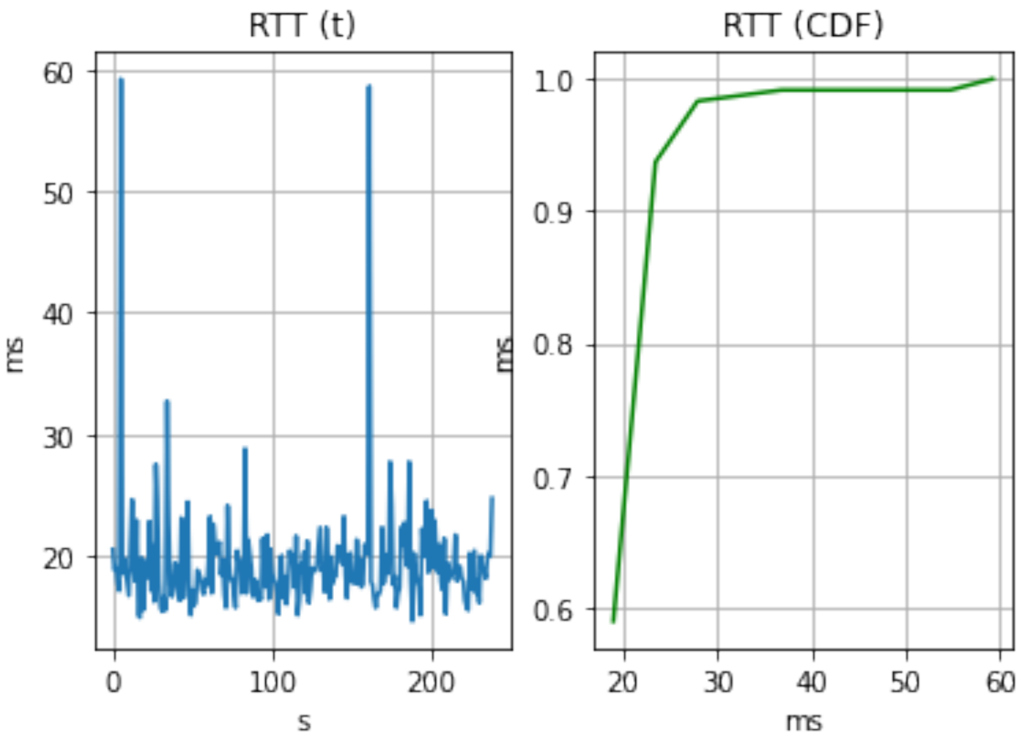


FIGURE 5-10 Uplink latency

5.1.3. Experimental Results

5.1.3.1.1. Experiment A: Movement Prediction

As described in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021), the Movement Prediction feature can predict a robot command that is missing due to packet loss, based on the historical records of the movements.

At first, a dataset of commands (in this context, the sequence of x, y, z Cartesian coordinates given to the robot) was built manually, controlling the robotic arm with the joystick to perform the same task several times. This is what happens in real scenarios: the robot is operated by a human who uses a controller to make the robot perform the same task over time. Thus, the resulting sequence of commands is something that can be predicted, as the task is repetitive *per se* and the commands are stochastic.

Initially, to validate the solution the Digital Twin stack was deployed in an Edge environment equivalent to that of D3.2 (D3.2, 2021): 1 vCPUs and 2 GB of RAM in an Edge Server (Dell PowerEdge R430). Still, the Robot Drivers were kept locally on the robotic arm. Then, the intelligence engine with the actual Movement Prediction module was deployed directly in the Niryo One robotic arm, using a TPU accelerator for ML inferring.

Performance comparison of the algorithm was conducted with respect to 3 approaches:

1. Vector Autoregressive (VAR): AI/ML algorithm, *statsmodel* library
2. Sequence-to-Sequence Neural network (seq2seq) [ref]: AI/ML, Tensorflow, 200 LSTM units, 163200 parameters.
3. Moving Average (MA), no AI/ML.

The dataset consists of 4 tasks/actions, executed 20 times. Commands were issued every 20ms, as usual. Packet-loss was set at 5% over Wi-Fi.

The training of the algorithm was realized using 80% of the dataset, the remaining 20% was used to test the prediction accuracy.

The Root Mean Square Error (RMSE) was adopted to estimate the accuracy of each algorithm, thus a preliminary performance comparison was carried out in Figure 5-11. The error in question is the difference between the predicted robot coordinate and what would be the real one.

Performance is affected by the size of the window of commands to be predicted. The algorithm accuracy was first evaluated for several window sizes of commands (5, 10, 50, etc.).

In Figure 5-12 and Figure 5-13, it can be noticed the difference between predictions of different actions with a window size of 5 (100 ms) and 50 (1000 ms) movements, respectively. As one can see, VAR performs the best. VAR is known to work well with correlated signals, as this was the case.

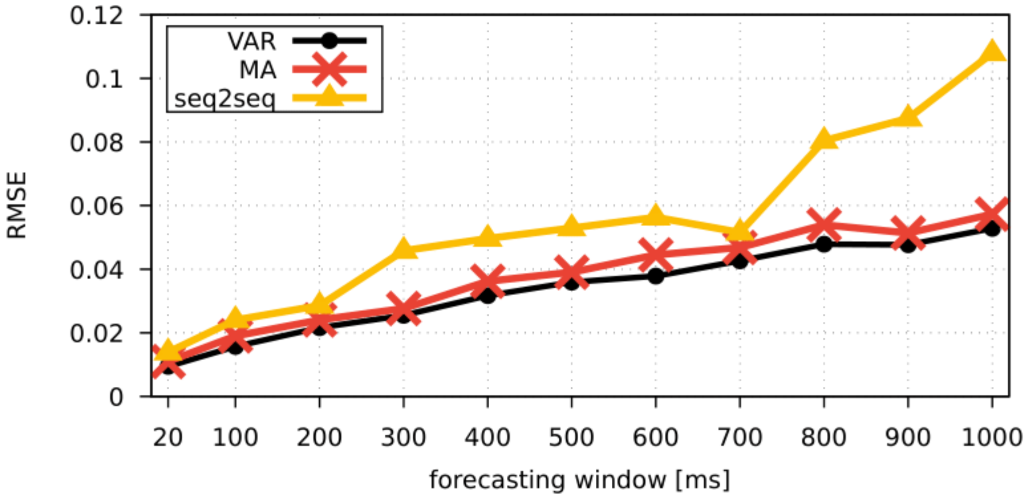


FIGURE 5-11 Comparison of movement prediction algorithms (training set).

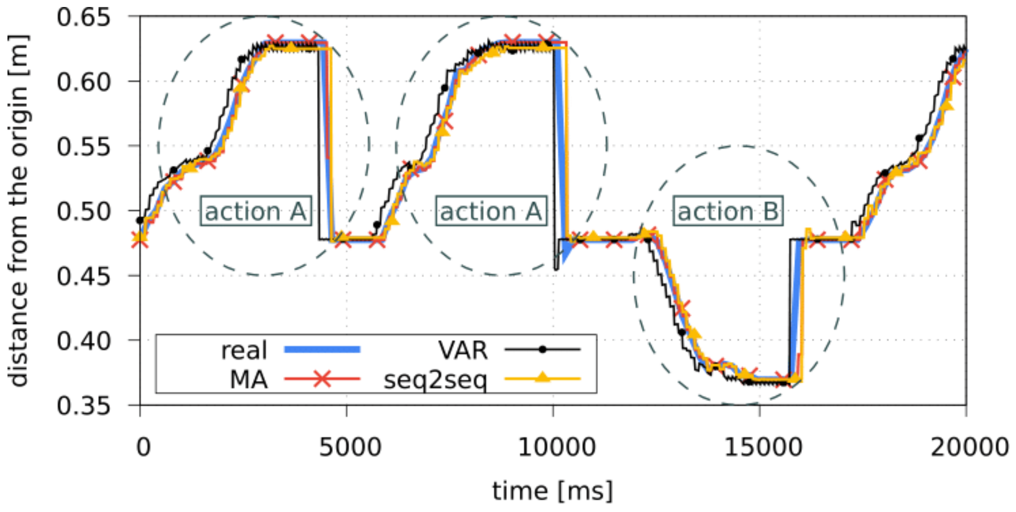


FIGURE 5-12 Movement predictions for different actions and different algorithms (5 movements).

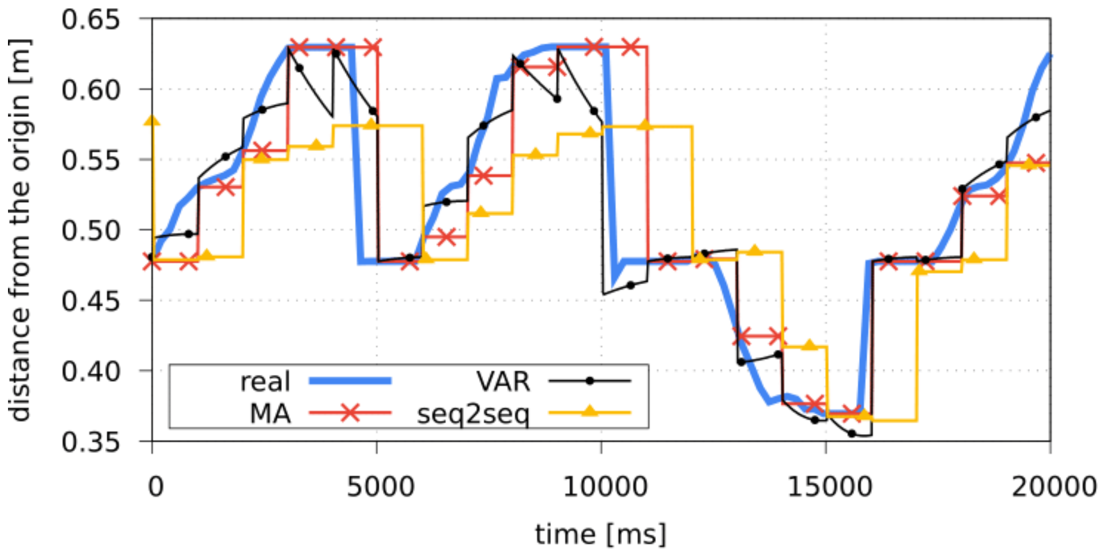


FIGURE 5-13 Movement predictions for different actions and different algorithms (50 movements).

Finally, the algorithms were tested with the physical robot evaluating, introducing a packet loss of 5% in a Wi-Fi link issuing the commands to the tele operated robotic arm, for time-spans of 200 ms (corresponding to a window size of 10 movements where predictions need to happen).
Qualitatively, the robot experienced more stable and less jerky movements when the VAR algorithm was applied, w.r.t other algorithms or no AI/ML at all, as shown in the performance representation of Figure 5-14.

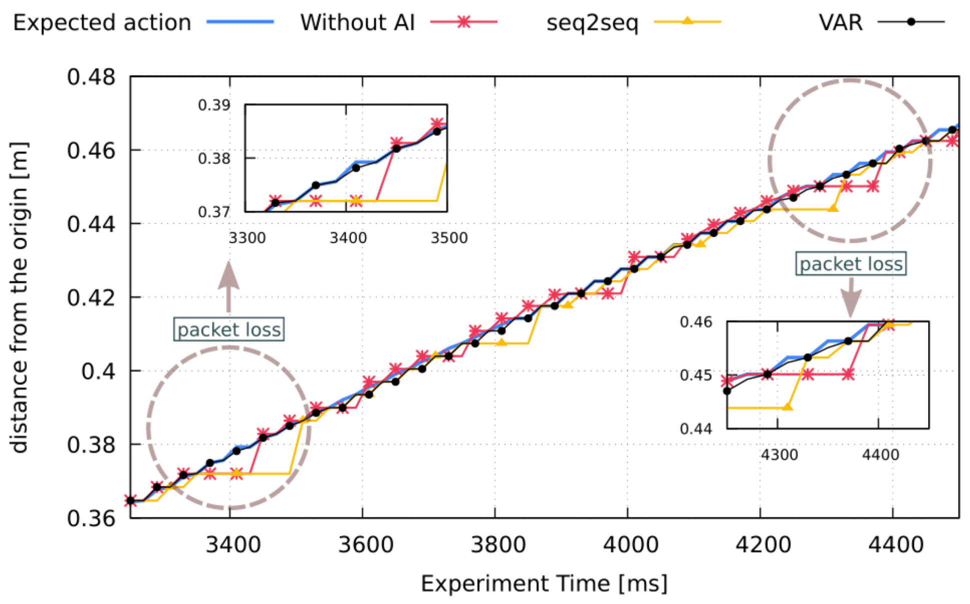


FIGURE 5-14 Performance of movement prediction when applied to the physical robot

As the Digital Twin use case in practice utilizes a close-by fog module (Raspberry Pi3), where the Drivers are deployed for optimal control-loop time and commands' latency (around 2ms), and the movement prediction needs to act promptly with comparable latency times, further experiments were conducted by running the training of the algorithms and the inference of the command directly in this constrained device, serving closely the robot. The RPi proved to be suitable to sustain the needs of the use case, carrying out the training in a reasonable amount of time (around 6ms) and predicting a missing command in less than 2 ms (see Table 5-3). Nevertheless, training is a sporadic event to be performed for building the model from scratch or when updating. For the sake of comparison, the time performance of more powerful hardware is reported in Table 5-4.

TABLE 5-3 Inference times of the movement prediction module in the Raspberry pi

	Load Data (s)	Down Sampling (s)	Check Quality (s)	Training Model (s)
Raspberry Pi3 (Robot)	1.95 ± 0.02	0.26 ± 0.007	306.38 ± 3.15	50.98 ± 0.54

TABLE 5-4 Hardware performance comparison

	Training (min)	Inference (ms)
Raspberry Pi3 (Robot)	5.99 ± 0.06	1.60 ± 0.16
NVIDIA Jetson Nano (Robot)	1.31 ± 0.01	0.61 ± 0.28
Laptop (UE)	0.36 ± 0.01	0.22 ± 0.10
Local Server (Edge)	0.23 ±0.007	0.0001 ± 0.00003

5.1.3.1.2. Experiment B: Obstacle Avoidance

As explained in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021), the obstacle avoidance feature allows the robot to calculate the motion plan to move an object from a source point to a destination, avoiding an obstacle. Thus, as a first step, the 2D coordinates that the end effector has to follow to perform the pick-and-place task, were evaluated using the popular reinforcement learning algorithm Q-Learning.
The algorithm was first designed and then evaluated in a virtual Python environment representing the robot reachability workspace, an *anulus* of *radii* R and r with the robot placed in the middle of the inner circle, with the following characteristics:

- R = 40 cm, outer radius
- r = 10 cm, inner radius
- Δ = 4 cm, discretization step of the environment
- a set of 8 possible actions, A = {*up, right, left, down, up-right, up-left, down-right, down-left*}

A parallelepiped, whose base is a square of size $(R-r)/r$ and arbitrary height, short enough so that the robot can overcome it from above, representing the bounding box of a potential obstacle impeding the robot movements was added to the virtual environment for every possible position within the discretized workspace. For a taller obstacle, the inference of 2D coordinates would not have been enough, also joint collision must have been considered. Actually, this would be the case of a real application, but for the sake of simplicity we carried out initial results with the obstacle being way shorter than the robot, so that collision could be neglected in the learning phase.
Q-Learning was used to calculate the 2D trajectory coordinates that the end effector of the robot had to follow to move an object from fixed source to fixed destination position for every possible position of the obstacle, a total of 238 sub episodes with the given Δ.
In QL an "episode" is the set of states (positions in the workspace) that the learning agent has to follow to move from the initial state (source) to the final state (destination). Throughout its movements, the agent gains a reward inversely proportional to the distance of the destination, a negative reward if it hits an obstacle, and a greater positive reward when it reaches the destination. In fact, the reward function R was chosen as follows, depending on the state observed by the agent:

$$\begin{cases} R = -1000, & if\ state\ \in obstacle \\ R = 1000, & if\ state = d \\ R = \ 80 - \|x-d\| & \end{cases}$$

Where d is the destination where 80 was chosen as it is the maximum distance in the environment corresponding to the diameter of the circular workspace.
Every time the agent hits an obstacle, it is restarted, so a complete episode corresponds to full agent travel from source to destination. The Q-Learning training consists in updating a Bellman equation stored in a q-table for each state/action pair [complete here].

We called "sub episode" the Q-Learning training related to a particular permutation of the environment, i.e., a specific obstacle position. The algorithm was run over these sub episodes for N=4000 episodes, a number that was found to be adequate for the algorithm to reach reward convergence for each possible sub episode/environment configuration. Also, the ε-greedy strategy was implemented, where the agent could choose between *exploration* and *exploitation* with probability ε, representing the balance between the two policies. Moreover, the parameter ε was decayed exponentially by a decaying factor of 0.95 ($\varepsilon(n) = 0.95^n$, where n is the episode number), so that after approximately 200 episodes ε was almost zero, and a longer phase of *exploitation* could start. Actually, when the algorithm is run, pure exploitation is what is adopted (always the best action according to the q-table is selected).
Overall training time over all the sub episodes and for the chosen number of episodes was 23 minutes. Below, in Figure 5-15, you can find the plot of the average cumulative reward per episode on a logarithmic scale exhibiting the desired convergence behaviour after episode 120.

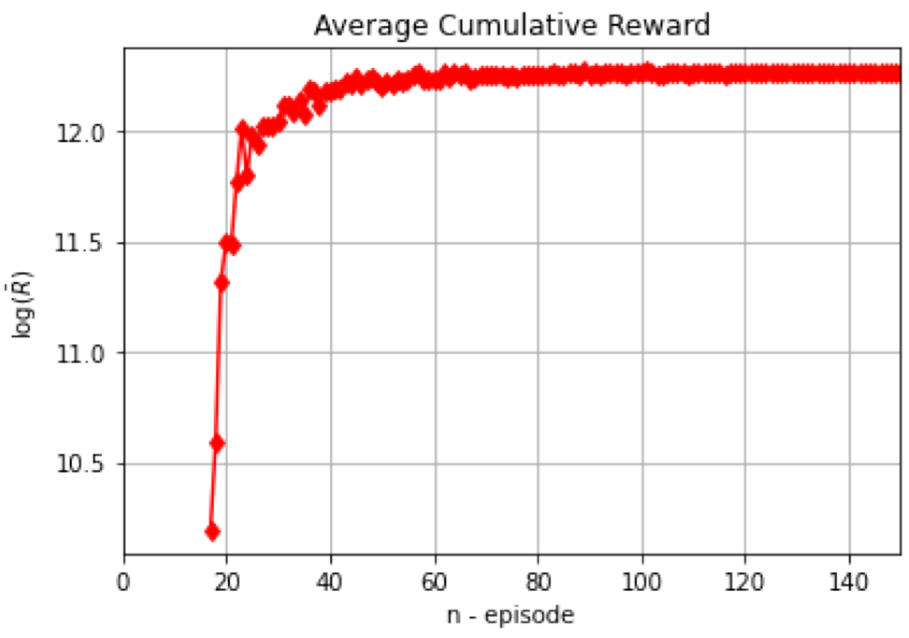


FIGURE 5-15 Average cumulative reward during training

In Figure 5-16, the virtual environment is displayed with the coordinates steps representing the travel the end effector of the robot has to make to pick up an object from the source and drop it at the destination.

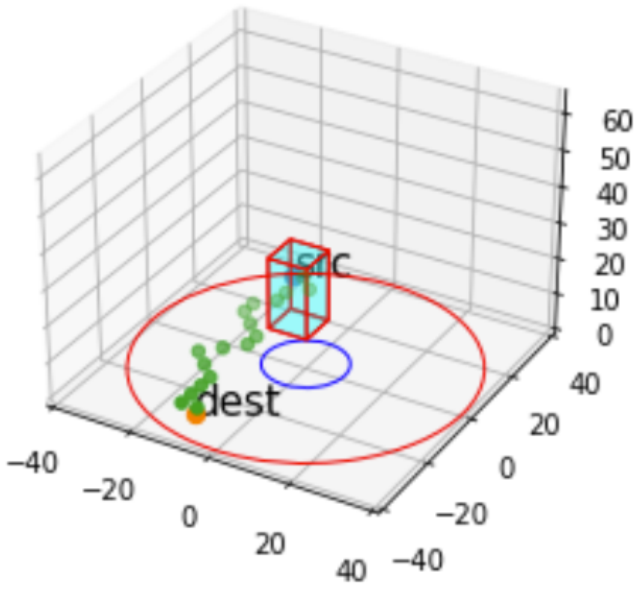


Figure 5-16 Virtual environment for the training of the algorithm representing the robot workspace
Finally, the solution was validated in the simulated robot, using rviz, the robot visualizer, and *moveit*, the ROS Motion Planning calculating the *inverse kinematics* transformation between the coordinates of the robot end-effector, calculated with Q-Learning, and the joint state movements. The feature was not validated with the physical robot, due to time constraints, but it laid the premises for further development and integration in more advanced proof of concepts of the Digital Twin use case.

5.1.3.1.3. Experiment C: SLA Enforcer

An SLA Enforcer proof-of-concept was realized for the Digital Twin use case. The e2e application latency was the metric identified as the most impacting the QoE, as it affects the control loop of the robot and can easily produce a loss in the precision of the synchronization with the digital replica. Latency thresholds were set as possible SLA requirements not to be violated, for which two digital twin SLOs were set:

1. Precision task: maximum e2e latency allowed 500 ms
2. Screening task: maximum e2e latency allowed 1000 ms

The parameters influencing the latency performance were found to be mainly the CPU and memory utilization.

The SLA enforcer's overall workflow is presented in Figure 5-17. During the service creation, the DEEP platform configures the SLA Enforcer based on the vertical intend:

1. The Vertical Service Coordinator extracts any intend-based SLA from the vertical oriented descriptor, forwarding them to the SLA & Policy Management.
2. The SLA & Policy Management validates the intend, translating them into a set of policies that identify their scope, thresholds and validation data.
3. Based on the extracted policies, the SLA & Policy Management requests an AI/ML model from the IESS and the deployment of the monitoring probes towards the Active Monitoring
4. Using AutoAI, the IESS selects the AI/ML model from its catalogue, performs its training and cross-validation, delivering the trained model to the SLA & Policy Management.

Finally, the SLA & Policy Management triggers new instance of SLA Enforcer tied to the vertical service SLAs.

The modules are deployed as Docker containers. The *Monitoring Probe* in-between the robot and the service stack relays the data to the DRL SLA Enforcer model through the DASS/Zenoh router. Then the SLA Enforcer can scale the desired resources using the Docker orchestrator.

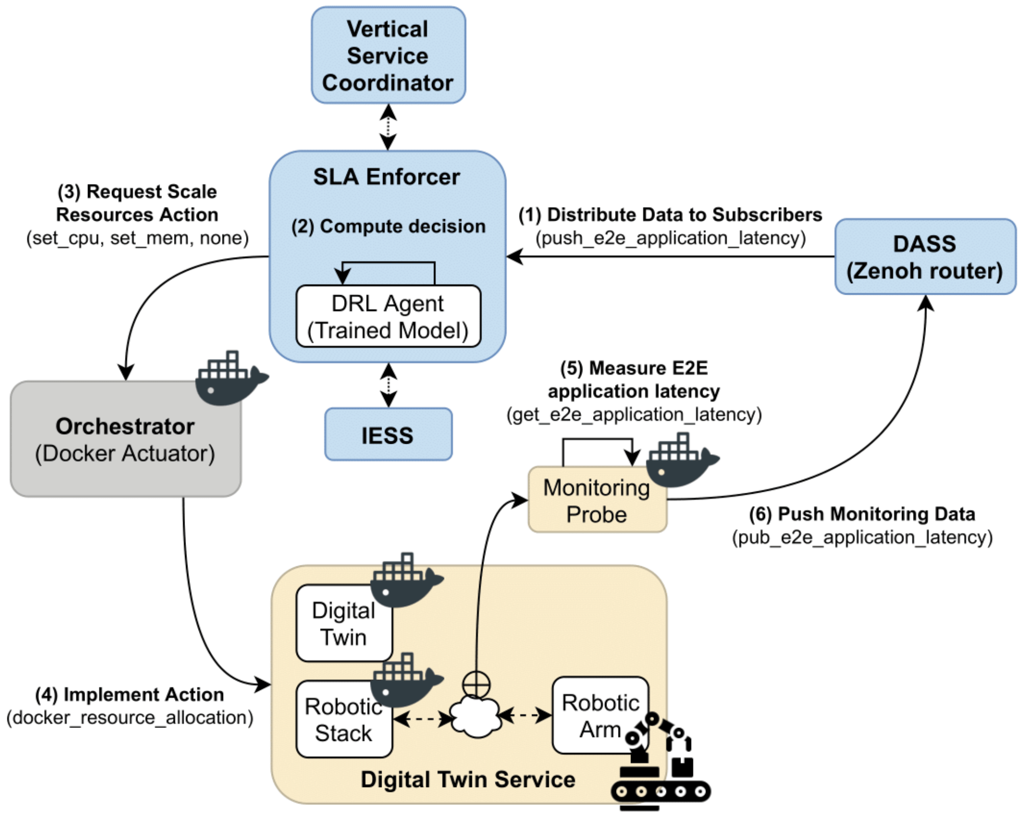

Vertical Service Coordinator
SLA Enforcer
(2) Compute decision
DRL Agent (Trained Model)
(1) Distribute Data to Subscribers (push_e2e_application_latency)
DASS (Zenoh router)
(3) Request Scale Resources Action (set_cpu, set_mem, none)
Orchestrator (Docker Actuator)
IESS
(5) Measure E2E application latency (get_e2e_application_latency)
Monitoring Probe
(6) Push Monitoring Data (pub_e2e_application_latency)
Digital Twin
Robotic Stack
Robotic Arm
Digital Twin Service
(4) Implement Action (docker_resource_allocation)

FIGURE 5-17 Overall workflow of the SLA enforcer

A Deep Reinforcement Learning (DRL) algorithm was developed to evaluate the optimal minimum MEM and CPU allocation for the system in order not to violate the system above.
The following experiment was performed: commands were exchanged between the Digital Twin stack and the Robotic Arm, in both directions. A module referred to as Monitoring Probe was placed in between to inspect the E2E latency of these commands, which is the time from when commands are sent and it is executed by the robot. As commands are processed by the Robotic Stack (Control, Motion, Interface), the resources associated with this module were found to be the ones affecting the performance and the ones to be scaled accordingly.
In the training of the algorithm, the latency values were constantly monitored to see if they violated the thresholds. When this would happen, the associated reward penalized the associated CPU and MEM percentage utilization. At the end of the training, the RL algorithm was able to carry out optimal CPU and MEM allocations for meeting the e2e application latency requirements.
The training phase required approximately 14.500 steps for the algorithm to be able to learn, as can be noticed in Figure 5-18 and Figure 5-19, picturing the learning behaviour in the two SLOs cases. After

that number of steps, the CPU and MEM predicted limit percentages with the associated reward values settled at a level that could guarantee the required latencies (see Response Times pictured in grey).

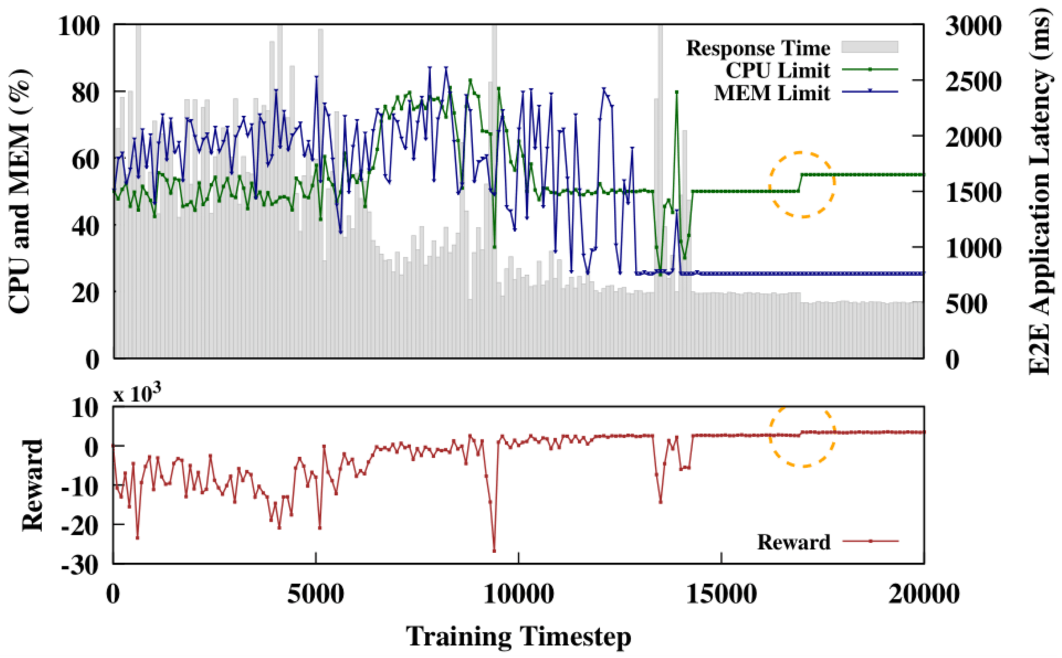


FIGURE 5-18 Learning behaviour of SLA enforcer algorithm for precision tasks (500 ms)

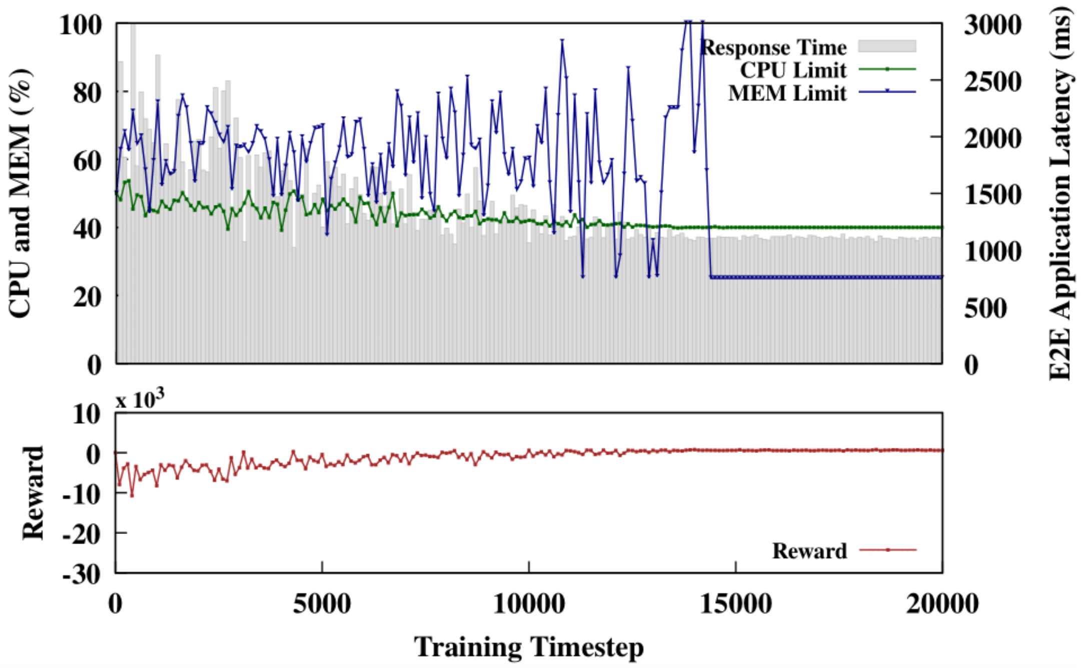


FIGURE 5-19 Learning behaviour of SLA enforcer algorithm for screening tasks (1000ms)

5.1.4. Scalability

In this section we will have a look into the vertical scaling of the Digital Twin service and how does the increase of resource consumption in an edge server can directly impact the service performance. This analysis aims to characterize the main resource usage of the designed Digital Twin service and help the potential AI-based scaling algorithms that can be implemented in the DEEP as part of the BASS. It is worth mentioning that the experimental evaluation of this section is performed over wired technology (Ethernet). The main reason is that when investigating the resource consumption of the Digital Twin service, the bottleneck is not on the communication technology.

5.1.4.1. CPU and RAM consumption

In this first set of experiments, we used the Base Digital Twin modules that were introduced in Section 3.1.1.1 in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021) The Niryo One robotic arm is connected to the Digital Twin using a 10Gb Ethernet connection. The robot is equipped with Raspberry Pi 3 with a 1.2 GHz 64-bit CPU and 1GB RAM. Each experimental run consists of the Digital Twin controlling 1-axis of the robot arm for 30 seconds and a moving offset of 0.01 rad. The experiments consider the following on-device configurations: i) Robot Drivers VNF, ii) Robot Drivers and Control VNF, ii) Robot Drivers, Control and Motion Planning VNFs, and iv) full Robot Stack. The experiments were repeated 10 times for each deployment, being presented the average values and their standard deviation. In the robot arm we obtained data for CPU and MEM consumption using the psutil (Psutil cros-platform python library) cros-platform python library.

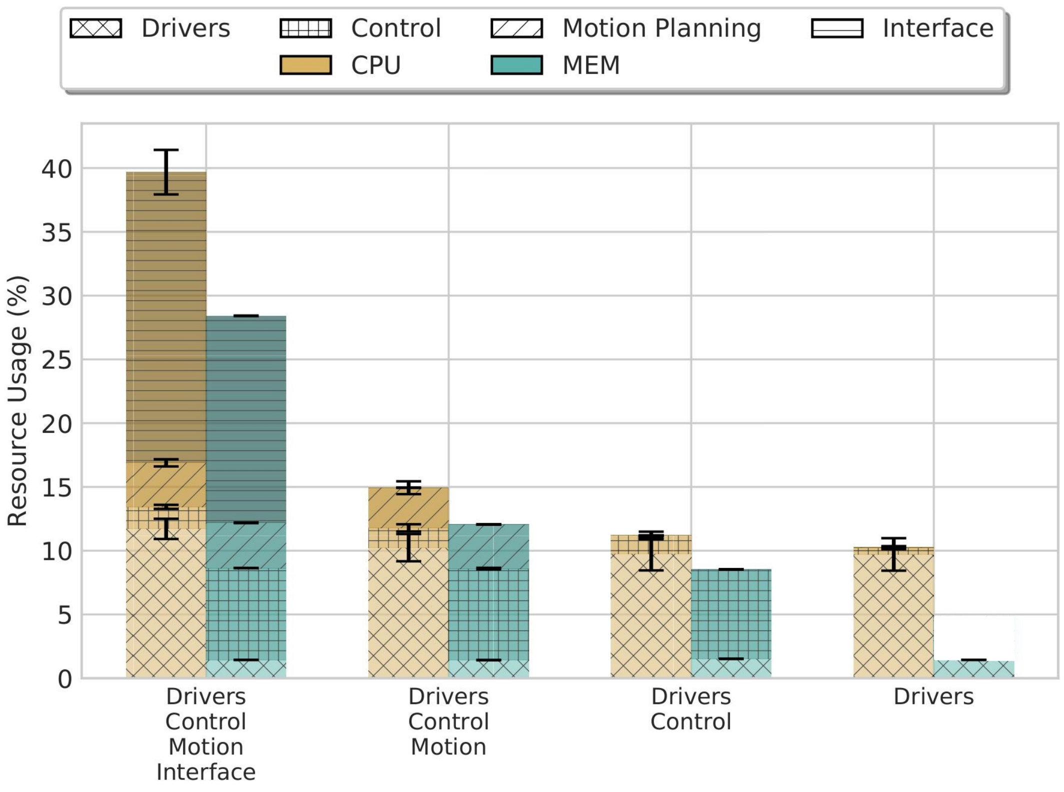


FIGURE 5-20 CPU and RAM usage for the Digital twin modules when deployed on the robot

Figure 5-20 presents the average CPU and RAM usage for all the deployment configurations while processing navigation commands. It can be seen that the CPU consumption of the whole Robot Stack (Control, Motion and Interface) is approximately 30% of the total CPU consumption on the Raspberry Pi 3 with a 1.2 GHz CPU. In particular, 23% of the CPU is being used by Interface VNF, around 5% by the Motion Planning VNF and 2% by the Control VNF. The Interface VNF has the highest computational requirements needed to perform the command translation, validation and real-time

synchronization between the Digital Twin and the ROS system. For what concerns the Motion planning VNF, the low CPU consumption is because of the simple path planning in our experiments. Finally, the Control VNF CPU consumption is mainly due to the controllers that it implements to run a control loop towards the Robot Drivers.
Regarding the MEM usage, the Interface VNF consumes approximately 16% of the total RAM usage, while the Motion Planning and Control VNFs around 4% and 7% respectively. The 16% of MEM used by the Interface VNF is partially due to the robot action server that handles the Digital Twin concurrent requests, checks if the command can be processed, validate parameters and calls required controllers (e.g., Motion Planning VNF or Control VNF). The MEM usage of Motion Planning and Control VNF is smaller since the robotic arm receives simplified navigation commands (small offset of 0.01 between two separate commands) from the Interface VNF.

5.1.4.2. Control VNF scaling

In the second set of experiments, we focused on the Control VNF because it is the most time-sensitive function that implements a generic control-loop feedback mechanism used for robot manipulation. We investigated how does the CPU usage of the Edge node that hosts the Control VNFs, influences the Digital Twin performance.

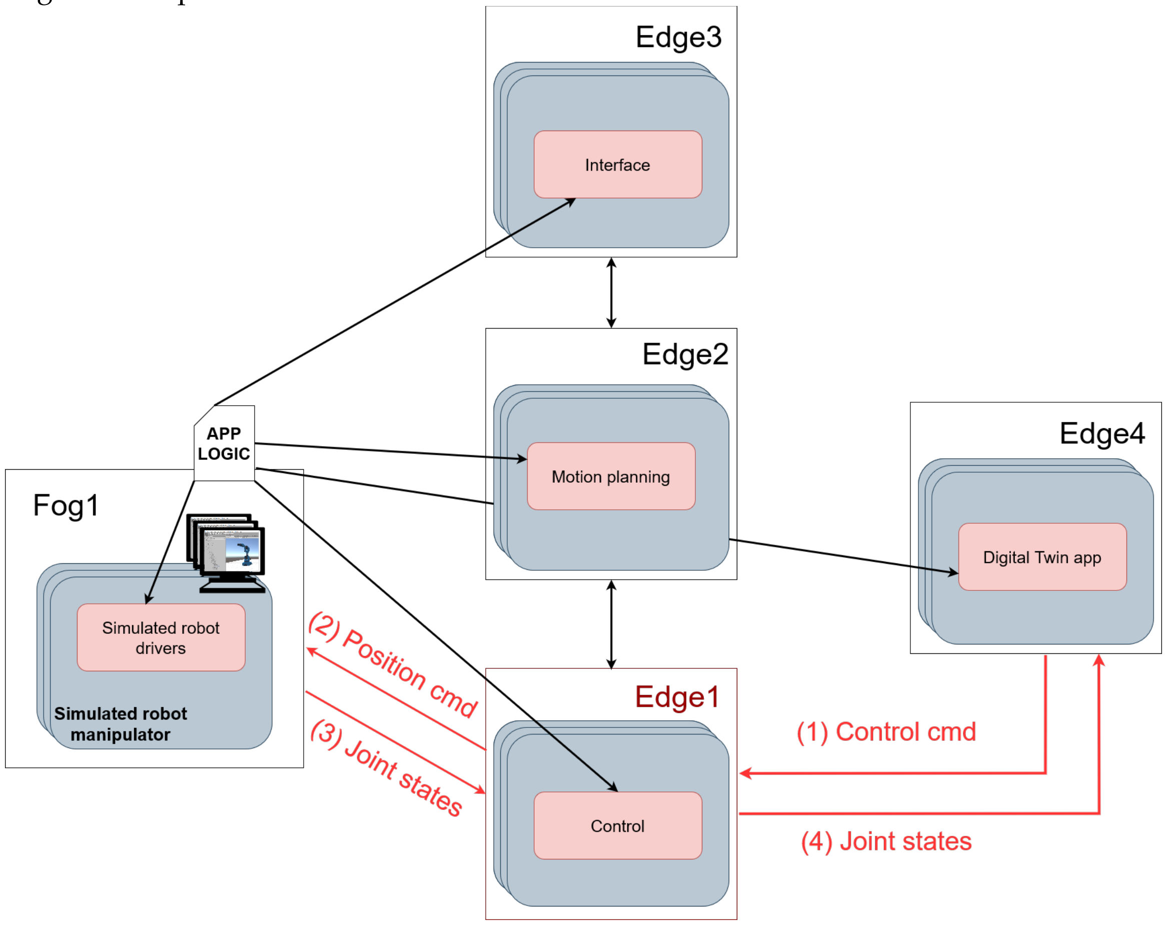


FIGURE 5-21: Testbed setup

Figure 5-21 shows the experimental testbed that we created in 5TONIC. The testbed is composed of: i) 1 Fog node (Fog1), equipped with 2nd gen Intel Core i7 and 6GB RAM, ii) 3 Edge nodes (Edge2, Edge3 and Edge4) implemented as virtual machines equipped with 3vCPUs and 8GB of RAM and iii) 1 Edge node (Edge1) implemented as a virtual machine equipped with 1vCPU and 8GB of RAM. All the nodes are interconnected with 1Gb Ethernet connection and are configured to be part of a Docker swarm. Fog1 node hosts the Simulated Robot Drivers VNF, while Edge1, Edge2, Edge3 and Edge4 host the Control, Motion Planning, Interface and Digital Twin VNFs respectively. The VNFs are implemented as Docker containers.

The experiment starts when an application logic that is hosted in the Fog node deploys a Digital Twin instance over the Docker swarm. Once the Digital Twin instance is deployed the Digital Twin VNF will start controlling the simulated robot in a closed-loop fashion as presented in Figure 5-21. (Red arrows). (1) The Digital Twin sends a control command with offset of 0.01 rad and command execution time of 20ms to the Control VNF of the Robot Stack. (2) The Control VNF translates the control command into position commands and sends them towards the Robot Driver VNF. The Robot Driver VNF executes the received position command, moves to the instructed position and generates a joint state message that through the Control VNF (3) is propagated to the Digital Twin (4) as a validation for successfully executed command. It is worth mentioning that the messages of the joint state are generated in an open loop fashion every 20ms independently if the robot changed the position or not. The application logic adds a new Digital Twin instance every 5 minutes and each instance is following the described workflow.

Throughout the duration of the experiment, in the Edge4 node, we obtained the Digital Twin control-loop time, Digital Twin update time and Digital Twin command loss rate, while in the Edge1 node we obtained the CPU load. Finally, in the Fog node the Digital Twin command time. The collected metrics are defined as follows:

- The Digital Twin control loop is the time elapsed from when the Digital Twin VNF sends a control command until the corresponding joint state message that validates the successful command execution is received in the Digital Twin VNF.
- Digital Twin update time is the time elapsed from when a joint state message is generated by the Robot drivers until the message is received by the Digital Twin VNF.
- Digital Twin command loss is the ratio of total number of commands send and the total number of commands that are not executed by the Robot Drivers. A command is considered to be lost if a joint state message is not received in the period of 400ms.
- Digital Twin command time is the time elapsed from when a command is sent from the Digital Twin VNF until the command is received by the Robot Driver VNF.

The obtained data from the first Digital Twin instance is analysed and aggregated to generate the results presented in Figure 5-22 Note that the application logic was adding a Digital Twin instance every 5 minutes until 20 instances were present in the system.

The top graph shows the average CPU load of the Edge1 node that hosts the Control VNFs. A first observation is that the average CPU load increases linearly by approximately 7% when a new instance enters the system. It can be seen that for up to 11 robots (61% of the average CPU load of Edge1), the Digital Twin instance works correctly reporting stable average values of around 40ms (20ms for command execution + 20ms for generating a joint state update) for the control-loop, and 1ms for the update and command time with 0.04% of command loss. The update and command time are 1ms due to the fact that we are using Ethernet as our underlying technology. Please note that update and command time values do not represent the frequency at which joint states and commands are being generated. These values are the propagation and processing delay of the service upstream and downstream.

As we continue to increase the Digital Twin instances in the system, in the interval from 11 to 15 Digital Twin instances, we can notice an increase in the update and command time that also result in increased variability of the control-loop of and increased number of lost commands. This is due to the fact that as the CPU load of the Edge1 increases, the processing latency of the Control VNF will increase and this will result in increased update and command times. The increased number of lost commands is because the Robot Drivers VNF will discard all the delayed control commands and with this degrade the performance of the Digital Twin service.

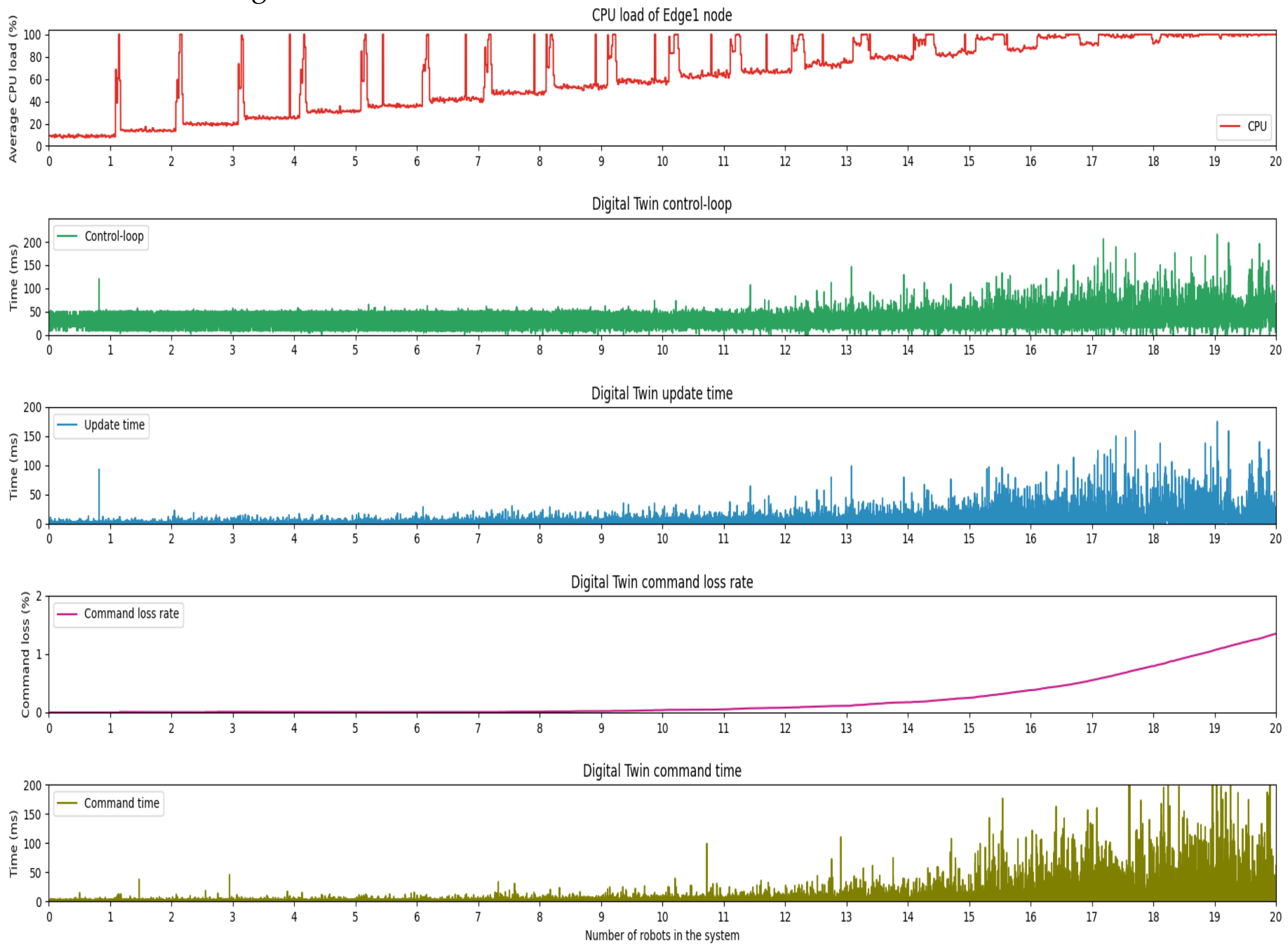


FIGURE 5-22: Edge node overall CPU usage, Digital Twin control loop, update and command time and command loss rate

Finally, when the CPU load of the Edge1 node is above 80% (from 15 to 20 Digital Twin instances) we can notice that the command loss starts to increase exponentially with command and updates times riching values of up to 200ms. In this interval, the Digital Twin service is completely degraded resulting in frequent stop and move behaviour of the robot.

As concluding remarks, the results show how the CPU load of the Edge node that hosts the Control VNF have direct implications on the Digital Twin control-loop, update time, command time and command loss rate. The results give hints into how the Control VNF does behaves under different CPU loads and they can be used by the SLA manager in the BASS in order to trigger scaling of the Edge resources or even migration of the Control VNF.

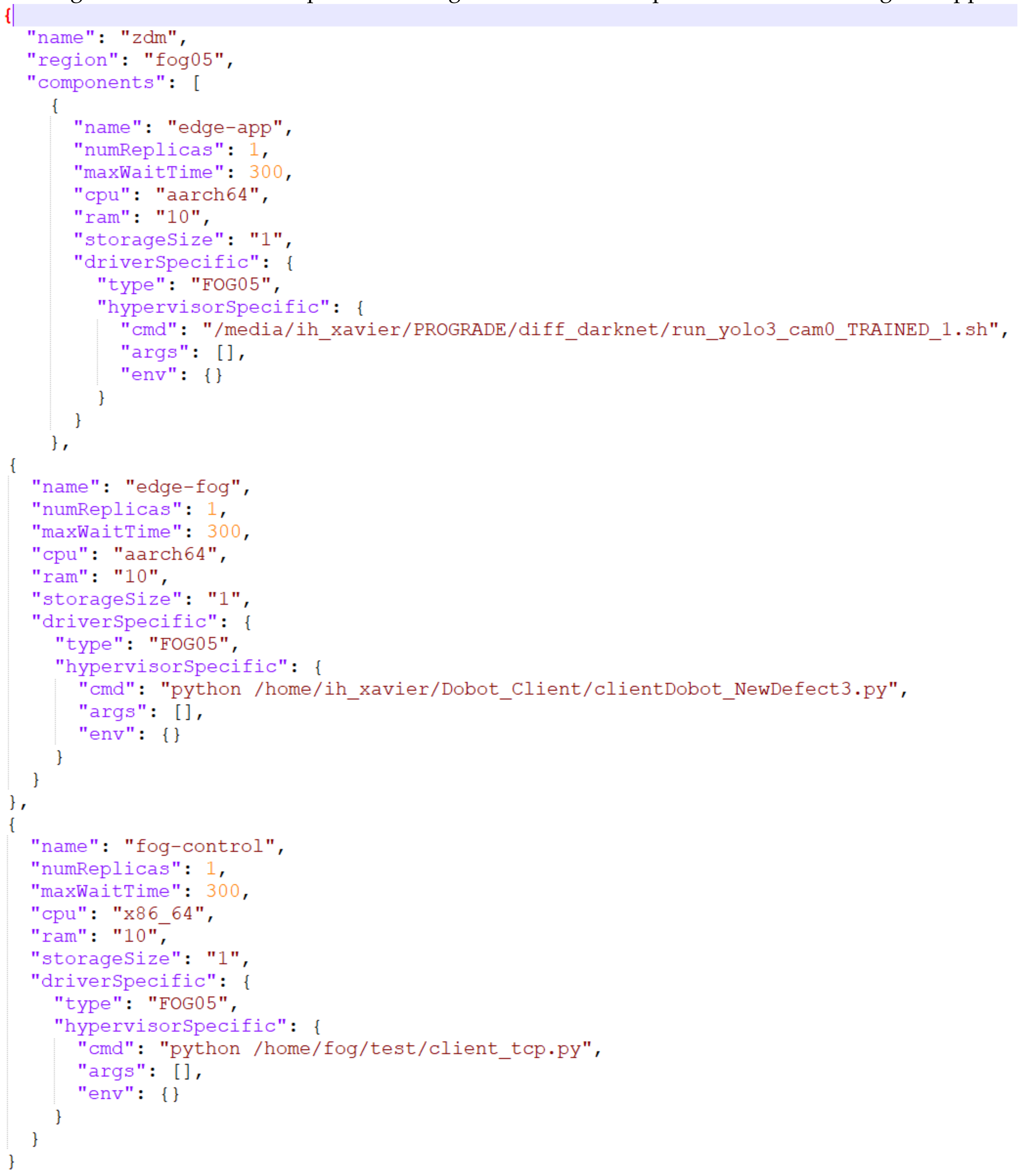
5.2. I4.0-UC2: ZDM

The ZDM use case evolution from its initial stages has been reported in D2.3 [7]. A new object detection engine has been described and the ZDM use case has been integrated with AWS Wavelength, Amazon's Edge Computing services (telco-Edge), where the ZDM object detection runs. In this section, the final ZDM setup is described, as well as the setup of the different experiments to evaluate it, and the experimental results obtained, both for the use case and for the DEEP integration. The DEEP integration subsection describes the integration of the ZDM functionalities with the DEEP platform. Finally, present a summary of the section, scalability aspects of the ZDM are addressed.

5.2.1. DEEP Integration Validation

The ZDM deployment is automated through the BASS using Eclipse Fog05 driver for the Fog and Edge nodes. The BASS offers the benefits of deploying native applications on the physical entities at the factory floor through the Fog05 driver. This feature of the BASS is useful for the ZDM use case applications because the physical devices, i.e., the robots and components of the factory, are customer-specific hardware that has proprietary libraries and programing interfaces that do not support virtualization. In order to deploy the ZDM services, a remote operator fills the BASS VSD that contains abstracted information about the modules that is part of the service. Screen capture of the vertical service descriptor is seen in Figure 5-23.

Once the VSD is completed and submitted through the BASS web-view, all the applications will be instantiated. Figure 3-1 shows the BASS upon successful deployment and instantiation. After this step the Digital Twin service is up and running and the remote operator can start using the application.



```
{
  "name": "zdm",
  "region": "fog05",
  "components": [
    {
      "name": "edge-app",
      "numReplicas": 1,
      "maxWaitTime": 300,
      "cpu": "aarch64",
      "ram": "10",
      "storageSize": "1",
      "driverSpecific": {
        "type": "FOG05",
        "hypervisorSpecific": {
          "cmd": "/media/ih_xavier/PROGRADE/diff_darknet/run_yolo3_cam0_TRAINED_1.sh",
          "args": [],
          "env": {}
        }
      }
    },
{
  "name": "edge-fog",
  "numReplicas": 1,
  "maxWaitTime": 300,
  "cpu": "aarch64",
  "ram": "10",
  "storageSize": "1",
  "driverSpecific": {
    "type": "FOG05",
    "hypervisorSpecific": {
      "cmd": "python /home/ih_xavier/Dobot_Client/clientDobot_NewDefect3.py",
      "args": [],
      "env": {}
    }
  }
},
{
  "name": "fog-control",
  "numReplicas": 1,
  "maxWaitTime": 300,
  "cpu": "x86_64",
  "ram": "10",
  "storageSize": "1",
  "driverSpecific": {
    "type": "FOG05",
    "hypervisorSpecific": {
      "cmd": "python /home/fog/test/client_tcp.py",
      "args": [],
      "env": {}
    }
  }
}
```

FIGURE 5-23. ZDM use case VSD file

5.2.2. Experimental Results

In D3.2 (D3.2, 2021), the connection between the production line and the edge side was reported for two connectivity options, namely Wi-Fi and 4G LTE. The main measurements carried for both connectivity options covered the bandwidth and latency measurements. In this deliverable, E2E 5G connectivity between the Fog node and the Edge node is reported. These results are taken from a pilot reported in the project with Vodafone Uk and Amazon Web services, where the Edge computing resources are located within the Vodafone UK CN (telco-edge). Specifically, the Fog node is directly connected to the 5G modem at the production line side, whereas the Edge node is directly connected to the Vodafone 5G network. Two main measurements are performed:

Network-Layer Measurements

In this deliverable, E2E 5G connectivity between the Fog node and the Edge node is reported. Specifically, the Fog node will be directly connected to the 5G modem at the production line side, whereas the Edge node will be directly connected to the 5TONIC 5G network. Two main measurements will be performed:

- Bandwidth measurements: using the IPERF tool (GUEANT, 2021) both in the DL (Edge to Fog) and in the UL (Fog to Edge). This provides the measurements of the available BW for the video stream in the UL and the control signalling in the DL:
    - Uplink: Iperf
    - Downlink: Iperf
- Latency measurements: Using the PING tool between the Fog and Edge nodes.
    - Ping (Fog-Edge)

Application-Layer Measurements

In addition to the connectivity measurements, the application-level communication between the edge and the fog side is further investigated in this deliverable. The ZDM use case relies on video streaming and an object detection engine at the edge. The application layer protocol used in the ZDM is NDI. NDI requires extra time for particular tasks to be executed before a video frame can be transmitted over a network. There are, therefore, task-related factors that can influence the latency of video streams, on top of the delivery network used itself. These factors include video encoding pipeline duration, ingestion and packaging operations, video segment length, user policies on buffering and resilience, video codec used, compression and decompression of the video frames, the number of frames per second, etc. The added latency of these factors needs to be factored in in a use case like ZDM that relies on fast defect detection and consequent automated actions.
Specifically on the ZDM setup, measurement software is employed at both the fog and edge nodes, where both devices are time-synchronized via time input from the same NTP server. Additionally, the measurement software has in built functions where both fog and edge nodes communicate and synchronize their CPU time further. The measurement software is executed using two devices, local and remote, where the local device is the fog device located at the production line, while the remote device is the edge side, as shown in Figure 5-24.

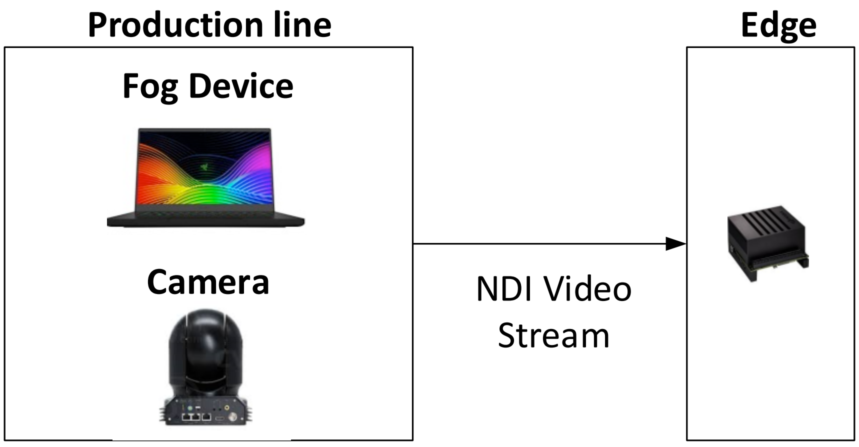


FIGURE 5-24. Application-level measurement software

The camera employed in the ZDM use case uses a specific network protocol to convey the video that is the Network Device Interface (NDI). This allows the high-definition video to be streamed over the network in real-time. The software uses fog and edge devices in order to monitor the NDI video frames streamed from the production line side to the edge side over the 5G network. Using this software, the main KPI of interest to the ZDM use case was measured, the latency of the video. The achieved results are summarized in Table 5-5.

TABLE 5-5 Network and Application layer measurement

	Measured KPI	Video settings	
		720p50	1080p60
Network layer	Latency	11ms	
	Throughput	35.5Mbps	
Application Layer	Latency	473.6ms	564.7ms

The results from the pilot validate the ZDM use case and were taken over intervals of 5 minutes over different days. At the network layer, 11ms latency is a very good latency for tasks performed in the ZDM. The measured throughput in the uplink meets and exceeds the requirement in D1.1 (D1.1: 5G-DIVE architecture and detailed analysis of vertical use cases, 2020). The Application layer latency is sufficient because every object is monitored over a few seconds.

5.2.2.1. Experiment A: Bandwidth usage optimization: Adaptive video settings

The ZDM use case relies very strongly on the transmission of the video stream from the ZDM factory, towards the Edge side. The camera used in the setup is capable of outputting a full High Definition (HD) video stream. The quality associated with HD video has a 5 Mbps UL Bandwidth requirement. This requirement is indicative of factors that can influence the required bandwidth such as the number of frames per second (fps), video encoding method, etc. . It is, therefore, an important challenge to meet this requirement in a 5G network – the network could easily provide the required bandwidth if the video stream was the only traffic running through it. Because other services and traffic are served by the same network, and because the ZDM relies on the video stream reception at the Edge side, it makes sense to address bandwidth consumption aspects in the context of the ZDM.

To address the scalability aspects of the ZDM use case, a simulated, larger factory environment is envisioned. In this scenario, multiple stages in a production line process are considered. The monitoring of all the production stages is an effective way of improving predictive maintenance aspects. Predictive maintenance reduces maintenance frequency to certain minimum values and keeps resources in good condition, leading to cost savings (Mohammed M. Mabkhot, 2018) Besides this, a failure in the production line equipment at the earlier stages of the process might cause a halt to the entire production line – factory production lines are usually a sequential process (Mohammed M. Mabkhot, 2018) .In smart factories, data is collected about the resources and products, to then be analysed in real-time, at all stages of the production process (Mohammed M. Mabkhot, 2018). As a result, equipment status and failures can be forecasted, patterns can be drawn, and a suitable maintenance strategy can be tested and evaluated.

Predictive maintenance is therefore a key aspect of smart factories and the ZDM scalability study relates to it. There are, however, some limitations on the ZDM use case. Its simulated factory environment can be seen as the output or the last stage of the production process of the factory. Obviously, in a real factory environment setup, the production process would have more steps involved, and the steps can also be monitored via filming and streaming of video. Hence, in this section, scaling up on the factory process via video monitoring is addressed. It would be unfeasible to increase the number of cameras in the real ZDM setup. Therefore, a study is presented simulating a larger factory environment, where multiple steps in a production line are being filmed and their video feed is being streamed towards the Edge.

For the bandwidth evaluation, the ZDM use case makes use of InterDigital's proprietary software that is able to measure packet statistics between two ends connected with a Content Delivery Network (CDN). The SW measures statistics at the application layer, providing an understanding of the QoE for the measured traffic. This is an important aspect to be measured because of the need for the Yolo [12] detection engine to receive the video packets properly, which can only be considered after the packets become an input to the engine. Bandwidth measurements were performed for different video settings in the camera and are presented in Figure 5-25. The measurements were performed for four different camera settings for resolution and fps. The x-axis represents relative time starting from the initial video stream capture moment by the receiving end SW component, in minutes.

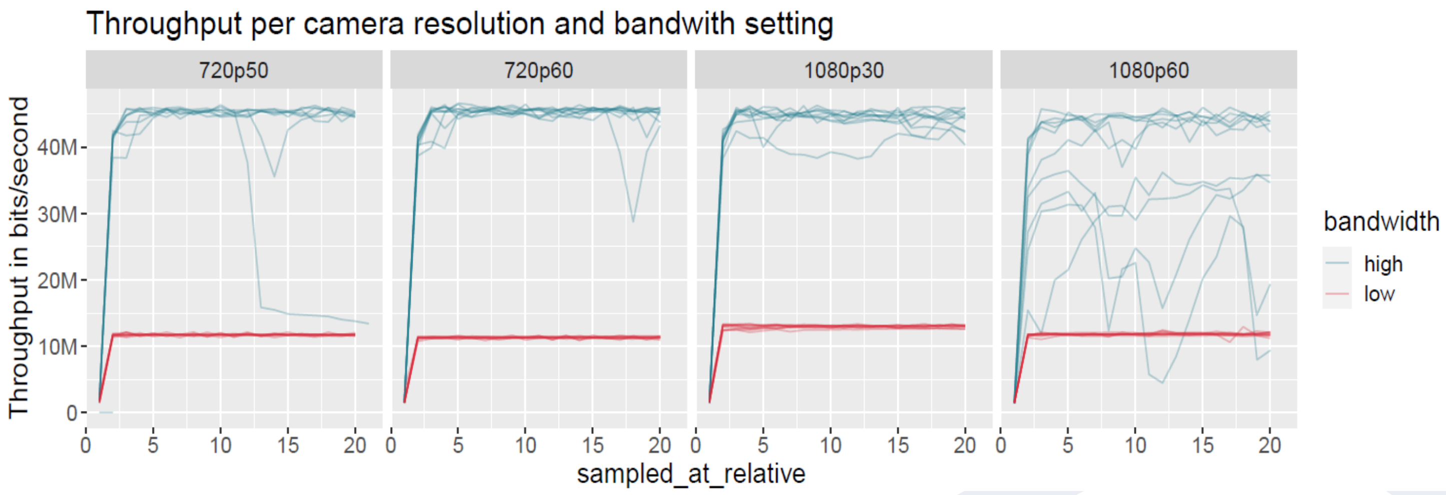
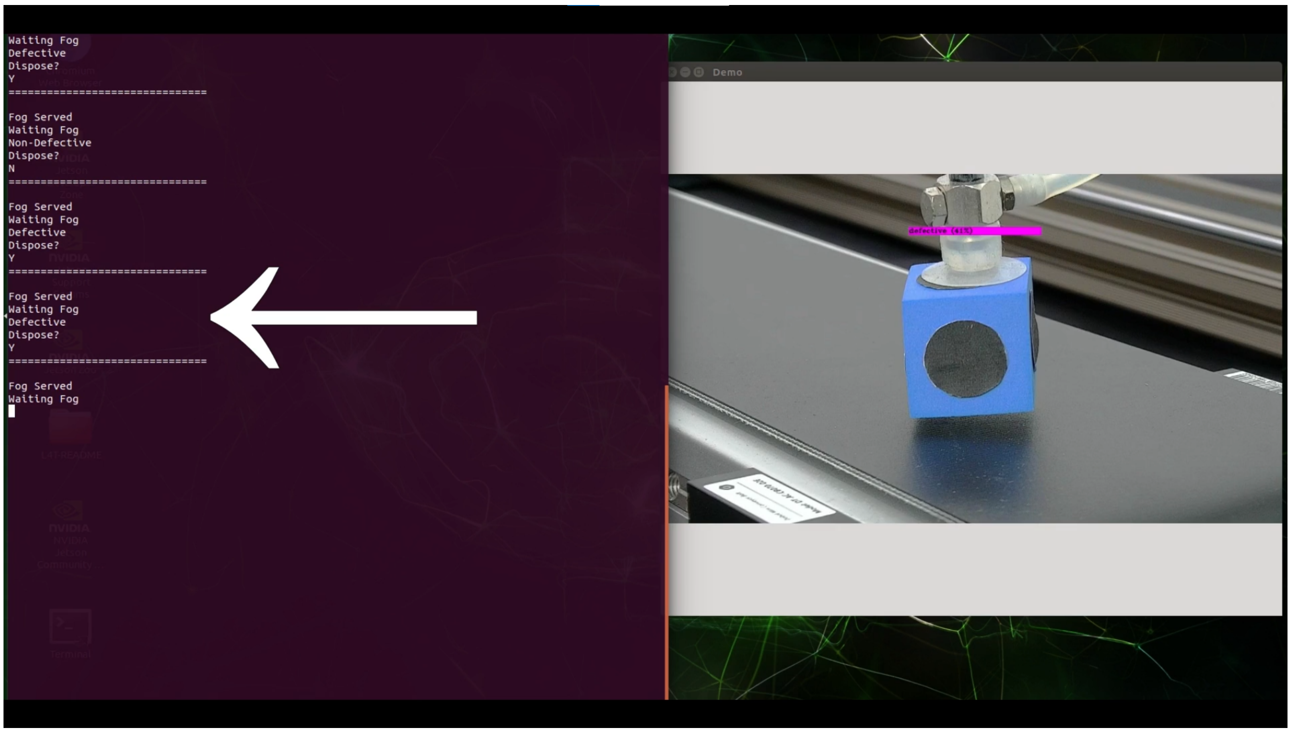


FIGURE 5-25 ZDM video stream initial throughput assessment

The presented results show two lines per video setting. The NDI library allows choosing a "high" or "low" bandwidth setting. In practice, this means that a high setting will try to encode and transmit the desired video settings (the ones represented in the figure). The low setting is an NDI feature that will automatically set the video settings to a certain resolution and number of frames per second, that the NDI protocol deems suitable, given the current network conditions, in order to maximize the video experience. The figure shows this very clearly, as the red line displays a much lower throughput than the high setting for all four possibilities. It can be seen as well that, when the number of fps is set to 30 and 50, higher throughput results are achieved with the "low" adaptive setting. The highest quality setting provided by the camera is 1080p60. The results obtained for this particular setting show that there is a throughput degradation, implying that the connection can sustain the video in the uplink, but there are video frames dropped, which is much less noticeable with any of the other three video settings.

This is a very important aspect to look further into when considering the scalability aspects of the ZDM use case. Its setup is designed so that cubes with a defect mark are placed in the conveyor belt (D2.3: Final Specification of 5G-DIVE Innovations, 2021), on their way to a) out of the factory or b) to a disposal bin. Because of the nature of this design, and as previously shown in the available 5G-DIVE ZDM demonstrations, the camera is placed in a way that it can capture part of the conveyor belt, and the cubes themselves while running on top of it. In practice, this results in just a few moments when the cube is captured in the video when it is running on a specific area of the conveyor belt to where the camera points to, as illustrated in Figure 5-26.



```
Waiting Fog
Defective
Dispose?
Y
==============================

Fog Served
Waiting Fog
Non-Defective
Dispose?
N
==============================

Fog Served
Waiting Fog
Defective
Dispose?
Y
==============================

Fog Served
Waiting Fog
Defective
Dispose?
Y
==============================

Fog Served
Waiting Fog
```

FIGURE 5-26 Defective object on displayed on the edge screen

In practice, the specific small time window where the cube is captured on the conveyor belt is the only important part of the video stream, because those are effectively the only video frames that are relevant for the Yolo object detection. When there is no cube being captured by the camera, the Yolo engine will not be able to perform any detection, because it doesn't have any relevant frames (with cubes) to work with. This realization allows concluding immediately that the ZDM use case (and any other similar application relying on a video stream), has a very high unnecessary bandwidth consumption, considering all the time that there is no object being captured on the video and the video stream still remains with a very high-quality setting. Ideally, the camera would have its video quality settings tuned to the highest possible setting when an object is captured only (and the video stream is transmitted to the Yolo engine), and set to the lowest possible settings (or even switched off), when there is no cube being captured. This would allow for significant bandwidth, energy, and generalized network resource savings.

To address the bandwidth usage, a new IE has been developed for the ZDM. The telemetry framework in place for the use case allows for the collection of several telemetry parameters from several setup components (D2.3: Final Specification of 5G-DIVE Innovations, 2021). Part of the telemetry data collected from the Yolo engine was recorded and used to create a dataset where an example entry is given in Table 5-6.

TABLE 5-6 Yolo engine telemetry collection example data point

Parameter	Value
Time	25/05/2021 09:40:48,600454
Detection result & confidence	NON-DEFECTIVE & 99%
Bounding box coordinates in the frame	x : 758 || y : 3 || w : 517 || h : 312
Input timestamp	25/05/2021 09:40:48,610139
Required Detection time (s)	0.020067453384399414
Detection timestamp	25/05/2021 09:40:48,630243
Number of objects detected	1

The telemetry framework records amongst others, the input timestamp, that reflects the timestamp when the first frame used for object detection was used by the Yolo engine. It records as well the required detection time, i.e., the time that was required for the Yolo engine to conclude on the detection result, also shown in the above table. The recorded value (in Table 5-6) is 0.020067453384399414 s, and the average on the entire recorded dataset is 0.020s. The availability of these two telemetry parameters on the dataset was used to train an ML model that is capable of predicting the next timestamps for the first frame of a given object to reach the Yolo engine from the factory. The training of the model is based on a time series forecasting approach where the confidence values of the telemetry collected from the Yolo engine were used as the predicted feature – detection confidence values are high when the engine is performing detection (i.e. when the object is displayed on the screen at the Edge), and low when there is no object present. Figure 5-27 shows the time series predictions on the test set in purple and the predicted future confidence values.

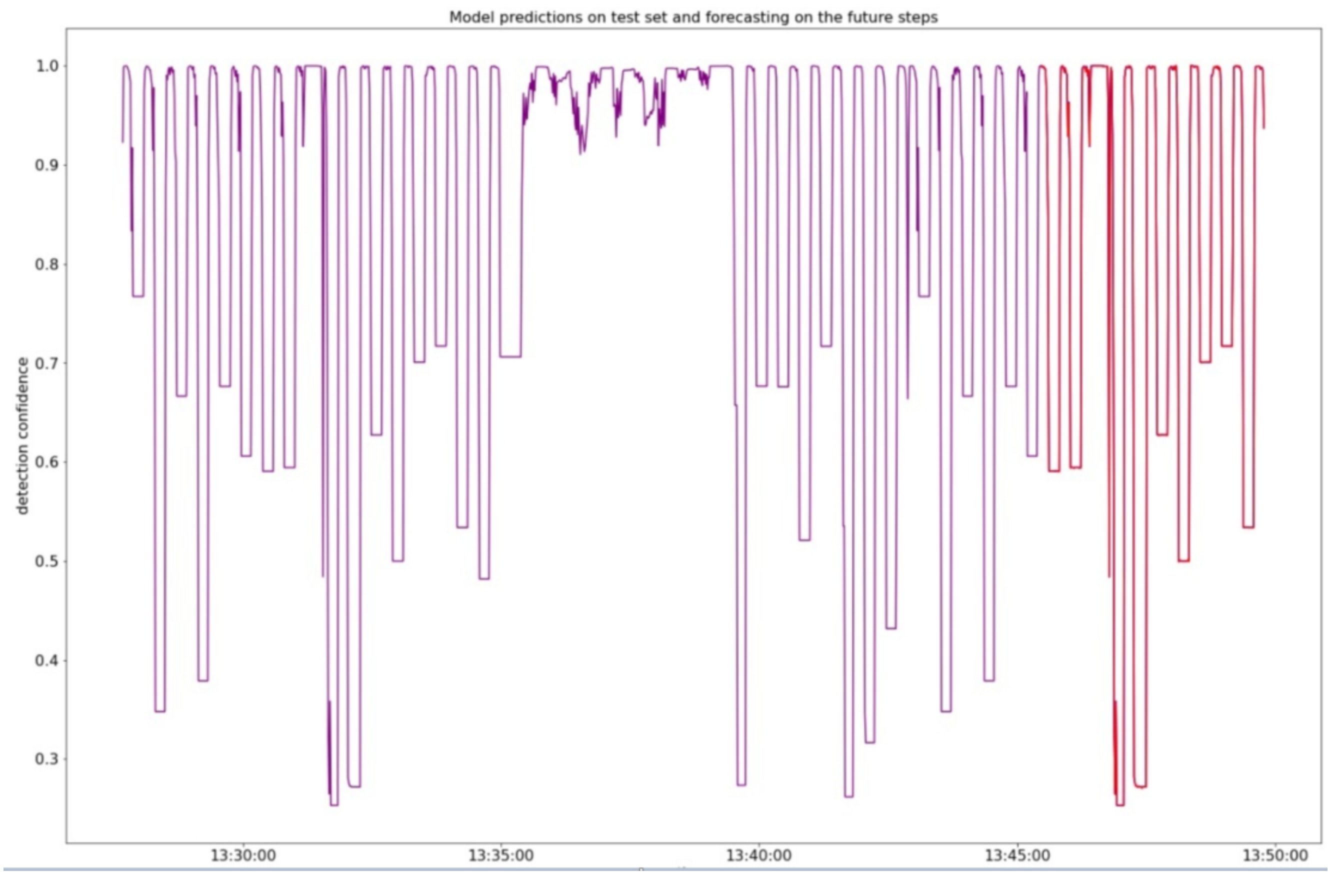


FIGURE 5-27 Training and predicted object detection confidence values

As mentioned, the best way to scale up the ZDM use case is by envisioning a larger scale factory. However, the HW limitations are obvious. This IE has therefore been developed to demonstrate that the underlying 5G connectivity needs re-dimensioning for a larger scale factory. The recorded measurements show that high-definition video can be close to the limits of a commercial 5G connection and that transmitting video streams with these fixed settings constantly is a waste of resources that will lead to a much denser deployment of the 5G infrastructure. With adaptive settings from the camera side, and ML predictions to assess when there is a need for high-quality video (20ms as input for Yolo), and low settings (or even video off) for the remainder of the time, large energy and bandwidth savings can be achieved, and the dimensioning of the 5G connectivity for a large scale factory would be executed with generalized lower requirements, resulting in the same QoE levels.

5.2.2.2. Experiment B: Access Traffic Steering, Spliting and Switching xApp

In D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021), an ATSSS xApp has been described to enhance the available throughput for the camera to transmit its video without risking dropped frames that can be important for a defect detection or predictive factory maintenance aspects. This section describes how the xApp works and some achieved results that were obtained with its usage.

The results obtained with the xApp consider that both 3GPP and non-3GPP access traffic are available at all times. Two machine learning models have been trained based on telemetry collected from a Wi-Fi router and a 5G UE.
The Wi-Fi router used was a Belkin RT3200 with an Open source, linux based OS named OpenWRT (OpenWRT, n.d.) flashed into it. OpenWRT allows for telemetry data collection from the router, collecting statistic related to CPU load, quality of the radio signals, available throughput, amongst others. This data has been collected and used to train a model that predicts future available Wi-Fi throughput. Figure 5-28 future available wi-fi throughput shows the predicted future Wi-Fi available throughput for a 10 seconds time window.

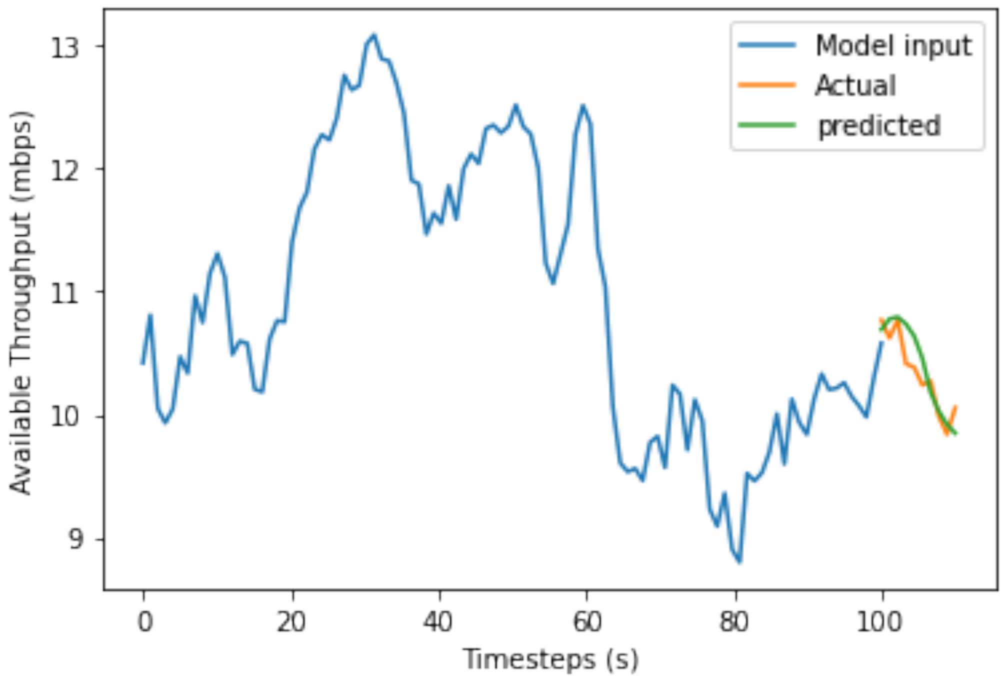


FIGURE 5-28 FUTURE AVAILABLE WI-FI THROUGHPUT

The 5G UE telemetry was obtained from a dataset available online in (5Gdataset, n.d.). The model also predicts future throughput, but in this instance for a 5G cell. Figure 5-29 FUTURE AVAILABLE 5G THROUGHPUT shows the predicted future available throughput from a 5G cell for the same 10 seconds time window.

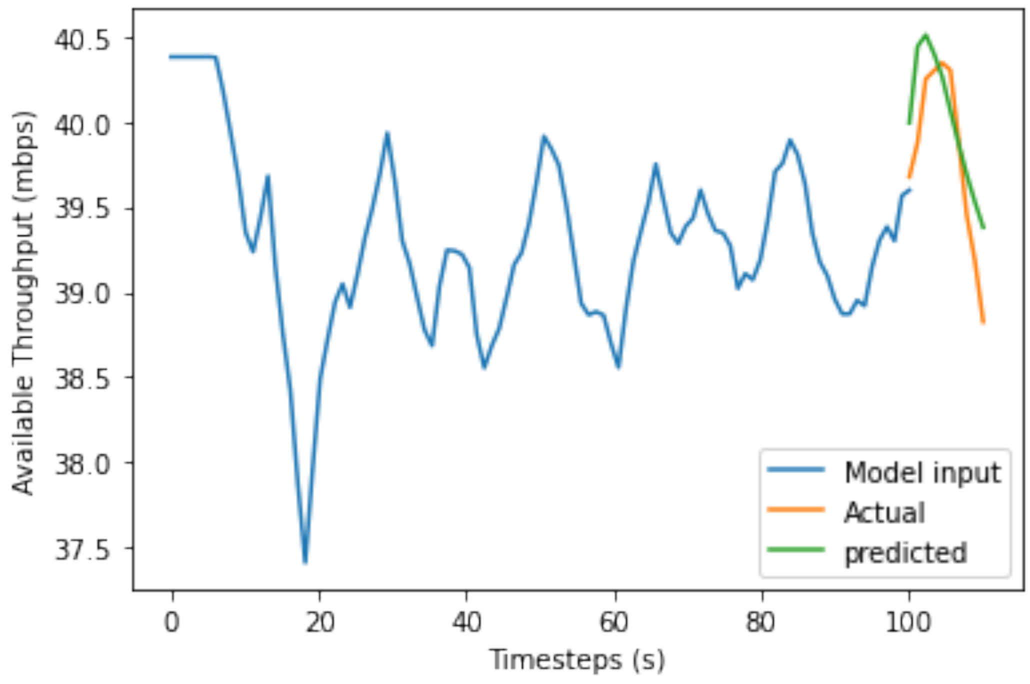
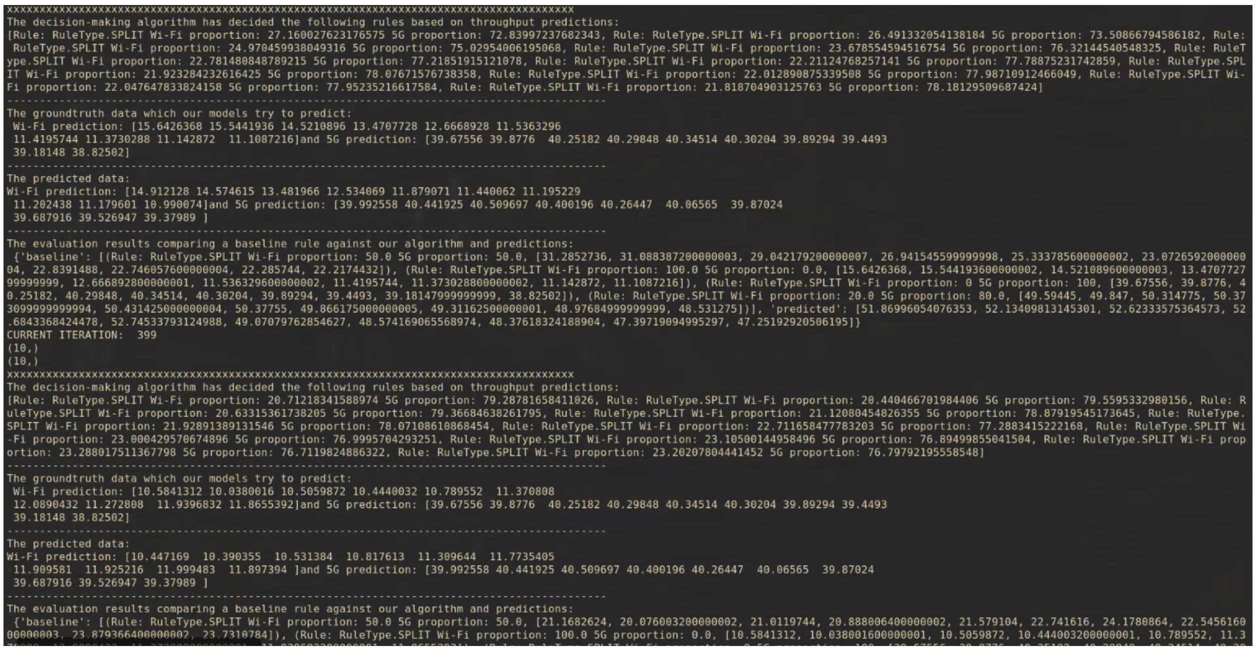


FIGURE 5-29 FUTURE AVAILABLE 5G THROUGHPUT

As both models can predict the future available throughput for a 10 seconds window, these predictions are then fed into the ATSSS xApp, that then has all the information to decide on e.g., the splitting rules in case of overlapping coverage of both 5G and Wi-Fi. Figure 5-30 Example output of ATSSS decision making shows an example output of the ATSSS decision making, where the predict throughput values for both Wi-Fi and 5G are displayed, along with the actual or true values, for performance assessment, and the final splitting rule, i.e., the traffic percentage allocation that is done for 5G and the same for Wi-Fi.



```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
The decision-making algorithm has decided the following rules based on throughput predictions:
[Rule: RuleType.SPLIT Wi-Fi proportion: 27.160027623176575 5G proportion: 72.83997237682343, Rule: RuleType.SPLIT Wi-Fi proportion: 26.491332054138184 5G proportion: 73.50866794586182, Rule:
 RuleType.SPLIT Wi-Fi proportion: 24.970459938049316 5G proportion: 75.02954006195068, Rule: RuleType.SPLIT Wi-Fi proportion: 23.678554594516754 5G proportion: 76.32144540548325, Rule: RuleT
ype.SPLIT Wi-Fi proportion: 22.781480848789215 5G proportion: 77.21851915121078, Rule: RuleType.SPLIT Wi-Fi proportion: 22.21124768257141 5G proportion: 77.78875231742859, Rule: RuleType.SPL
IT Wi-Fi proportion: 21.923284232616425 5G proportion: 78.07671576738358, Rule: RuleType.SPLIT Wi-Fi proportion: 22.012890875339508 5G proportion: 77.98710912466049, Rule: RuleType.SPLIT Wi-
Fi proportion: 22.047647833824158 5G proportion: 77.95235216617584, Rule: RuleType.SPLIT Wi-Fi proportion: 21.818704903125763 5G proportion: 78.18129509687424]
------------------------------------------------------------------------------------------
The groundtruth data which our models try to predict:
 Wi-Fi prediction: [15.6426368 15.5441936 14.5210896 13.4707728 12.6668928 11.5363296
 11.4195744 11.3730288 11.142872  11.1087216]and 5G prediction: [39.67556 39.8776  40.25182 40.29848 40.34514 40.30204 39.89294 39.4493
 39.18148 38.82502]
------------------------------------------------------------------------------------------
The predicted data:
Wi-Fi prediction: [14.912128 14.574615 13.481966 12.534069 11.879071 11.440062 11.195229
 11.202438 11.179601 10.990074]and 5G prediction: [39.992558 40.441925 40.509697 40.400196 40.26447  40.06565  39.87024
 39.687916 39.526947 39.37989 ]
------------------------------------------------------------------------------------------
The evaluation results comparing a baseline rule against our algorithm and predictions:
 {'baseline': [(Rule: RuleType.SPLIT Wi-Fi proportion: 50.0 5G proportion: 50.0, [31.2852736, 31.088387200000003, 29.042179200000007, 26.941545599999998, 25.333785600000002, 23.0726592000000
04, 22.8391488, 22.746057600000004, 22.285744, 22.2174432]), (Rule: RuleType.SPLIT Wi-Fi proportion: 100.0 5G proportion: 0.0, [15.6426368, 15.544193600000002, 14.521089600000003, 13.4707727
99999999, 12.666892800000001, 11.536329600000002, 11.4195744, 11.373028800000002, 11.142872, 11.1087216]), (Rule: RuleType.SPLIT Wi-Fi proportion: 0 5G proportion: 100, [39.67556, 39.8776, 4
0.25182, 40.29848, 40.34514, 40.30204, 39.89294, 39.4493, 39.18147999999999, 38.82502]), (Rule: RuleType.SPLIT Wi-Fi proportion: 20.0 5G proportion: 80.0, [49.59445, 49.847, 50.314775, 50.37
3099999999994, 50.431425000000004, 50.37755, 49.866175000000005, 49.31162500000001, 48.97684999999999, 48.531275])], 'predicted': [51.86996054076353, 52.13409813145301, 52.62333575364573, 52
.6843368424478, 52.74533793124988, 49.07079762854627, 48.574169065568974, 48.37618324188904, 47.39719094995297, 47.25192920506195]}
CURRENT ITERATION:  399
(10,)
(10,)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
The decision-making algorithm has decided the following rules based on throughput predictions:
[Rule: RuleType.SPLIT Wi-Fi proportion: 20.71218341588974 5G proportion: 79.28781658411026, Rule: RuleType.SPLIT Wi-Fi proportion: 20.440466701984406 5G proportion: 79.5595332980156, Rule: R
uleType.SPLIT Wi-Fi proportion: 20.63315361738205 5G proportion: 79.36684638261795, Rule: RuleType.SPLIT Wi-Fi proportion: 21.12080454826355 5G proportion: 78.87919545173645, Rule: RuleType.
SPLIT Wi-Fi proportion: 21.92891389131546 5G proportion: 78.07108610868454, Rule: RuleType.SPLIT Wi-Fi proportion: 22.711658477783203 5G proportion: 77.2883415222168, Rule: RuleType.SPLIT Wi
-Fi proportion: 23.000429570674896 5G proportion: 76.9995704293251, Rule: RuleType.SPLIT Wi-Fi proportion: 23.10500144958496 5G proportion: 76.89499855041504, Rule: RuleType.SPLIT Wi-Fi prop
ortion: 23.288017511367798 5G proportion: 76.7119824886322, Rule: RuleType.SPLIT Wi-Fi proportion: 23.20207804441452 5G proportion: 76.79792195558548]
------------------------------------------------------------------------------------------
The groundtruth data which our models try to predict:
 Wi-Fi prediction: [10.5841312 10.0380016 10.5059872 10.4440032 10.789552  11.370808
 12.0890432 11.272808  11.9396832 11.8655392]and 5G prediction: [39.67556 39.8776  40.25182 40.29848 40.34514 40.30204 39.89294 39.4493
 39.18148 38.82502]
------------------------------------------------------------------------------------------
The predicted data:
Wi-Fi prediction: [10.447169  10.390355  10.531384  10.817613  11.309644  11.7735405
 11.909581  11.925216  11.999483  11.897394 ]and 5G prediction: [39.992558 40.441925 40.509697 40.400196 40.26447  40.06565  39.87024
 39.687916 39.526947 39.37989 ]
------------------------------------------------------------------------------------------
The evaluation results comparing a baseline rule against our algorithm and predictions:
 {'baseline': [(Rule: RuleType.SPLIT Wi-Fi proportion: 50.0 5G proportion: 50.0, [21.1682624, 20.076003200000002, 21.0119744, 20.888006400000002, 21.579104, 22.741616, 24.1780864, 22.5456160
00000003, 23.879366400000002, 23.7310784]), (Rule: RuleType.SPLIT Wi-Fi proportion: 100.0 5G proportion: 0.0, [10.5841312, 10.038001600000001, 10.5059872, 10.444003200000001, 10.789552, 11.3
```

FIGURE 5-30 EXAMPLE OUTPUT OF ATSSS DECISION MAKING

The models have then been given as input data, data that is part of the same dataset, but that hasn't been used for training purposes. The performance of the xApp has then been evaluated and it is depicted in Figure 5-31 achieved ATSSS xApp throughput for different splitting rules.

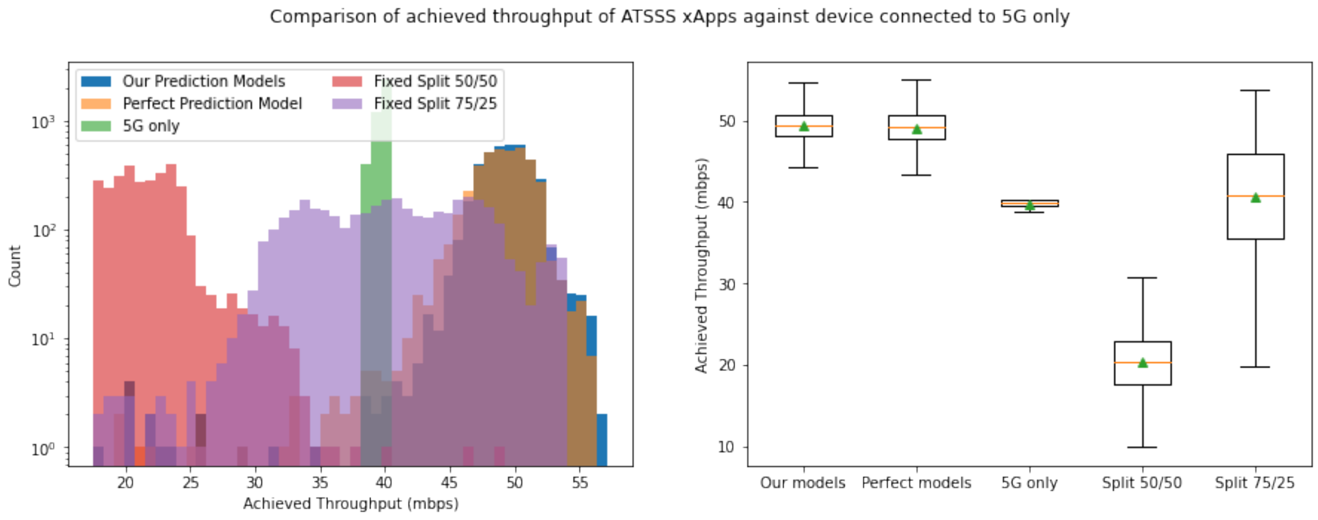


FIGURE 5-31 ACHIEVED ATSSS XAPP THROUGHPUT FOR DIFFERENT SPLITTING RULES

LEFT: ACHIEVED THROUGHPUT FOR DIFFERENT SPLITTING RULES

RIGHT: BOX PLOTS OF THE ACHIEVED THROUGHPUT FOR DIFFERENT SPLITTING RULES

The experiment was executed over 400 time steps. Every 10 time steps, the predictions from the trained models for Wi-Fi and 5G are generated and fed into the xApp that then dictates the splitting of the traffic. The achieved throughput is then measured and compared over different splitting rules, namely a fixed 50%-50% split, and a fixed 75%-25% split, where 75% of the traffic is allocated to the 5G cell. The baseline for the performance benchmark is the "5G only" split, where there is no traffic directed via the Wi-Fi access point. The Figure on the left shows as well results obtained with "our prediction models", and the "perfect prediction models". The former represents the enhancements introduce by the xApp, while the latter represents the possible achievable performance, in case both the 5G and Wi-Fi models had a 100% prediction accuracy. The baseline results obtained for the 5G only split are around 40 Mbps for the duration of the simulation. This has been chosen in the dataset to match the reported throughput at the network layer, reported in Table 5-5 Network and Application layer measurement, to match the experiment and demonstrated executed in partnership with Vodafone UK and Amazon. It is also a borderline available bandwidth for a 4K video streaming, which has a typical bandwidth requirement of 35 Mbps.

Results validate the predictive models for both 5G and Wi-Fi as the result of the steering rules with both trained and perfect models are very similar, and both outperform the 5G connectivity only option by a maximum of 45%. The 50%-50% split always performs worse than the baseline. The 75%-25% split sometimes outperforms the baseline but almost never outperforms our xApp. In roughly the same proportion, this split also performs worse than the baseline, achieving less throughput than the 5G cell only option. In addition to this, the right hand side figure also shows a much higher variance between minimum and maximum achieved throughput than the ATSSS xApp.

These results validate the idea that the ATSSS xApp can improve RAN functions and achieve higher available throughput for the camera, enabling better quality video streaming, higher reliability, and general QoE improvements.

5.3. I4.0-UC3: massive MTC

Figure 5-32 shows the system setup for the mMTC trial. We deploy one radio-head which connects to 7 IoT devices/nodes. The radio head connects to the 5TONIC Edge Data Center via a Gigabit Ethernet link or a 5G connectivity. In the 5TONIC Edge Data centre, 3 VMs are provided for mMTC use case. We configured the 3 VMs as a Kubernetes cluster hosting with 3 nodes (1 Master node and 2 Worker nodes), which host the edge services of mMTC use cases. The DEEP platform is also deployed in the same data centre, where the services of other use cases are also deployed. Therefore, all three I4.0 use cases are integrated in the same data centre infrastructure using the same DEEP platform for management and coordination, as well as performance monitoring. Specifically, in mMTC trial, we use BASS in DEEP, to manage the mMTC service comprising three microservices/containers, namely Contiki, RAN, and Fingerprinting, deployed in the Kubernetes cluster.

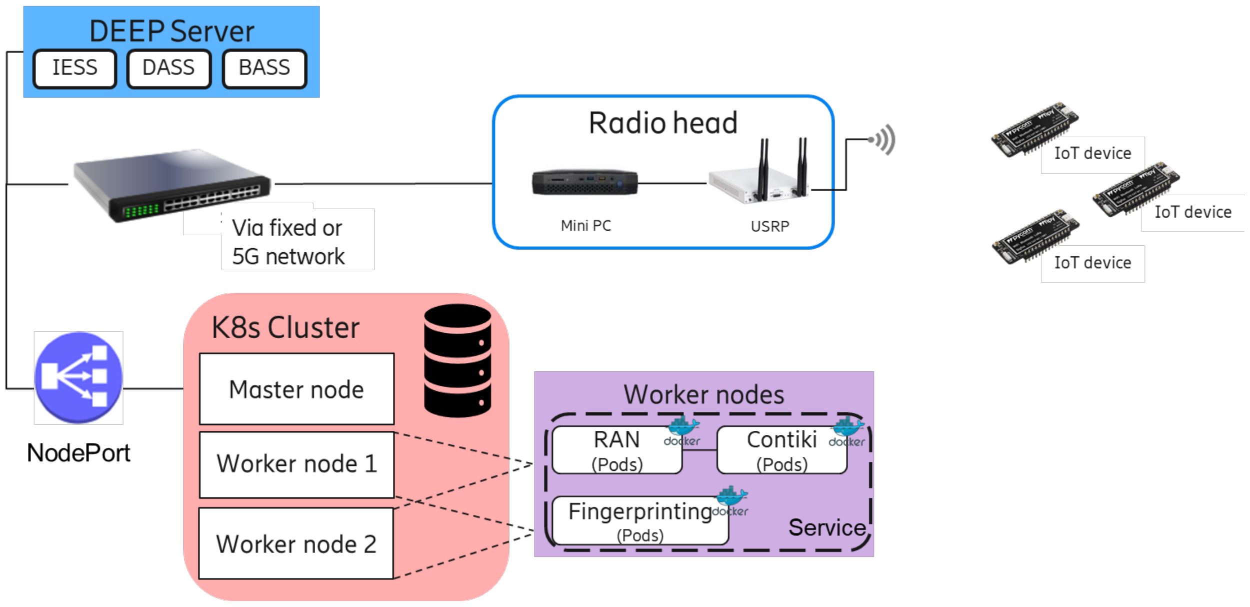


FIGURE 5-32 5TONIC trial setup of massive MTC use case

In the following, the functionality of each microservice deployed in the Kubernetes cluster is briefly described.

- **RAN:** The RAN instantiates an IEEE 802.15.4 O-QPSK 250kbps physical layer, which modulates and demodulates packets to and from the radio-head. It uses the ZMQ PUSH-PULL message pattern to communicate to the Contiki and the Radiohead.
- **Contiki:** As described in D 2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021), Contiki acts as the IoT networking stack. In our experiments, we use our asynchronous MAC as our MAC protocol. We use RPL (RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks) as our routing protocol. We configure the nodes to only act as leaf nodes to create a star topology network. The IP layer connects via a TUN interface to the host networking stack. We also integrated a modified Leshan (OMA Lightweight M2M server and client in Java) server as our LWM2M server that connects to the Contiki stack through the TUN interface. It manages the registration of the IoT nodes and subsequent application-layer communication to the IoT nodes. It also provides an HTTP server that translates HTTP requests to LWM2M requests.
- **Fingerprinting:** The fingerprinting module uses our Siamese network architecture introduced in D2.3 to identify the sender (IoT node) based on the IQ samples to enhance device-level security. The Fingerprinting module collects IQ samples from the radio head using ZMQ TCP. We use the RAN microservice (i.e. IEEE 802.15.4 PHY) to identify the starting and ending of a single packet from the IQ samples. Next, we parse the demodulated packet to collect the MAC address of the sending node from the received packet. We also check for the correctness of the data packet using the CRC. The fingerprinting module has a dictionary of reference samples for the nodes used in the trial. We collect the reference samples in our office in Stockholm by placing each node next to a USRP. Hence, they can be assumed to be free from noise and multi-path effects. An AI model based on deep neural network is trained in our office in Stockholm using IQ samples from the nodes in the experiments, using the training process outlined in D 2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021). Once the samples corresponding to a packet

are identified, the model compares the IQ samples with the reference samples corresponding to the MAC address identified in the packet. It outputs a similarity score which highlights how similar the radio characteristic of the sender is to the sender of the reference packet.

In the setup, we have a radio head comprising a USRP and a mini-PC. The radio head transmits and receives IQ samples using a USRP B210 software-defined radio (SDR). It acts as a forwarder of the IQ samples to and from the RAN microservice in the mMTC service deployed at the edge. We use NodePort in Kubernetes to expose the mMTC service to external network. In the mini-PC, we implemented a preamble detection function in the receive chain to reduce the amount of data sent to the RAN microservice. The radio head is not managed by the BASS as this will be placed in the industry premises and not at the edge. It is connected to the mMTC service through a Gigabit Ethernet link or a 5G link using a 5G CPE connected to the 5G system used in the trial.

Figure 5-33 shows the actual trial deployment in 5TONIC. Seven IoT nodes are deployed at three locations where each of them are powered by a USB hub. The radio head is deployed on one table, where the connectivity to the edge data centre is provided via a RJ45 socket on the wall or the 5G-CPE on the same table.

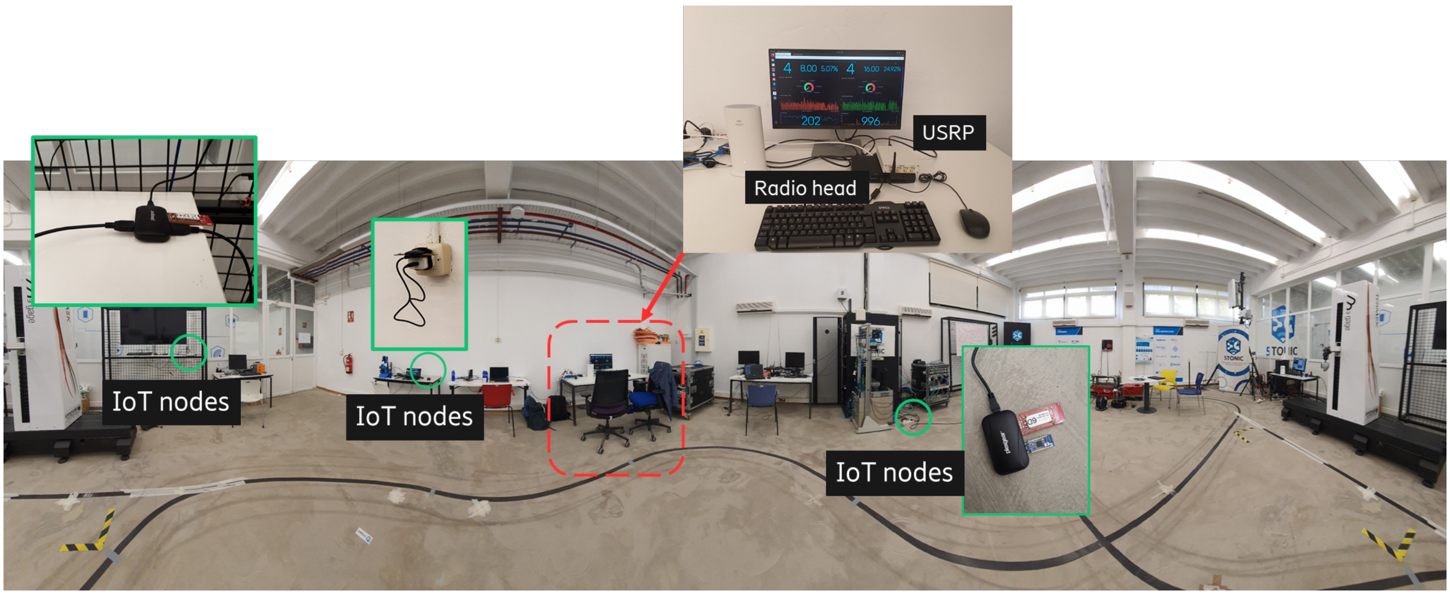
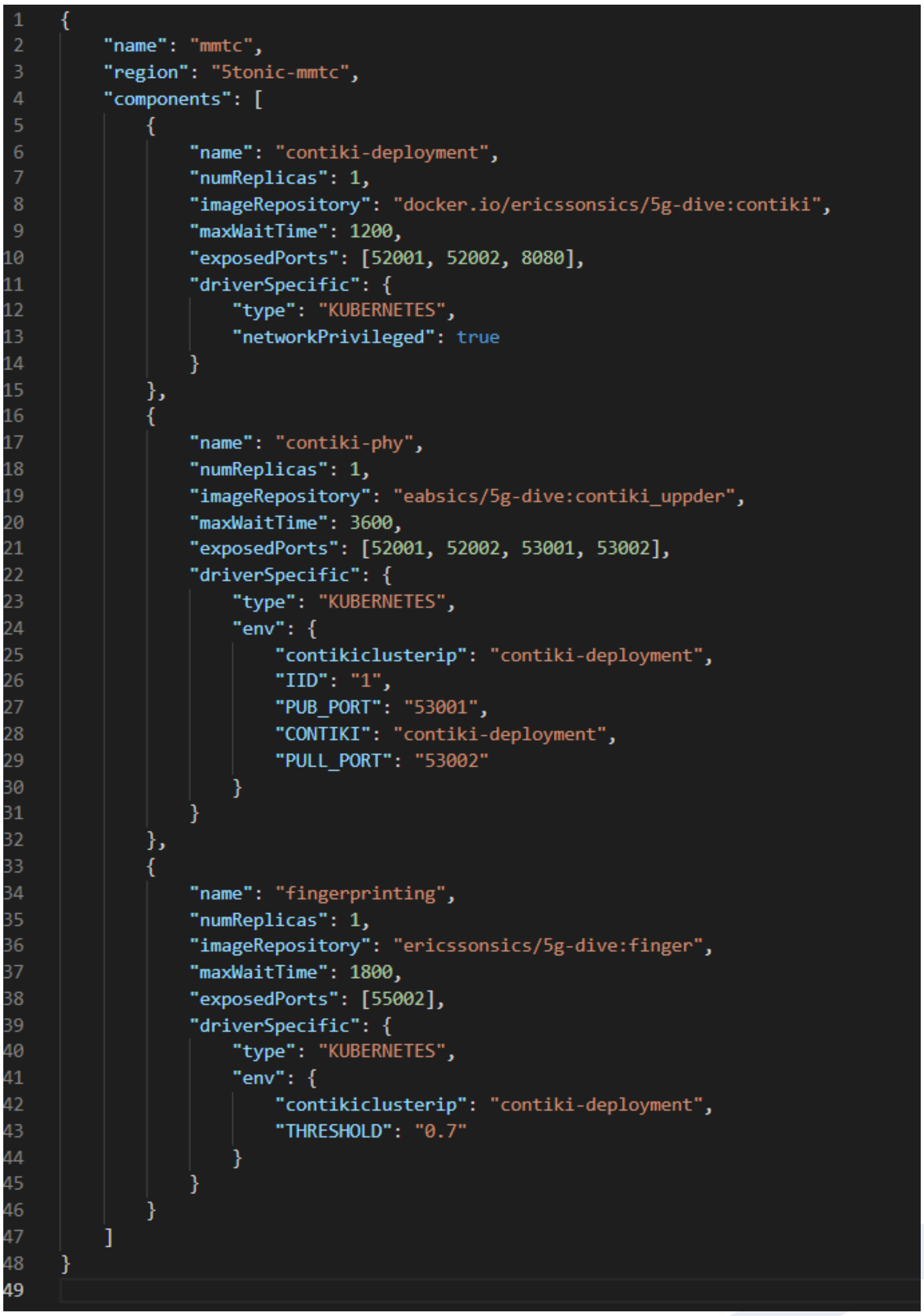


FIGURE 5-33 mMTC trial setup deployment in 5TONIC

5.3.1. DEEP Integration Validation

In the trial, the BASS is fully integrated with the mMTC service. In this section, we describe details of the integration with the DEEP platform, showing that DEEP works properly with the mMTC service. In (D2.3: Final Specification of 5G-DIVE Innovations, 2021) we presented the Kubernetes deployment scripts and explained the mapping of mMTC use case to the DEEP platform, focusing on BASS service instantiation and IESS automation.

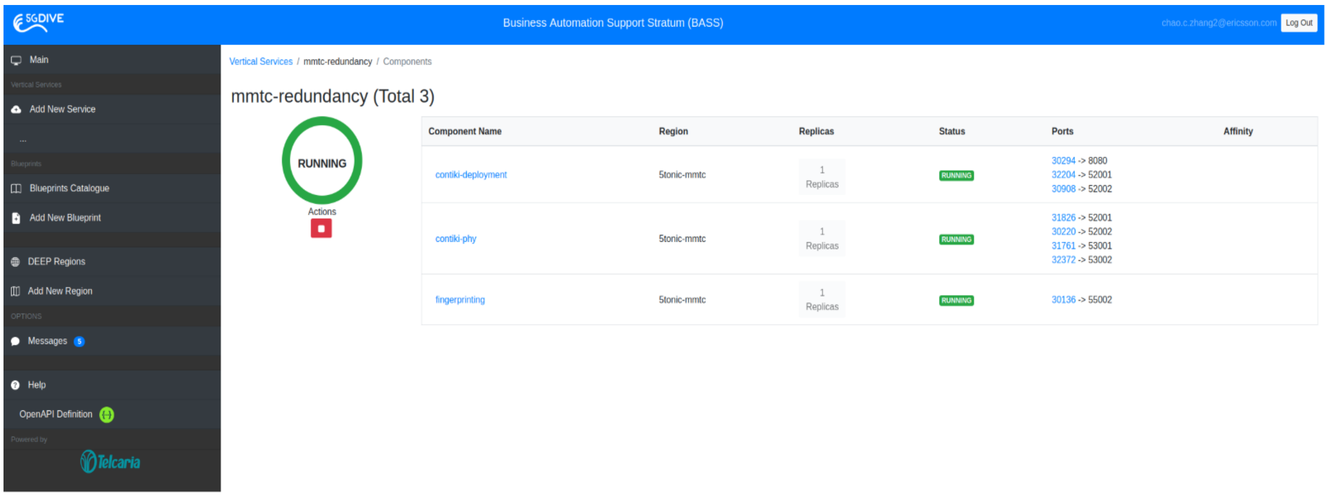
Figure 5-34 illustrates the BASS VSD file for the mMTC use case. We deploy three microservices in the cluster and expose them with an external IP address. The BASS descriptor is expressed in JSON format, compared with Kubernetes deployment YAML file. In (D2.3: Final Specification of 5G-DIVE Innovations, 2021), we discussed that for each microservice deployed in the vertical service, we need to create the corresponding Deployment to provide declarative updates for Pods and ReplicaSets configurations. Besides, we also need to create the Service which defines a policy to access the Pods inside the cluster. In Section 3.1.2, the LOC for Kubernetes deployment files and BASS VSD are compared to show the low complexity as well as easy management with BASS deployment.



```
{
    "name": "mmtc",
    "region": "5tonic-mmtc",
    "components": [
        {
            "name": "contiki-deployment",
            "numReplicas": 1,
            "imageRepository": "docker.io/ericssonsics/5g-dive:contiki",
            "maxWaitTime": 1200,
            "exposedPorts": [52001, 52002, 8080],
            "driverSpecific": {
                "type": "KUBERNETES",
                "networkPrivileged": true
            }
        },
        {
            "name": "contiki-phy",
            "numReplicas": 1,
            "imageRepository": "eabsics/5g-dive:contiki_uppder",
            "maxWaitTime": 3600,
            "exposedPorts": [52001, 52002, 53001, 53002],
            "driverSpecific": {
                "type": "KUBERNETES",
                "env": {
                    "contikiclusterip": "contiki-deployment",
                    "IID": "1",
                    "PUB_PORT": "53001",
                    "CONTIKI": "contiki-deployment",
                    "PULL_PORT": "53002"
                }
            }
        },
        {
            "name": "fingerprinting",
            "numReplicas": 1,
            "imageRepository": "ericssonsics/5g-dive:finger",
            "maxWaitTime": 1800,
            "exposedPorts": [55002],
            "driverSpecific": {
                "type": "KUBERNETES",
                "env": {
                    "contikiclusterip": "contiki-deployment",
                    "THRESHOLD": "0.7"
                }
            }
        }
    ]
}
```

FIGURE 5-34 BASS VSD for the mMTC use case

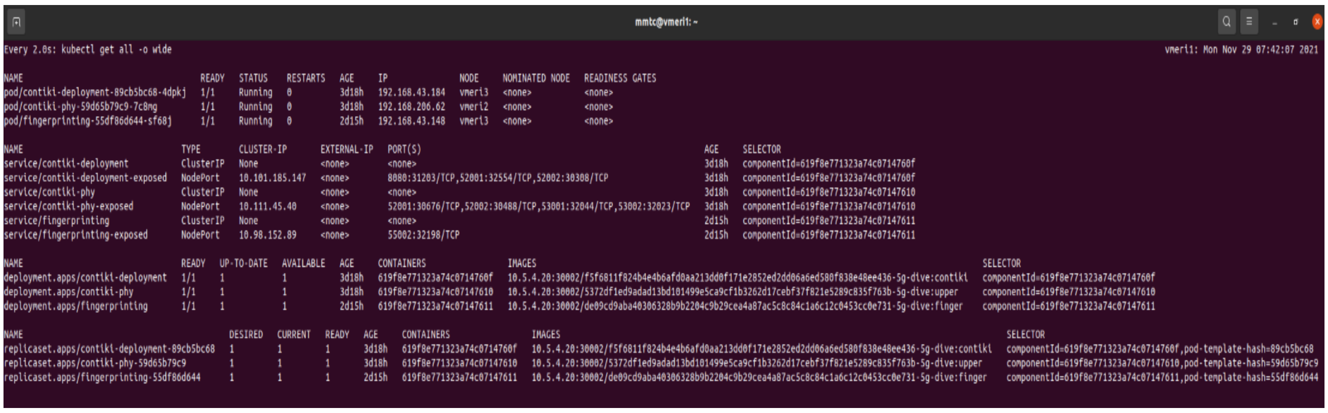
After the VSD file is created and deployed by the BASS, we observe a service list from the web GUI with details on components for the mMTC use case, as shown in Figure 5-35. The web GUI offers easy life-cycle management of the vertical service, i.e., it allows the vertical service operator to start and stop the service in a declarative way. Each time a vertical service is deployed, the BASS will generate a new namespace in the Kubernetes cluster and the components from the vertical service will be deployed therein. Figure 5-36 shows the screenshot of the microservices deployed in the Kubernetes cluster using BASS.



mmtc-redundancy (Total 3)

Component Name	Region	Replicas	Status	Ports	Affinity
contiki-deployment	5tonic-mmtc	1 Replicas	RUNNING	30294 -> 8080 32204 -> 52001 30908 -> 52002	
contiki-phy	5tonic-mmtc	1 Replicas	RUNNING	31826 -> 52001 30220 -> 52002 31761 -> 53001 32372 -> 53002	
fingerprinting	5tonic-mmtc	1 Replicas	RUNNING	30136 -> 55002	

FIGURE 5-35 BASS web view of mMTC use case with details on its components



```
Every 2.0s: kubectl get all -o wide                                                                          vmeri1: Mon Nov 29 07:42:07 2021

NAME                                  READY   STATUS    RESTARTS   AGE     IP               NODE     NOMINATED NODE   READINESS GATES
pod/contiki-deployment-89cb5bc68-4dpkj   1/1     Running   0          3d18h   192.168.43.184   vmeri3   <none>           <none>
pod/contiki-phy-59d65b79c9-7c8mg         1/1     Running   0          3d18h   192.168.206.62   vmeri2   <none>           <none>
pod/fingerprinting-55df86d644-sf68j      1/1     Running   0          2d15h   192.168.43.148   vmeri3   <none>           <none>

NAME                                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                                           AGE     SELECTOR
service/contiki-deployment           ClusterIP   None             <none>        <none>                                                            3d18h   componentId=619f8e771323a74c0714760f
service/contiki-deployment-exposed   NodePort    10.101.185.147   <none>        8080:31203/TCP,52001:32554/TCP,52002:30308/TCP                    3d18h   componentId=619f8e771323a74c0714760f
service/contiki-phy                  ClusterIP   None             <none>        <none>                                                            3d18h   componentId=619f8e771323a74c07147610
service/contiki-phy-exposed          NodePort    10.111.45.40     <none>        52001:30676/TCP,52002:30488/TCP,53001:32044/TCP,53002:32023/TCP   3d18h   componentId=619f8e771323a74c07147610
service/fingerprinting               ClusterIP   None             <none>        <none>                                                            2d15h   componentId=619f8e771323a74c07147611
service/fingerprinting-exposed       NodePort    10.98.152.89     <none>        55002:32198/TCP                                                   2d15h   componentId=619f8e771323a74c07147611

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS                 IMAGES                                                                                        SELECTOR
deployment.apps/contiki-deployment   1/1     1            1           3d18h   619f8e771323a74c0714760f   10.5.4.20:30002/f5f6811f824b4e4b6afd0aa213dd0f171e2852ed2dd06a6ed580f838e48ee436-5g-dive:contiki   componentId=619f8e771323a74c0714760f
deployment.apps/contiki-phy          1/1     1            1           3d18h   619f8e771323a74c07147610   10.5.4.20:30002/5372df1ed9adad13bd101499e5ca9cf1b3262d17cebf37f821e5289c835f763b-5g-dive:upper     componentId=619f8e771323a74c07147610
deployment.apps/fingerprinting       1/1     1            1           2d15h   619f8e771323a74c07147611   10.5.4.20:30002/de09cd9aba40306328b9b2204c9b29cea4a87ac5c8c84c1a6c12c0453cc0e731-5g-dive:finger    componentId=619f8e771323a74c07147611

NAME                                           DESIRED   CURRENT   READY   AGE     CONTAINERS                 IMAGES                                                                                        SELECTOR
replicaset.apps/contiki-deployment-89cb5bc68   1         1         1       3d18h   619f8e771323a74c0714760f   10.5.4.20:30002/f5f6811f824b4e4b6afd0aa213dd0f171e2852ed2dd06a6ed580f838e48ee436-5g-dive:contiki   componentId=619f8e771323a74c0714760f,pod-template-hash=89cb5bc68
replicaset.apps/contiki-phy-59d65b79c9         1         1         1       3d18h   619f8e771323a74c07147610   10.5.4.20:30002/5372df1ed9adad13bd101499e5ca9cf1b3262d17cebf37f821e5289c835f763b-5g-dive:upper     componentId=619f8e771323a74c07147610,pod-template-hash=59d65b79c9
replicaset.apps/fingerprinting-55df86d644      1         1         1       2d15h   619f8e771323a74c07147611   10.5.4.20:30002/de09cd9aba40306328b9b2204c9b29cea4a87ac5c8c84c1a6c12c0453cc0e731-5g-dive:finger    componentId=619f8e771323a74c07147611,pod-template-hash=55df86d644
```

FIGURE 5-36 mMTC Kubernetes cluster deployed using BASS

To visualize the system metrics of the mMTC use case, we then integrate the use case with BASS active monitoring feature. As explained in Section 3.1.3, the BASS automatically supports the monitoring of basic metrics, e.g., CPU usage, memory usage, etc. for each component deployed using a Kubernetes cluster. A Dashboard, as shown in Figure 5-37, has been established to monitor system metrics. It is worth mentioning that for other customized metrics such as latency and data loss rate, we use log data to save the metrics for post-processing.

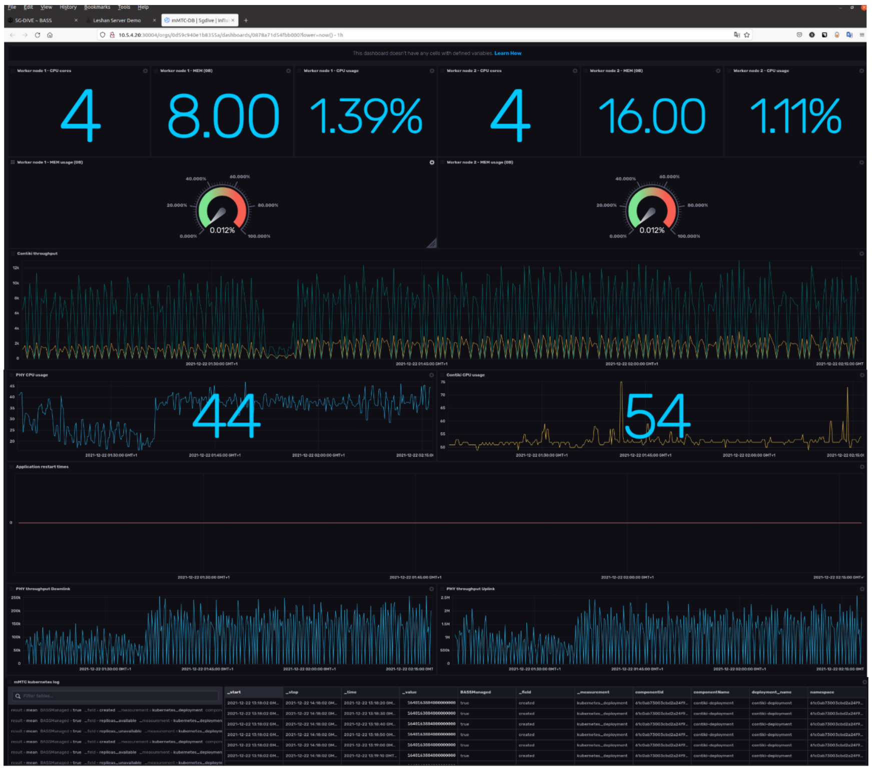


FIGURE 5-37 Influxdb monitoring dashboard for mMTC use case

5.3.2. Experimental Results

Three types of Experiments, i.e. Experiments A, B and C, have been performed in the trial. Experiment A is to measure application layer performance. Experiment B is to measure RF fingerprinting performance. Experiment C is to measure the performance of orchestration and automation features. Each type of experiment comprises several measurements to cover different aspects. For example, Experiment A has 4 measurements labeled as Experiment A.1, A.2, A.3 and A.4.

5.3.2.1. Experiment A: Application Layer Performance

In the experiments of Experiment A, we evaluate the performance of our mMTC service with an application that periodically collects temperature measurements from the IoT nodes. The implementation of experiments is shown in Figure 5-38. We developed a control and measurement module which acts like an application. It sends HTTP GET requests to the Leshan server to collect temperature data from all the connected IoT nodes. The Leshan server translates the HTTP GET request to a CoAP (Constrained Application Protocol) GET request. We modify the Leshan server to measure the round-trip time for data retrieval.

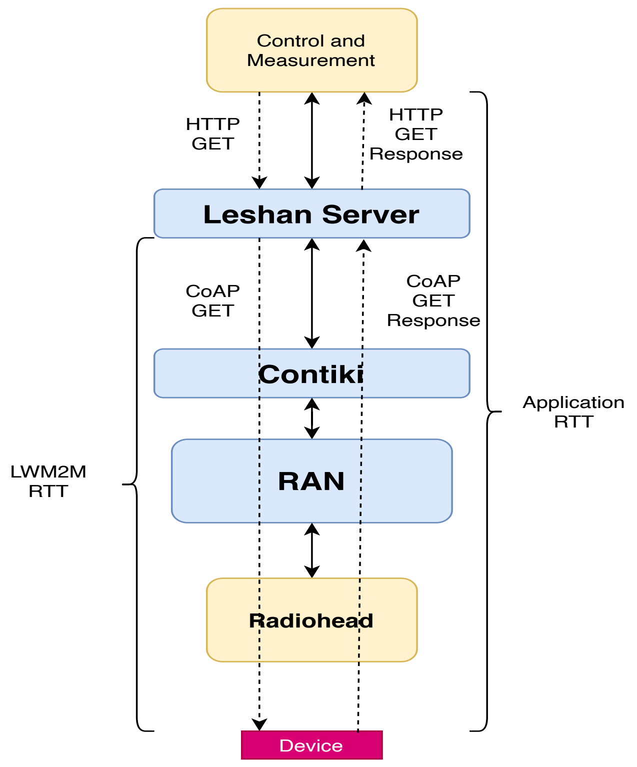


FIGURE 5-38 Implementation of Experimental Setup for experiment A

5.3.2.1.1. Methodology

The following describes the basic steps of performing the experiment.

1. Instantiate the mMTC service using the BASS of DEEP
2. Power up the IoT nodes which have been programed with a LWM2M client.
3. Wait for the clients to connect to the gateway.
4. Using the control and measurement module:
    a. Set a collection frequency for all nodes (reading of all nodes, scheduled sequentially or asynchronously by Leshan server)
    b. Request the IoT nodes for temperature sensor data (logged by the control and measurement module). Measure application data integrity or losses. Data losses signify the time-out of the HTTP response.
5. Measure the round-trip time for the response (logged by the control and measurement module together with sensor data simultaneously with step 4).
    a. LWM2M RTT: Measures the round-trip time (RTT) for each LWM2M request. It is measured by the Leshan server.
    b. Application RTT: Measures the RTT for each application (HTTP) request. The Leshan server uses synchronous requests and sends the subsequent request on successful reception of the current request by the target IoT node. This results in a queuing delay of a request at the Leshan server. So, the Application RTT includes LWM2M RTT, queuing delay at the Leshan server, and the HTTP request to CoAP request translation time.

5.3.2.1.2. Experiment A.1: Asynchronous Message Collection

IoT nodes with a specified message frequency can wake up and send the data asynchronously at random intervals. To model this messaging pattern, we set the control module to enable asynchronous polling of the nodes. The control module instantiates threads for polling each connected IoT node. Each thread sleep for a random delay between 0 and 20 seconds before sending the HTTP request for data collection from its designated client. The collection frequency is set to 1 data reading per minute for each node. We collected the data for 10 days.

5.3.2.1.2.1. Results

Figure 5-39 shows the CDF of the measured round-trip times (RTT) from all the clients. We receive a response to over 90% of requests within 400 ms. The application RTT closely follows the LWM2M RTT with a slight delay incurred by HTTP to CoAP translation and queuing delay. The DLR (data loss rate) for application data retrieval is 1.039% for close to 86000 application requests. Our application does not have a retransmission mechanism and relies on link-layer retransmissions. Due to the asynchronous nature of the responses from the IoT nodes for message requests with random delay initialization, over-the-air collisions are probable. This leads to link failure and failure of the application data retrieval. Some performance degradation is also due to the IoT node behavior. The 5 newly bought IoT nodes didn't perform as well as the 2 nodes we have used for a long time. This phenomenon is also shown in Figure 5-40, which will be discussed there. Nevertheless, we can further improve the reliability of our application by using application-layer retransmissions. For example, by one retransmission on the application layer by request again the data if no response received on the first request, the DLR can be reduced to less than 0.011%. Further reduction can be done with more retransmissions on the application layer

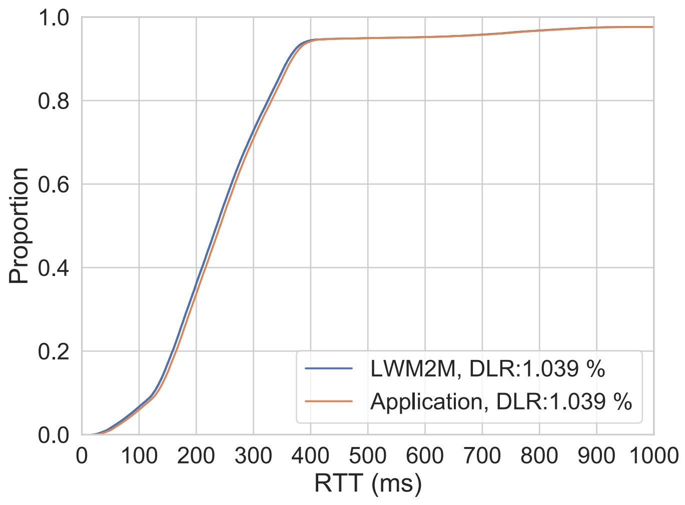


FIGURE 5-39 CDF of RTT for asynchronous requests

Figure 5-40 shows the application performance per node. We use boxplots to show the distribution of the LWM2M RTT for each node. The nodes have similar RTT distribution with a median of approximately 250 ms. However, the DLR for each node varies considerably. The nodes with the nRF52840 platform (A527, B60E, F16B, 5105, EDA1 in the figure) which were newly bought show higher

DLR than nodes with the Zolertia Firefly platform (B16D, B4D4 in the figures) which we have used for a long time. The nodes with the nRF52840 platform also have lower LWM2M RTT than the nodes with the Zolertia Firefly platform. It points to a failure of our MAC retransmissions on the nodes to send the requested data. Basically, B16D and B4D4 have been more robust than the other 5 nodes, which represents more the normal performance expected. It can be due to RF compatibility issues or an issue regarding configuration optimization between our SDR receiver and the nRF52840 platform.

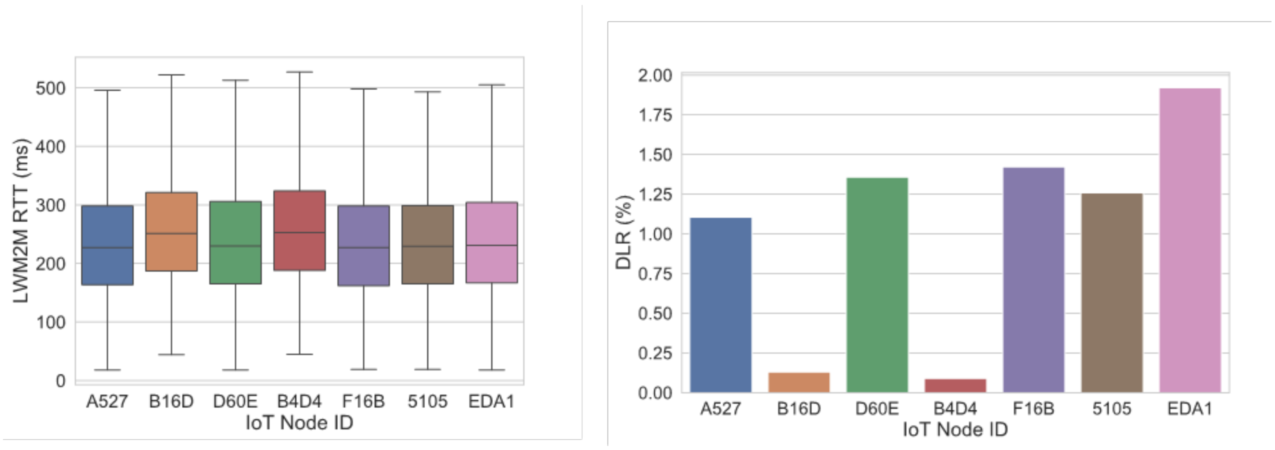


FIGURE 5-40: Result of application performance per node for asynchronous requests

5.3.2.1.3. Experiment A.2: Scheduled Message Collection

In an application-driven architecture, the application schedules the data retrieval from the IoT nodes. We model this messaging pattern by using the control module to schedule the requests to the IoT nodes. The collection frequency is set to 1 data reading per minute per node. The control module instantiates threads for data retrieval from each connected IoT node. The control module instructs the threads to sleep for a particular delay. On waking up from sleep, each thread sends an HTTP request for data collection from its designated IoT node. For this measurement, the control module schedules data collection every 5 seconds. The data are collected for 7 days.

5.3.2.1.3.1. Results

Figure 5-41 shows the CDF of the measured RTT from all the clients. Similar to the asynchronous polling case, we receive a response to over 90% requests before 400 ms with a slight difference between the application RTT and the LWM2M RTT. We achieve a lower DLR of 0.639% with scheduled requests than 1.039% with asynchronous requests. The scheduling of data retrieval by the control module reduces the chances of over-the-air collision between packets from different nodes. This results in better reliability, as demonstrated by the lower DLR.

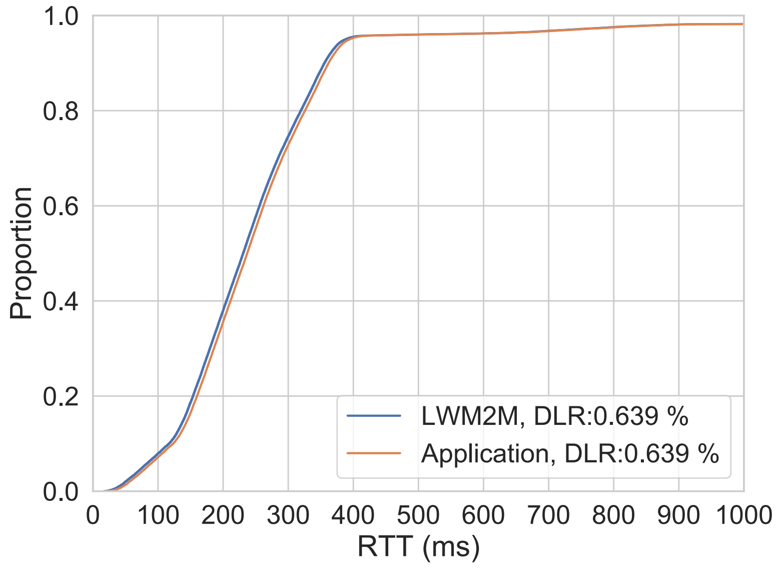


FIGURE 5-41 CDF of RTT for scheduled requests

We show the LWM2M RTT and DLR for each node in Figure 5-42. For LWM2M RTT distribution, we observe similar results to the asynchronous case. The DLR across different IoT nodes once again highlights the discrepancy between nodes with different platforms. The Zolertia Firefly nodes have only one data loss event for the duration of this experiment. However, scheduling requests to the nodes reduces the DLR from each node. It means the actual performance in practice with good IoT nodes should perform as well as the Zolertia Firefly nodes.

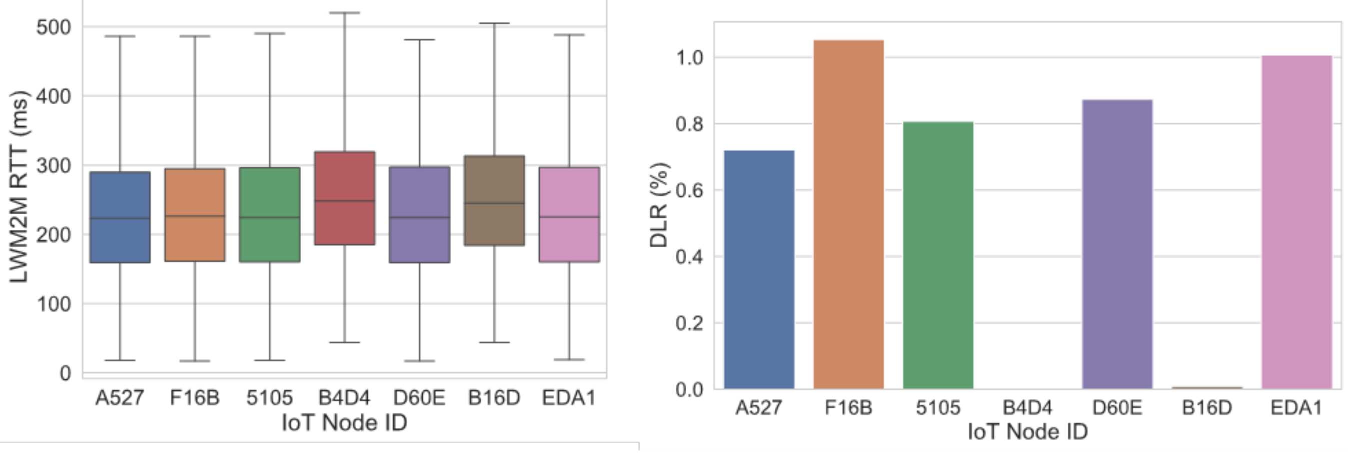


FIGURE 5-42 Application performance of different nodes with scheduled requests

5.3.2.1.4. Experiment A.3: Performance over 5G

In this experiment, we connect the radio-head to our mMTC service using 5G instead of the 1Gbps ethernet link. We use asynchronous polling with a collection frequency of 1 message per minute for each node. The data is collected for 1 hour.

5.3.2.1.4.1. Results

Figure 5-43 shows the CDF of the RTT with a 5G connectivity. We observe a higher RTT (about 100 ms in average) as compared to the results with the Ethernet link. This is mainly because of a higher 5G latency than the Ethernet link. However, we observe a DLR of 0.621%, similar to the scheduled message collection results. Hence, the higher latency of the 5G link does not impact the reliability of our data collection application.

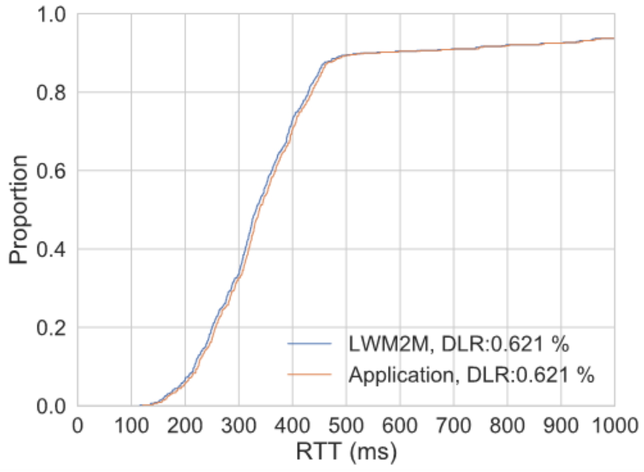


FIGURE 5-43 CDF of RTT with 5G connectivity

Figure 5-44 shows the breakdown of the LWM2M RTT and DLR per node. We observe a median LWM2M RTT of slightly over 300ms. Like our previous experiments, the nodes with the nRF52840 platform have higher DLR as compared to the nodes with Zolertia firefly platform. Overall, the results show that we can use a 5G connectivity to support our mMTC service. This is useful in cases where the radio-heads do not have access to a fixed network in an environment that requires high security or is in a remote area too costly to access a fixed network.

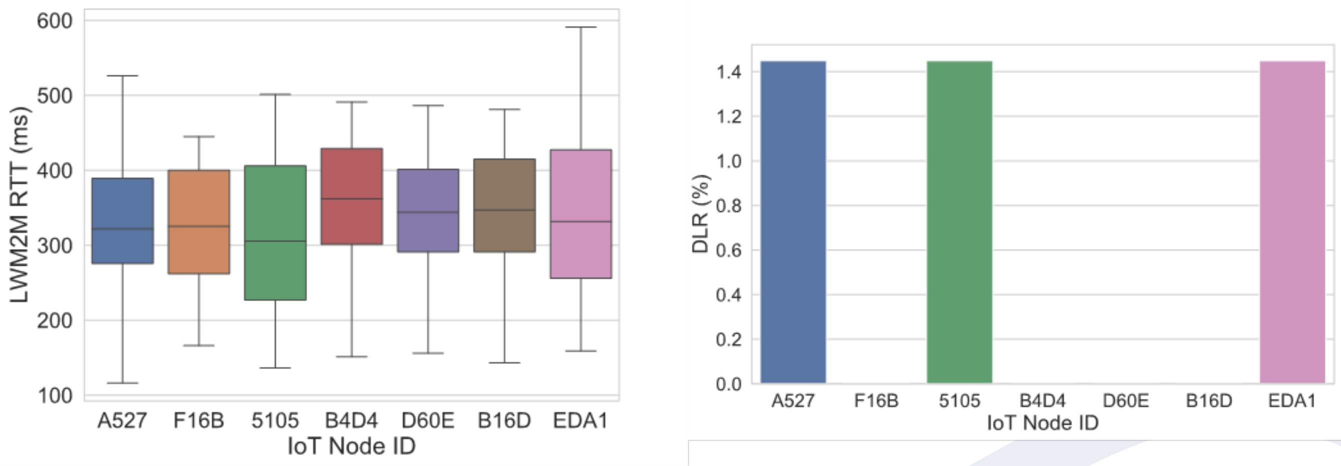


FIGURE 5-44 Application performance per node with 5G connectivity

5.3.2.1.5. Experiment A.4: Impact of Collection Frequency

In this experiment, we measure the impact of collection frequency on performance. We select the collection frequency of 5 and 10 messages per minute for each node. The data is collected for 30 minutes. The parameters used in the control module for the asynchronous message pattern and the scheduled message pattern are shown in Table 5-7.

TABLE 5-7 Parameters for different message patterns

Message Pattern	Packets per minute per node	Random Delay (async)/ Interval between requests (scheduled)
Async	5	0-5 seconds
	10	0-2.5 seconds
Scheduled	5	1.5 seconds
	10	700 milliseconds

5.3.2.1.5.1. Results

Table 5-8 shows the LWM2M RTT and DLR results for the two tested message frequencies for the asynchronous and the scheduled message patterns. We observe an increase in the DLR with increased message frequency for both the message patterns. However, due to higher chances of collision in the asynchronous message pattern, we observe significant degradation in the DLR with a message frequency of 10 packets per minute to each node. We also notice the impact of the over-the-air collisions in the 95th percentile value for the LWM2M RTT. The high RTT for this case, which indicates the application needed multiple MAC layer retransmissions for successful data retrieval. The scheduled message pattern achieves a 95th percentile value below 1000 ms with a PER of 1.5%, which is considerably better than the asynchronous message pattern.

We also evaluate the impact of collection frequency with a 5G connectivity. In this case, we use the scheduled message pattern. For both the collection frequencies, we observe DLRs comparable to the scheduled case. The higher 5G-link latency is highlighted in the mean, 5th percentile and 95th percentile values of the LWM2M RTT. However, given the low DLR for high collection frequency of 10 messages per node, we can argue that with application layer reliability we can also use 5G connectivity for high frequency message collection with our mMTC service.

TABLE 5-8 Impact of collection frequency on application performance

Message Pattern	Packets per minute per node	LWM2M RTT (50th percentile) in ms	LWM2M RTT (5th percentile) in ms	LWM2M RTT (95th percentile) in ms	DLR (%)
Async with Ethernet	5	233.00	81.00	868.75	1.06
	10	235.00	84.00	2419.50	3.40
Scheduled with Ethernet	5	241.50	85.35	398.55	0.70
	10	238.00	84.80	826.80	1.50
Scheduled with 5G	5	388.03	181.67	709.55	0.38
	10	446.61	158.50	1225.00	1.37

5.3.2.2. Experiment B: RF Fingerprinting Performance

In this experiment, we evaluate the functionality of our RF Fingerprinting module. IoT networks can be attacked by intruder nodes. The intruder nodes disguise their identify and communicate with the gateway. Our RF Fingerprinting module identifies nodes based on their signal characteristics and hence can be an effective solution to intrusion attacks at the gateway. We evaluate its performance in a longer running experiment for 18 days.

5.3.2.2.1. Methodology:

In our RF Fingerprinting module, we compare an incoming message from the node with MAC address "ID" to a reference signal from that node. The steps are highlighted below.

Identifying nodes based on their radio characteristics:

1. For every incoming packet, we collect the MAC address and verify the packet is correctly decodable (eliminates corrupted packets)
2. We randomly sample radio chunks of window size of 128 I/Q samples from the reference signals for that node.
3. We compare the incoming packet with the radio chunks and get a similarity score.
4. We log the similarity score for each packet along with the node details for post-processing.

5.3.2.2.2. Results:

We have 7 IoT nodes, comprising of 6 original nodes and 1 intruder node. We program the intruder node with the same MAC ID as one of our trained nodes (not used in the experiments). We log the similarity scores per-packet identification and then post-process the identification results. The data is collected for 18 days. In total, we collected 528928 packets, out of which 89058 packets are from the intrusion node.

Figure 5-45 shows the Receiver Operating Characteristics (ROC) curve for the duration of the experiment. The X-axis shows the false positive rate i.e., the fraction of the intrusion messages that get misidentified as messages from the correct devices. The Y-axis shows the true positive rate i.e., the fraction of messages from the correct devices getting identified as correct messages. Both the X-axis and Y-axis are normalized to the total number of messages from the intrusion node and the correct nodes.

The ROC curve shows the true positive rate and false positive rate that can be achieved by tuning the threshold on the similarity score. Ideally, we would like to find a threshold that enables the RF Fingerprinting module to correctly identify all messages from the correct nodes accurately, while flagging all messages from the intruder node as an intrusion. This scenario would be depicted by a straight line that has a Y value of 1 for an X value of 0. With our RF-Fingerprinting module, we can achieve a Y value of over 0.85 for an X value of 0. This means we can accurately identify 85% of messages from the correct devices when it flags all intrusion messages as an intrusion. Moving along the X-axis signifies a lowering of the threshold on the similarity score. A lower threshold would allow messages that have a lower similarity score to be identified as a correct message. This leads to an increase in the false-positive rate i.e. the number of intrusion messages identified as correct. We achieve a true positive rate of 1 i.e., all messages from correct devices get correctly classified at $X < 0.1$. So, our fingerprinting module can correctly identify all correct messages when we allow misidentification of around 10% of intruder messages.

For reference, we show the ROC curve for a random classifier as a dotted blue line. We observe that we can achieve a high true-positive rate for even a low false-positive rate signifying our model manages to differentiate the correct messages from the intrusion messages. The ROC Area under the Curve (ROC-AUC) measures the area under the ROC curve for our classifier. Our model achieves a high ROC-AUC of 0.9966 (ROC-AUC for the perfect classifier is 1) and signifies that we can find a threshold with our model that differentiates between correct and intruder nodes significantly well.

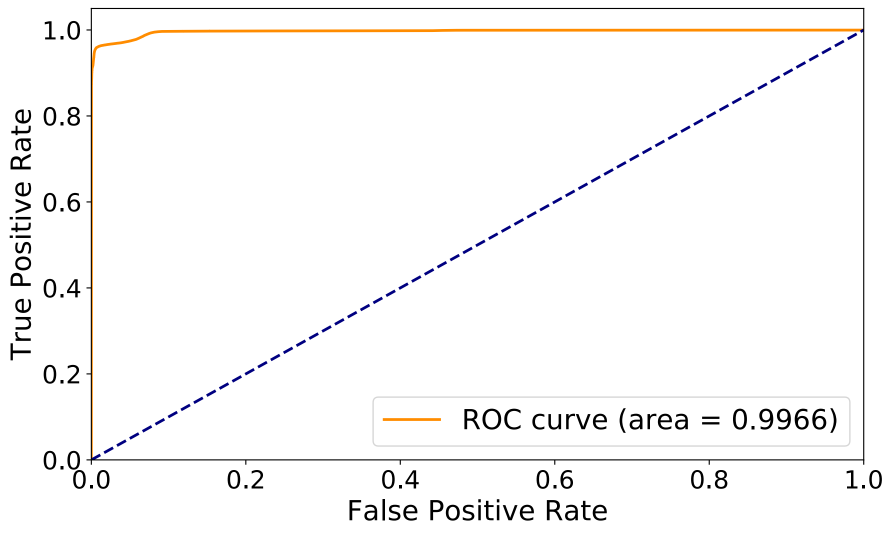


FIGURE 5-45 ROC curve for RF Fingerprinting

5.3.2.3. Experiment C: Orchestration and Automation Performance

In previous sections, we verify the functional features of the BASS and test the performance of our mMTC system. However, due to the heterogeneity of edge sites, e.g., using different types of CPUs, different CPU clock frequency, and memory footprint, varying background load, network interrupts, etc., the services deployed may experience unexpected slow execution, even crash in runtime (H. Zhao, 2020). The hardware nodes used in an Edge data centre, e.g. servers, are usually less robust than telecom-grade equipment, which can experience HW issues more often. Thus, it is important to understand the performance impact of such failures that may happen to the system and how these failures can be mitigated in mMTC use case.

In this section, we discuss the self/healing capability of Kubernetes and the life cycle management of the BASS. More specifically, we measured the performance of the mMTC service when pod/node failure happens. We have four test scenarios, i.e., no failure scenario (baseline scenario), pod failure scenario, pod failure with redundancy scenario, and node failure with redundancy scenario. For each measurement, we run the test for 10 minutes with the node traffic of 10 requests /min/IoT node.

5.3.2.3.1. Experiment C.1: Pod failure without redundancy

In this part, four experiments are conducted to analyse the performance impact of the pod failure. In all four experiments, we have the 'replicas' value set to 1 for 'Contiki-PHY' microservice (corresponding to the RAN microservice) in our VSD file, i.e., we have one RAN instance running in the edge. For the baseline case, there is no human intervention and for the failure test, we manually delete the pod which is running 'Contiki-PHY' microservice. It is worth mentioning that Kubernetes implements graceful termination by applying a default grace period of 30 seconds. In our experiments, we set the grace period value to 0 in order to achieve immediate pod termination to emulate the pod failure event.

Table 5-9 reports the measurement results for the pod failure without redundancy scenario. We observe that we receive 695 and 697 response packets for the two measurements, respectively, when no pod restart happens for the baseline case. The data loss rates for the two measurements are 1.44% and 1.58% respectively. For the pod failure case, we receive 596 and 589 response packets for the two measurements, respectively. This is due to that two nodes were disconnected from the Leshan LWM2M server when pod failure happens. We can also observe that the pod restart time is around 3.5 seconds which means that the service outage lasts for at least 3.5s and response data from the IoT devices during that period will be lost. For the two measurements, we got 28 and 29 data losses due to the pod failure.

TABLE 5-9 Comparison results for the Pod failure without redundancy scenario

Measurement #	Data received	Pod restart times	Pod restart time [s]	Data loss rate	Data loss due to pod failure	RTT Leshan Server (mean) [ms]	RTT application (mean) [ms]
1	695	0	-	1.44%	-	362.20	368.16
2	697	0	-	1.58%	-	351.58	357.61
3	596	1	3.4	5.87%	28	334.81	341.06
4	589	1	3.6	6.28%	29	412.13	418.46

Figure 5-46 shows the application performance per node for the pod failure test. We use the results of measurement 3 to show the average server RTT as well as application RTT in the left figure. There are four nodes with average RTT values below 300ms. The overall RTT values vary from 268ms to 391ms. However, in the right figure which shows the data loss rate for each node, we observe the great disparity between the nodes. The two nodes (node 'B16D' and 'B4D4') with the Zolertia Firefly platform only receive around 60 data packets as both nodes disconnect from the server when pod failure happens. The nodes located farthest from the receiver (node 'D60E' and 'F16B') introduce more data losses compared with the rest nodes. It is also worth mentioning that these two nodes have higher data loss in the baseline case also, as seen in Experiment A.1 and A.2.

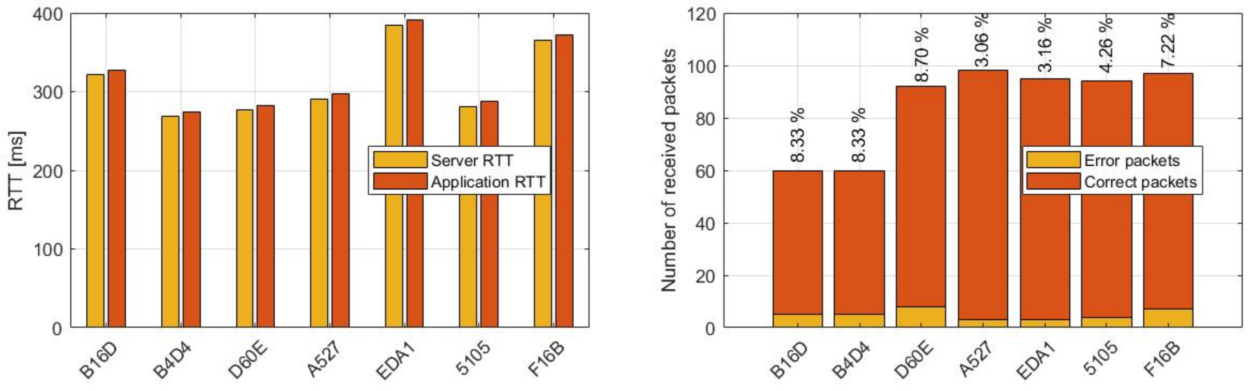


FIGURE 5-46 Results of application performance per node for pod failure scenario

5.3.2.3.2. Experiment C.2: Pod-level redundancy

In this part, we have 2 more measurements done with pod failure. However, we set the 'replicas' value to 2 instead of 1 to have redundancy in the system. By doing this, we have two 'Contiki-PHY' pods in our cluster and Kubernetes will route the traffic to either pod. During the measurement, we still manually delete one pod from the cluster, and Kubernetes will automatically start a new pod to meet the 'replicas' value set for the 'Contiki-PHY' deployment.

Table 5-10 reports the measurement results for the pod redundancy scenario. We observe from the table that the Leshan LWM2M server received nearly the same amount of response data in all four measurements. Besides, for the pod redundancy cases, no data loss due to the pod failure is observed, leading to data loss rates of 0.43% and 1.15% respectively. However, there is no data loss observed during the pod failure. The value is still reasonable since we know that some nodes may have unstable performance regarding data loss rate. Furthermore, with a redundant pod, we also obtain similar average server/application RTT for the measurements. It shows having a replica is an effective solution to increase the availability and reliability of the mMTC service.

TABLE 5-10 Measurement results for POD redundancy scenario

Measurement #	Packet received	Pod restart times	Data loss rate	Data loss due to Pod failure	RTT Leshan Server (mean) [ms]	RTT application (mean) [ms]
1	695	0	1.44%	-	362.20	368.16
2	697	0	1.58%	-	351.58	357.61
3	696	1	0.43%	0	365.55	372.79
4	697	1	1.15%	0	323.39	329.51

Figure 5-47 illustrates the application performance of each IoT node for pod redundancy scenario. We use measurement 3 as an example to show the node behaviour when failure happens in pod redundancy scenario. It is shown in the right figure that we haven't got a service outage during the whole measurement. All nodes are connected stably and only node 'D60E' introduces some data losses. In the left figure, we observe the average RTT for all nodes varies between 263ms to 567ms, with the highest value also from the node 'D60E'.

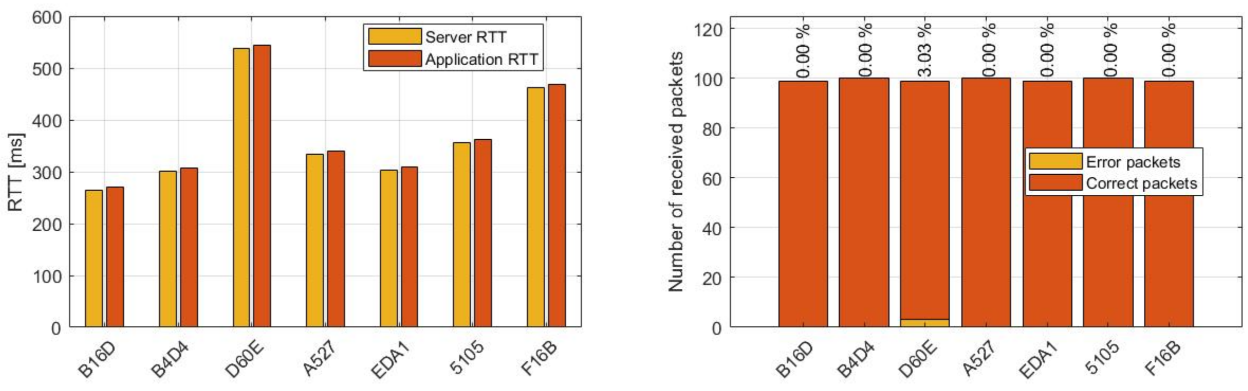


FIGURE 5-47 Results of application performance per node for POD redundancy scenario

5.3.2.3.3. Experiment C.3: Node-level redundancy

In this measurement, we have two 'Contiki-PHY' pods deployed on two worker nodes. Instead of manually deleting the pod, we reboot the worker node 1 where only 'Contiki-PHY' pod is running on. Thus, we can explore the performance impact of having node redundancy without harming other pods inside the cluster. Below we show what happens inside the cluster when a node failure happens.

- When the node fails, it becomes unreachable and the master sets the node to the 'NotReady' state.
- The master waits for 'pod-eviction-timeout' before taking any action. The default value is 5 minutes.
- After 'pod-eviction-timeout' time, the master sets the partitioned node's pod to 'Terminating' state and creates new instances of pods on different nodes.

For our case, we have two pods on two different worker nodes. After we manually reboot one node, there is only one other pod left in the cluster that takes over the traffic. The new pod will not be created until the default 'pod-eviction-timeout' time is up.

Table 5-11 reports the measurement results for the node redundancy scenario. It is worth mentioning that the measurement time for measurement 3 is 8 minutes 30 seconds, so we have fewer data received compared with the baseline case. We observe that with node redundancy, we manage to obtain a similar data loss rate compared with the baseline case. Furthermore, we observe zero data loss when the node failure happens, indicating a graceful handover of the traffic from one node to the other node. It shows that node-level redundancy is an effective way to protect the system from node failure events and therefore further enhance increase the availability and reliability of the mMTC service.

TABLE 5-11 Measurement results for node redundancy scenario

Measurement #	Packet received	Node restart times	Data loss rate	Data loss due to node failure	RTT Leshan Server (mean) [ms]	RTT application (mean) [ms]
1	695	0	1.44%	-	362.20	368.16
2	697	0	1.58%	-	351.58	357.61
3	582	1	1.20%	0	346.70	353.06

5.3.3. Scalability

In an IoT application where the amount of data can be unpredictable on occasion, it is important to have the ability to assign more resources when needed and withdraw the resources after the surge ends. One of the major capabilities of Kubernetes as an orchestrator lies in its ability to dynamically manage and respond to varying network traffic by performing effective auto scaling. In this section, we evaluate the scalability (in terms of resource utilization) of the mMTC microservices, i.e., RAN and Contiki microservices that are deployed in the edge under different traffic loads. We first present the evaluation utilizing the test node simulator (D2.3: Final Specification of 5G-DIVE Innovations, 2021) we developed in the project, with which we can emulate a large number of nodes to investigate the scalability. Then we present the results measured in the I4.0 trial in 5TONIC. This section is concluded with the discussions regarding the two service auto scaling approaches, i.e., horizontal scaling and vertical scaling that are supported by Kubernetes.

5.3.3.1. Simulation results with emulated test nodes

Figure 5-48 shows the experimental setup with emulated test nodes. The test node acts as a fully functional Contiki-based LWM2M Client. We create a Docker network and deploy our Contiki and PHY functions. We allocate one CPU core to Contiki and the two RAN functions. Next, we instantiate the test nodes and wait for the registration of these nodes. On successful registration of these nodes, we schedule ping to each of these nodes with an interval of 1 minute like our long-term measurements. The experiments were conducted on an Apple MacBook Pro equipped with an 8-core Intel i9 processor with hyper-threading which effectively gives us 16 cores. We measure the resource utilization of the Contiki and PHY instances using Docker stats every second.

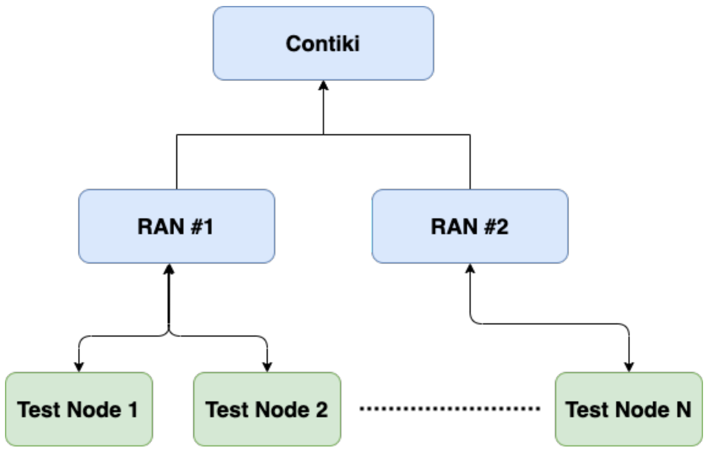


FIGURE 5-48 Experimental setup with emulated test nodes

We conduct two experiments to study the scaling behaviour of the mMTC components. As our Contiki and RAN functions are CPU-intensive processes, in these experiments we examine the CPU utilization with the varying number of test nodes.

Experiment 1: CPU Resource utilization with varying number of nodes

In this experiment, we initially instantiate 20 test nodes and measure the CPU utilization under ping load. We collect the measurements for 10 minutes. Then we instantiate 10 more test nodes and repeat the measurement process. We could emulate up to 40 nodes due to CPU resource limitation in this experiment setup, which is already much more than the number of nodes we could deploy in the trial field.

Figure 5-49 shows the variation of CPU utilization with the increase in the number of test nodes. The Y-axis shows the mean percentage of CPU utilization of one core assigned to the Contiki and RAN microservices. The error bars show the 95% confidence interval of the mean CPU utilization. For this experiment, all the test nodes are connected to 'RAN #1'. We observe that both the 'Contiki' and 'RAN #1' microservices scale with the number of the test nodes. Since the RAN must perform heavier signal processing operations, its CPU utilization increases at a higher rate compared to the Contiki. Broadcast packets are still sent over 'RAN #2', hence resulting in a small CPU utilization. From the trend, it can be estimated that one CPU core used in this experiment can support more than 100 IoT nodes under the test traffic pattern. Then it will scale with the number of CPU cores when even more IoT nodes are connected.

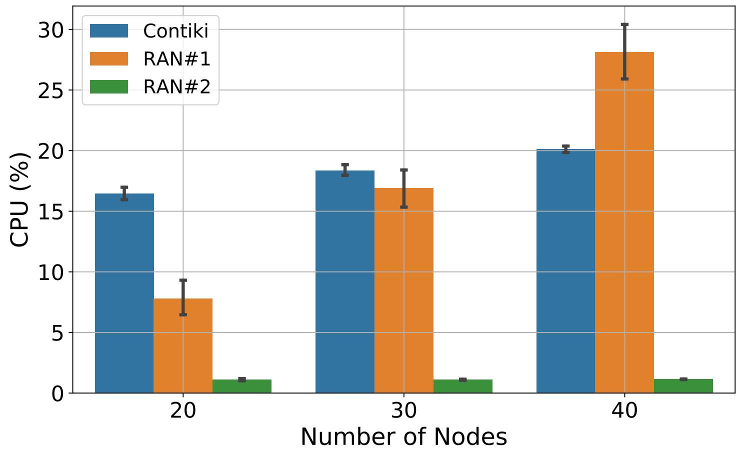


FIGURE 5-49 CPU utilization with respect to the number of test nodes

Experiment 2: Balancing the load across multiple RAN microservices

In this experiment, we examine if handling traffic from multiple RAN microservices impacts the resource utilization of the Contiki microservices. We instantiate 40 test nodes and distribute them equally among both the RAN microservices. Figure 5-50 shows the impact of balancing the nodes across the two RAN microservices. We observe that distributing the load evenly among the RAN microservices results in lower overall CPU utilization. This is because that the load balancing also makes the system more robust such that the number of retransmissions is reduced, which reduces the actual processing load. The CPU utilization of the Contiki microservice remains the same for both cases, highlighting there is no significant overhead for handling traffic from multiple RAN microservices.

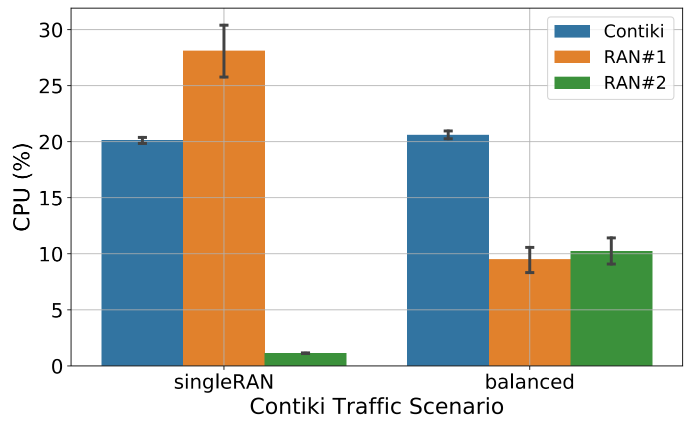


FIGURE 5-50 Impact of balancing the load across two RAN microservices

5.3.3.2. Trial results in 5TONIC

Using the BASS active monitoring, we measure the resource utilization of the mMTC microservices during the trial. Since the nodes are physically deployed at 5TONIC lab, we herein increase the traffic loads by sending more requests to the nodes. Figure 5-51 shows the CPU utilization of the RAN and Contiki microservices (or pods). The Y-axis shows the CPU utilization of one core assigned to the corresponding instance. We use boxplots to show the distribution of the CPU utilization under different traffic loads. We have tested with seven different traffic loads which vary from 1 data requested per minute per node to 25 data requested per minute per node. We observe that the average CPU utilization increases with more traffic loads for both RAN and Contiki microservices, as expected. For the RAN microservice, the median CPU utilization values of all test cases are 2.92%, 3.78%, 5.58%, 7.32%, 8.76% and 9.12%, respectively. The results show a significant scaling of CPU consumption for the RAN microservice and thus we can use it as an effective metric to scale out/in the service.

For the Contiki microservice, the median CPU utilization values for all test cases are 5.18%, 5.28%, 5.37%, 5.45%, 5.51% and 5.52%, respectively. Under the case of 25 data requested per minute, the median CPU utilization of Contiki microservice is 5.52% compared with 9.12% of RAN microservice, indicating that one Contiki instance can handle much more traffic than one RAN microservice. In addition, the slower trend of the increase in CPU utilization for Contiki further proves this. It is also worth mentioning that our use case is not memory demanding, and the average memory usage is always below 1%. Thus, we don't use memory usage as criteria for the scaling of the services.

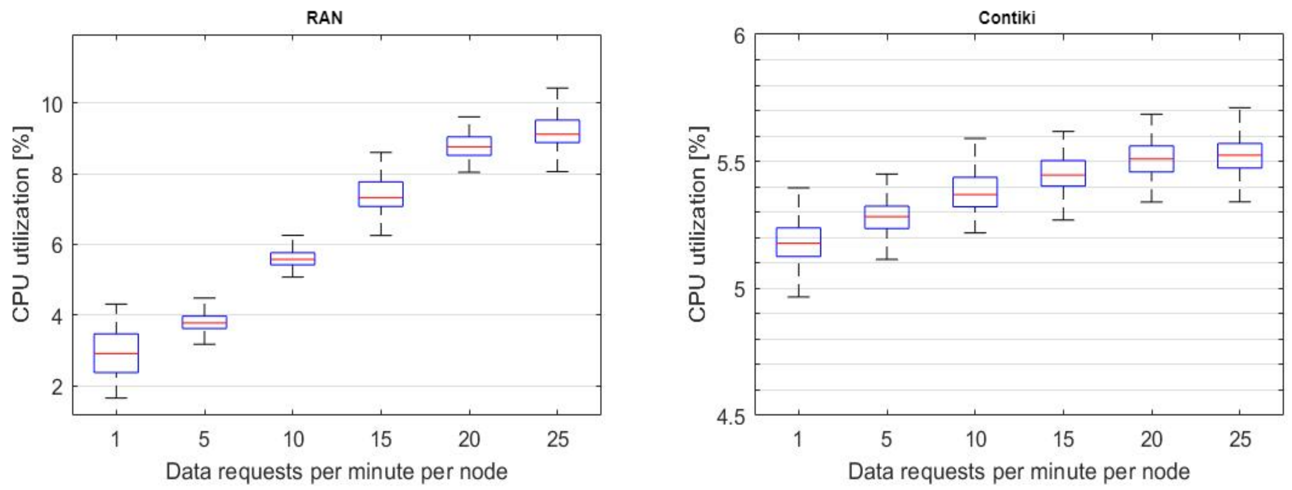


FIGURE 5-51 CPU utilization for PHY and Contiki under different traffic loads

5.3.3.3. Discussion regarding autoscaling

From the simulation results, for the RAN microservice, we get a mean CPU utilization of 28.13% for one CPU core with 40 test nodes under load. For the traffic pattern tested, considering the increasing trend in the CPU utilization with the number of test nodes, the RAN microservice will be able to support more than 100 IoT devices with one CPU core. Similarly, the Contiki microservices will be able to support more than 150 IoT devices with one CPU core. Taking this into account, we also discuss scaling strategies using horizontal and/or vertical scaling to dynamically scale the resources to the RAN and Contiki microservices according to the processing load.

Horizontal Pod Autoscaler (HPA)

Horizontal scaling allows Kubernetes administrators to automatically increase or decrease the number of running pods (or microservices) when the level of resource usage changes. The horizontal pod autoscaling controller periodically adjusts the desired scale of its target, e.g., a Deployment, to match observed metrics such as average CPU utilization, average memory utilization, etc. If the operator configures a workload to autoscale based on multiple metrics, the HPA evaluates each metric separately and uses the scaling algorithm to determine the new workload scale based on each one. The largest scale is selected for the autoscale action. The horizontal scaling doesn’t interrupt the traffic processing thanks to load balancing used which automatically routes the traffic to the available pods.

Vertical Pod Autoscaler (VPA)

Instead of changing the number of pods in the cluster, vertical scaling set limits on container resources based on the actual level of resource usage. A VPA allows the operator to scale a given service vertically within a cluster. It is more applicable for the cases where some services are not possible or not ideal to scale horizontally due to some constraint. It is also worth mentioning that VPA destroys a pod and recreates it with a new resource limit to vertically scale the resources allocated. Due to this, it is recommended to use VPA for more stable updates of pods over a longer period.

For mMTC, the RAN microservice is suitable for horizontal scaling as it is more sensible to the traffic loads. Besides, by enabling horizontal scaling for the RAN, the system will also benefit from having pod redundancy when the service scales out. Meanwhile, vertical scaling is recommended as a scaling strategy for the Contiki service as it is less sensitive to the traffic loads. Use of a single Contiki microservice to manage the overall IoT network leads to more optimized networking decision making, e.g which radio-head is better suited for a particular IoT node. Furthermore, it makes it enables the application developers to have a unified access and control interface for the entire network. In our current implementation, the MAC layer is located in the Contiki microservice. But, from the autoscaling perspective, it may be more suitable to have the MAC layer in the RAN microservice as this layer is more suitable for horizontal scaling.

6. Final validation results of ADS use cases

In this section, we presented the final validation results for ADS use cases. This will cover the connectivity. Where ADS replaced the 4G network solution with 5G-NSA solution. Also, it over new features added for ADS implementation. In particular, ADS performed the following implementations. First, the addition of a drone charging spot for the multiple drone trial and Internet Drone Operating System (IDrOS) validations. Second, Zenoh (Zenoh, n.d.) integration as a data transmission protocol for streaming data from the drone to the edge. And third, update of EagleEYE (Ardiansyah, Muhammad Febrian and William, Timothy and Abdullaziz, Osamah Ibrahiem and Wang, Li-Chun and Tien, Po-Lung and Yuang, Maria C, 2020) processing pipeline to better support multiple drone trials.

6.1. Experimental setup

The edge data center as well as the gNB/eNB combo small cell are located in the MIRC building. The MIRC building is an 8 floors high-rise building located inside the NCTU campus. The gNB/eNB combo small cell is mounted on the sixth floor of the MIRC building, while the edge data center is located in the basement of the MIRC building. In Figure 6-1 above, the complete connectivity mapping at the edge in support of the ADS mission is depicted. The edge data center is also connected with the BASS instance in 5TONIC lab in Spain for edge infrastructure management and orchestration. At the edge, is where all of the components supporting the disaster relief response system are deployed. More details on each of the components have been previously reported in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021), and D3.2 (D3.2, 2021).

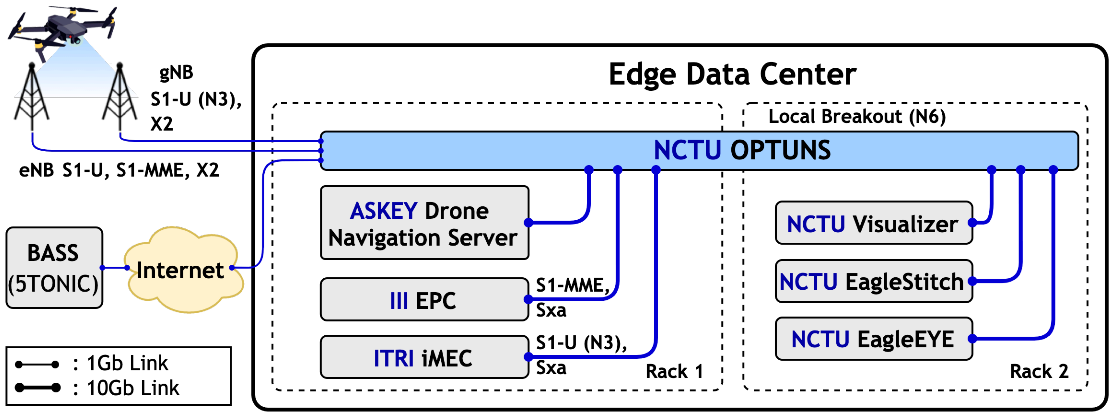


FIGURE 6-1 ADS Network connectivity mapping

Also, the utilized drone model is shown in Figure 6-2 and Figure 6-3. In Figure 6-2, the drone support 5G-NSA and Ubiquiti connectivity. 5G-NSA is mainly used to send sensory data and video streaming (i.e. range from 848x480 to 1920x1080 full HD based on drone camera) from drones to edge. While the Ubiquiti is useful to share information among drones whenever needed (see Figure 6-3). More details of drone hardware including the battery, weight, and camera are provided in D3.2 (D3.2, 2021).

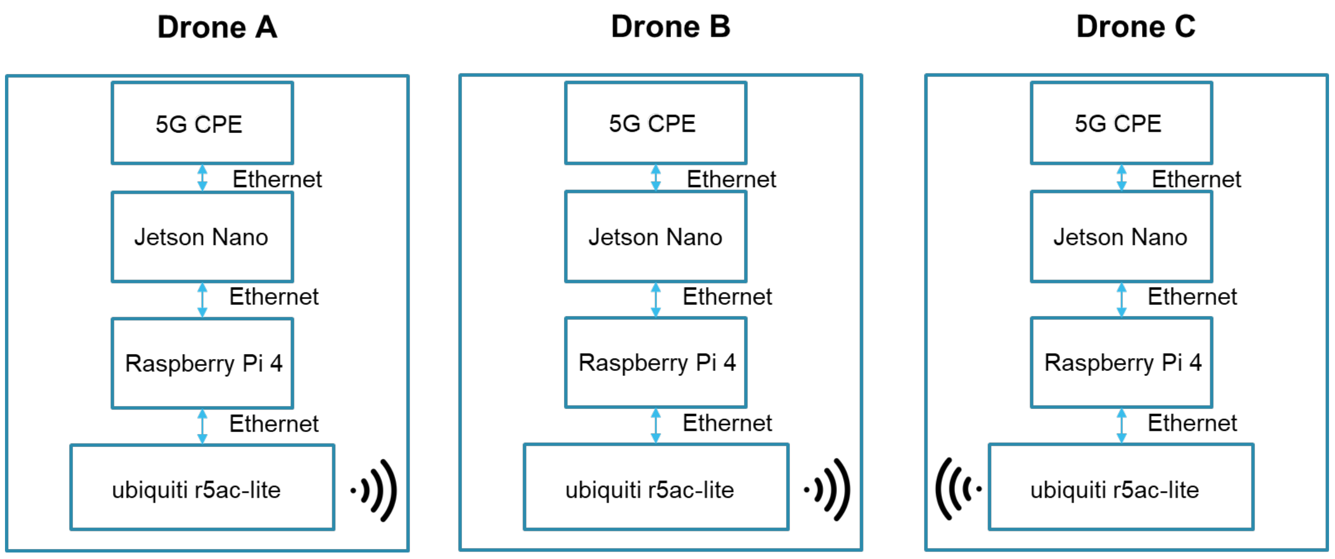


FIGURE 6-2 ADS drone connectivity mapping

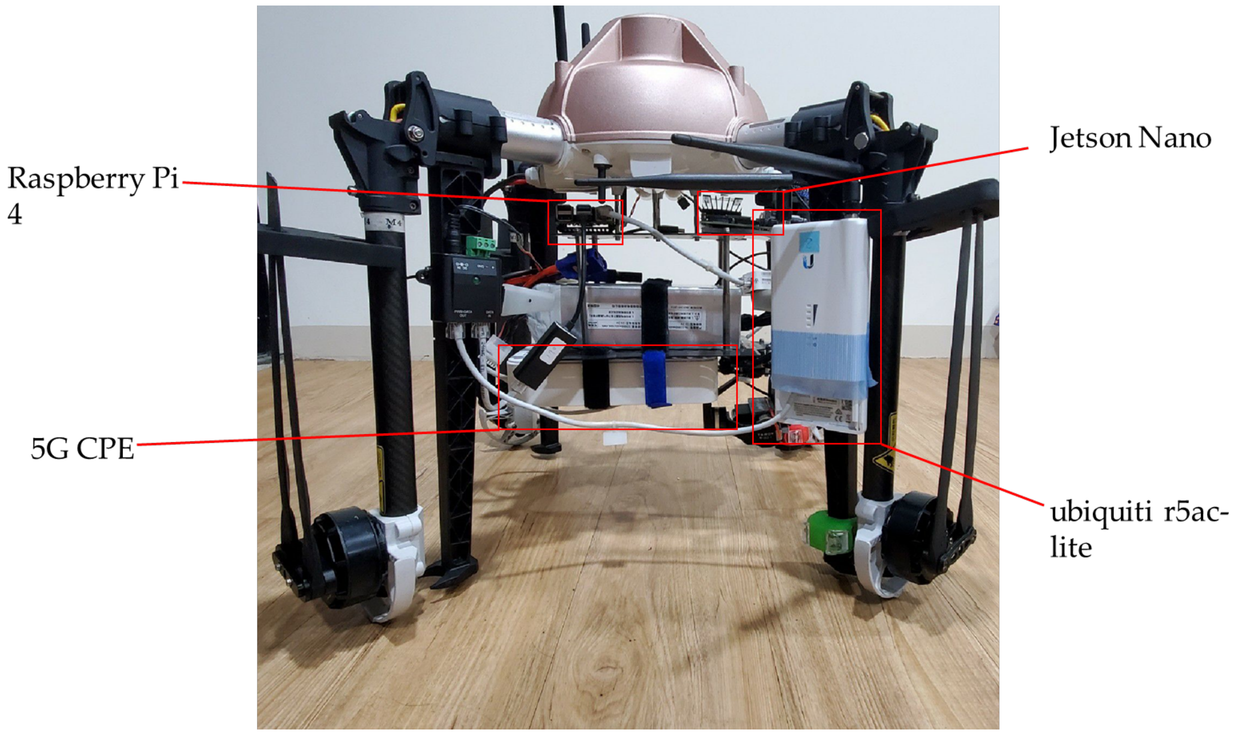


FIGURE 6-3 ADS Utilized drone

6.2. ADS-UC1: Drone Fleet Navigation

For ADSUC1, we present early results on the performance of the iDrOS drone operating system. We specifically focus on iDrOS' ability to migrate application components from drone to edge. In addition, we present the updated DCAS adopted at the edge with the new drone navigation server.
For IDrOS, we take as an example the migration of an application component implementing a dummy image processing functionality for object detection. We emulate the behaviour of a companion computer running onboard a drone using a RaspberryPI 4 device and simulate an edge installation using an Intel Canyon NUC i7 running at 3.4GHz. The Raspberry PI is connected to a wireless network

created by a Netgear 5G mobile router. The 5G network is the one of Vodafone Italia (5G trial in Milano). The Intel Canyon NUC is connected to the Vodafone backbone network via Ethernet. The average round-trip time between the Raspberry PI and the Intel Canyon NUC is 3.8ms before the start of the experiments. We use mahimahi (Mahimahi) to emulate different degrees of packet losses.
Figure 6-4 shows the experimental setup. The image processing components migrate from the "Data Processing" package on board the drone to the same class on the edge device. We measure the execution time of the image processing functionality as a function of the number of executions per minute and the times for performing the offloading, that is, from the drone to the edge, and inloading, that is, from the edge to the drone of the component from/to the drone as a function of the number of state variables storing the component state, which we call "state variables" from now on, that need to be migrated.

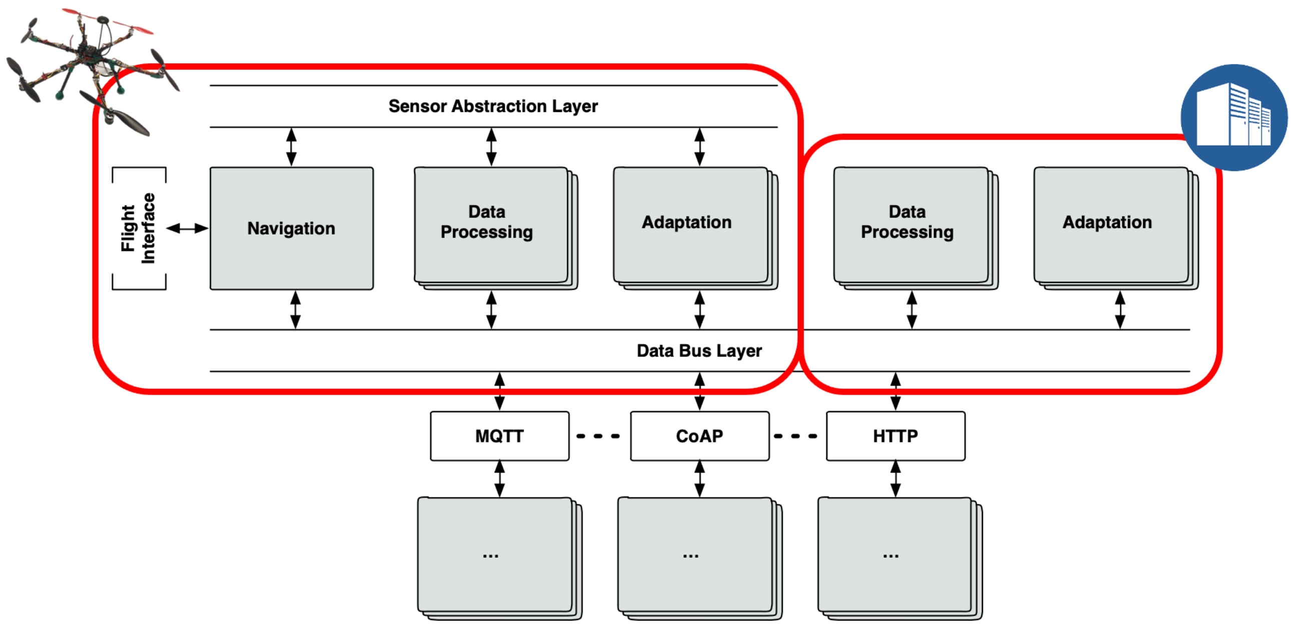
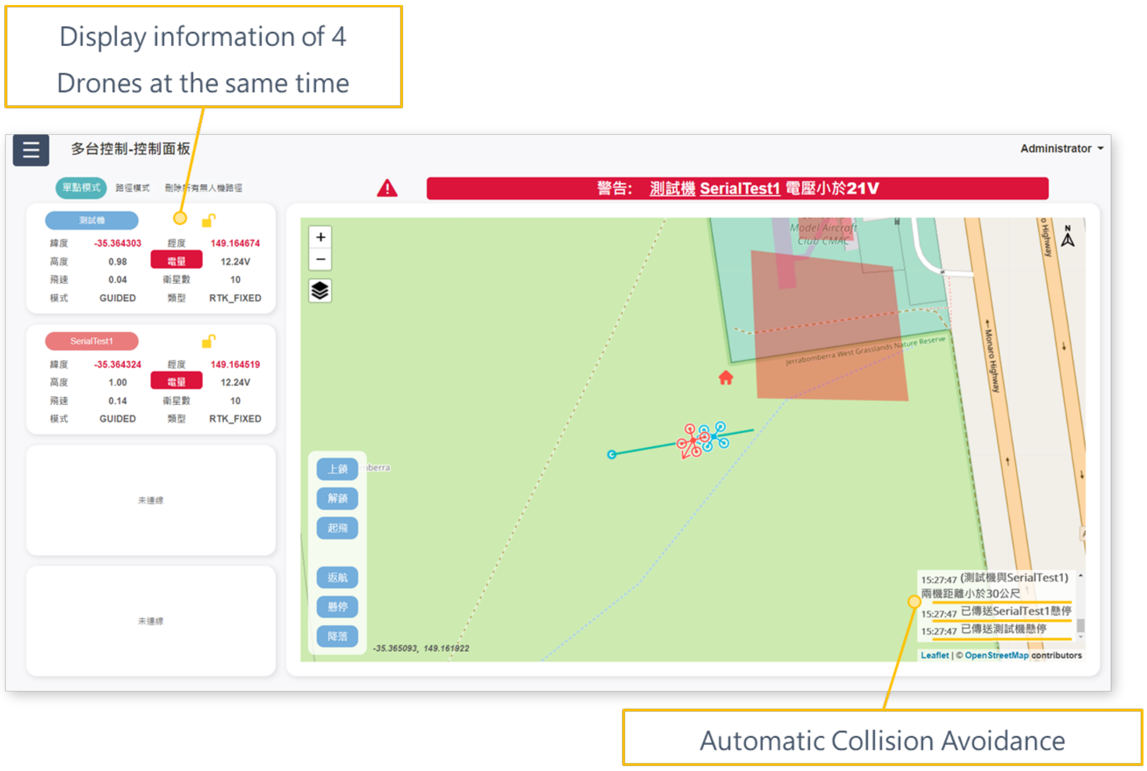


FIGURE 6-4: iDrOS experimental setup

As a baseline for comparison, we build a functionally equivalent implementation that uses the Python library Pyro4 (Python libarary Pyro4) to remotely expose the image processing functionality and we test the same range of queries per time unit.

For DCAS, the experimental setup is shown in Figure 6-5 for two drones. Basically, multiple drones are monitored at the same time to support collision avoidance between drones. In-flight, when the distance between drones is too short, the system will automatically issue avoiding commands. DCAS mechanism is exactly the same as elaborated in D3.2 (D3.2, 2021) but we move it to edge within the drone navigation server. This will allow us to utilize 5G-NSA and improve the visibility of drones by human operators.


Display information of 4 Drones at the same time
多台控制-控制面板
Administrator
警告: 測試機 SerialTest1 電壓小於21V
測試機
SerialTest1
上鎖
解鎖
起飛
返航
懸停
降落
Automatic Collision Avoidance

15:27:47 (測試機與SerialTest1) 兩機距離小於30公尺
15:27:47 已傳送SerialTest1懸停
15:27:47 已傳送測試機懸停

FIGURE 6-5 DCAS experimental setup

6.2.1. Mission Scenario and Flow

In ADS-UC1, the mission operation operators three drones. The drone navigation client is connected to the Drone navigation server through the mobile network. This will allow for drone monitoring and dynamic control as shown in Figure 6-5. Besides, DCAS functionality is available on each drone and can connect it to the drone navigation client and have been tested previously as elaborated in D3.2. Now, DCAS functionality is adopted in drone navigation servers and it was tested in and will be elaborated in this section.

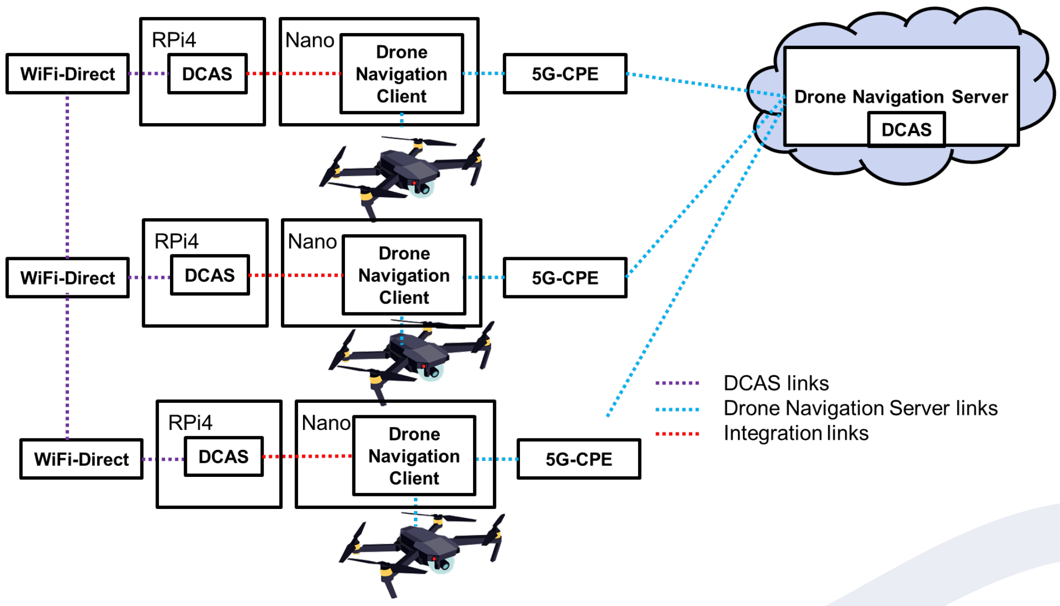


FIGURE 6-6: DCAS validation setup

In our validation, the drones fly on the ITRI campus. The Drone Navigation Server controls the drones to take off and sets them to fly to different destinations. During the flight, if the drones are too close to one another as depicted in Figure 6-7, the DCAS on the edge will detect potential collisions and eventually will take over the control of the affected drones for a limited period of time. This period is function of collision risk, as there is no risk, the control will be again with the drone navigation server. After the potential collision is resolved, the drone control returns to the Drone Navigation Server in order to continue the mission. During the field trial, three drones (i.e. Drone #1, Drone #2, and Drone #3) will be part of the drone fleet required to fly to certain locations as the relief mission required. Initially, two drones (Drone #1 and Drone #2) fly to perform this mission, and collision is detected and then the drone navigation server will perform DCAS. The mission still running, but Drone #1 was detected with a low battery by the drone navigation server. The drone navigation server will request Drone#3 to fly autonomously to replace Drone#1. Hence, Drone #3 will fly in the shortest path using DCAS to replace Drone #1 and continue the mission as shown in Figure 6-7.

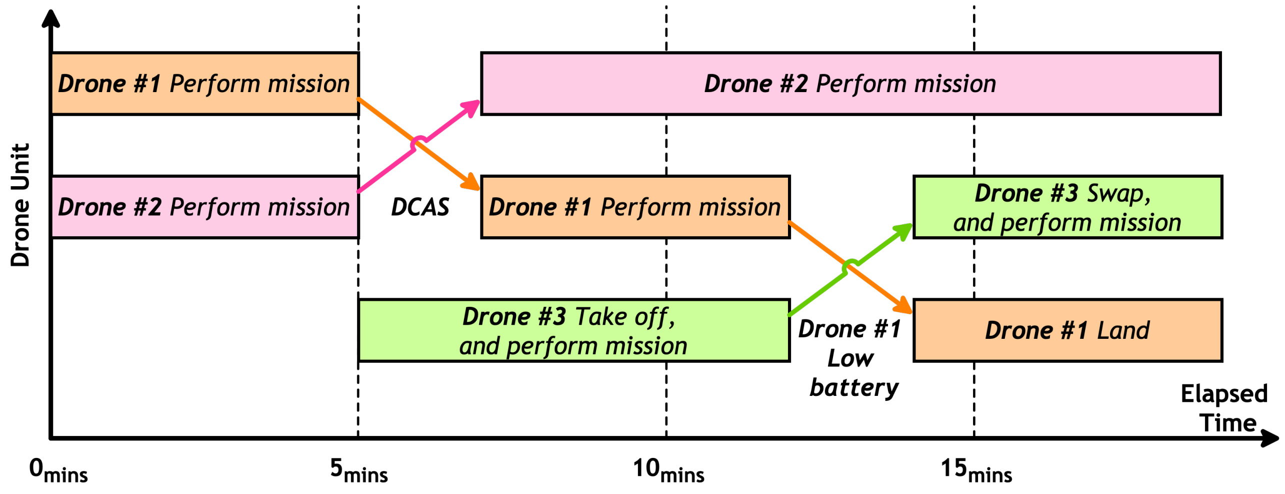


FIGURE 6-7 ADS-UC1 mission timing sequence

6.2.2. Experimental Results

In this section, the experimental results for Drone to Drone (Section 6.2.2.1), and IDroS System (Section 6.2.2.2) are reported.

6.2.2.1. Experiment A: Drone to Drone Communication

The evaluation is conducted on ITRI campus football field, the distance used between drones is approximately 7 meters and 15 meters, respectively. With respect to the latency performance evaluation, the system achieves an average of less than 3.2ms RTT (ping test; 100 samples). The results are collected for different drone arrangements as shown in Figure 6-8, Table 6-1 and
Table 6-2. The results show that the drone-to-drone communication system can support the packet transmission of DCAS in real-time. Having the DCAS deployed at the edge or the fog does not affect performance significantly. Also, the scalability test has been performed as shown in Section 6.2.3.

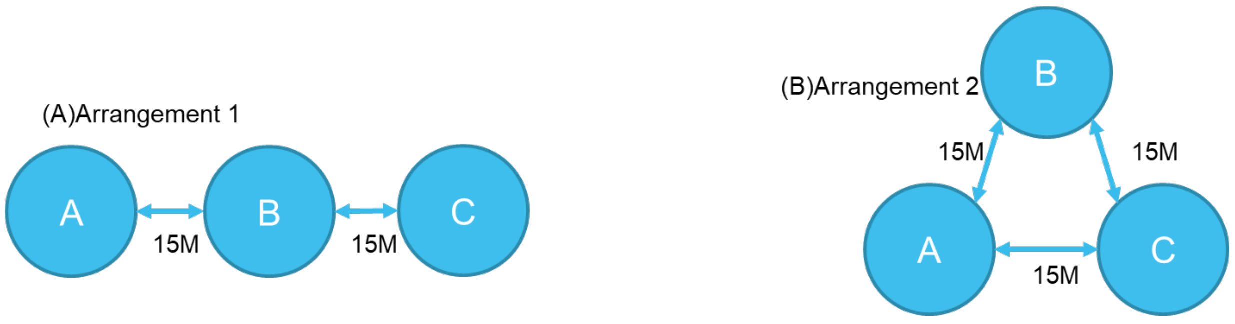


FIGURE 6-8: DCAS latency measurement setup

TABLE 6-1 DCAS RTT latency measurements with 7 meter spacing

	Arrangement 1	Arrangement 2
Drone_A~Drone_B	3.352ms	2.478ms
Drone_B~Drone_C	1.963ms	1.177ms
Drone_C~Drone_A	1.635ms	1.522ms

TABLE 6-2 DCAS RTT latency measurements with 15 meter spacing

	Arrangement 1	Arrangement 2
Drone_A~Drone_B	2.976ms	2.913ms
Drone_B~Drone_C	1.519ms	1.545ms
Drone_C~Drone_A	1.624ms	1.365ms

6.2.2.2. Experiment B: iDrOS Migration

The results of the comparison between IDrOS and Pyro4 experiments are reported in Figure 6-9: the performance of IDrOS is slightly better than the one of Pyro4. On average, the execution time of IDrOS is ~450 milliseconds shorter than the execution time of the Pyro4 implementation. We repeated the experiment 10 times and averaged the results. The standard error of IDrOS means is 0.169 seconds, the standard error of Pyro4 means is 0.33 seconds.

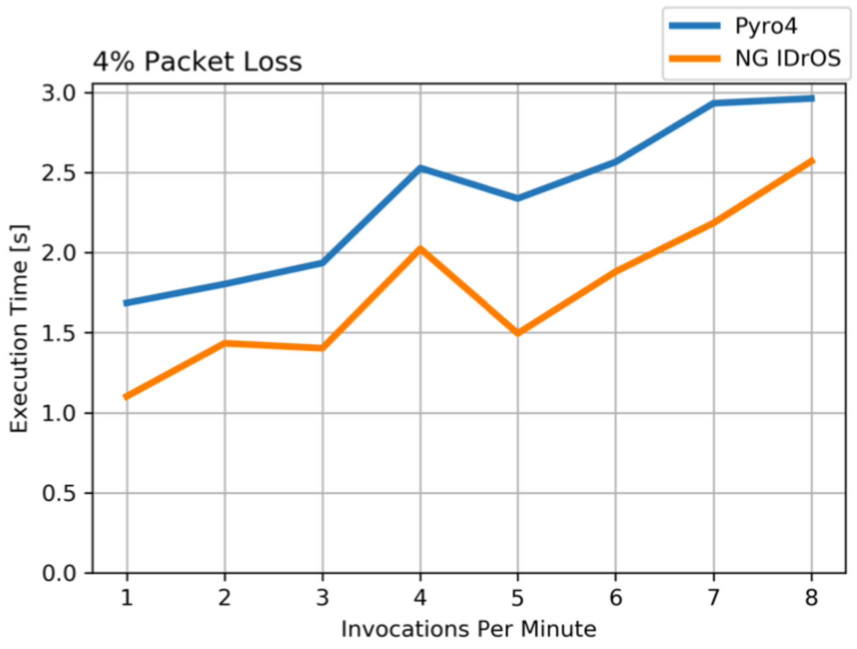


FIGURE 6-9 IDRoS execution times when migrating components

Figure 6-10 reports the results when migrating components to and from the edge. From the graphs, we can derive the following conclusions:

- The number of state variables (see Section 6.2) in the range we tested does not significantly affect the in loading and offloading times, since there is no increasing trend in the average times.
- The average in loading time (see Section 6.2) is higher than the average offloading time because it is influenced by the time needed to restore the variables, and in our experiments, it is driven by the performance of the Raspberry.
- The packet loss rate in the interval we evaluated does not significantly affect the in loading and offloading times, because the network transfer time, including TCP retransmissions, is negligible with respect to the time needed to serialize, deserialize, and restore the state.

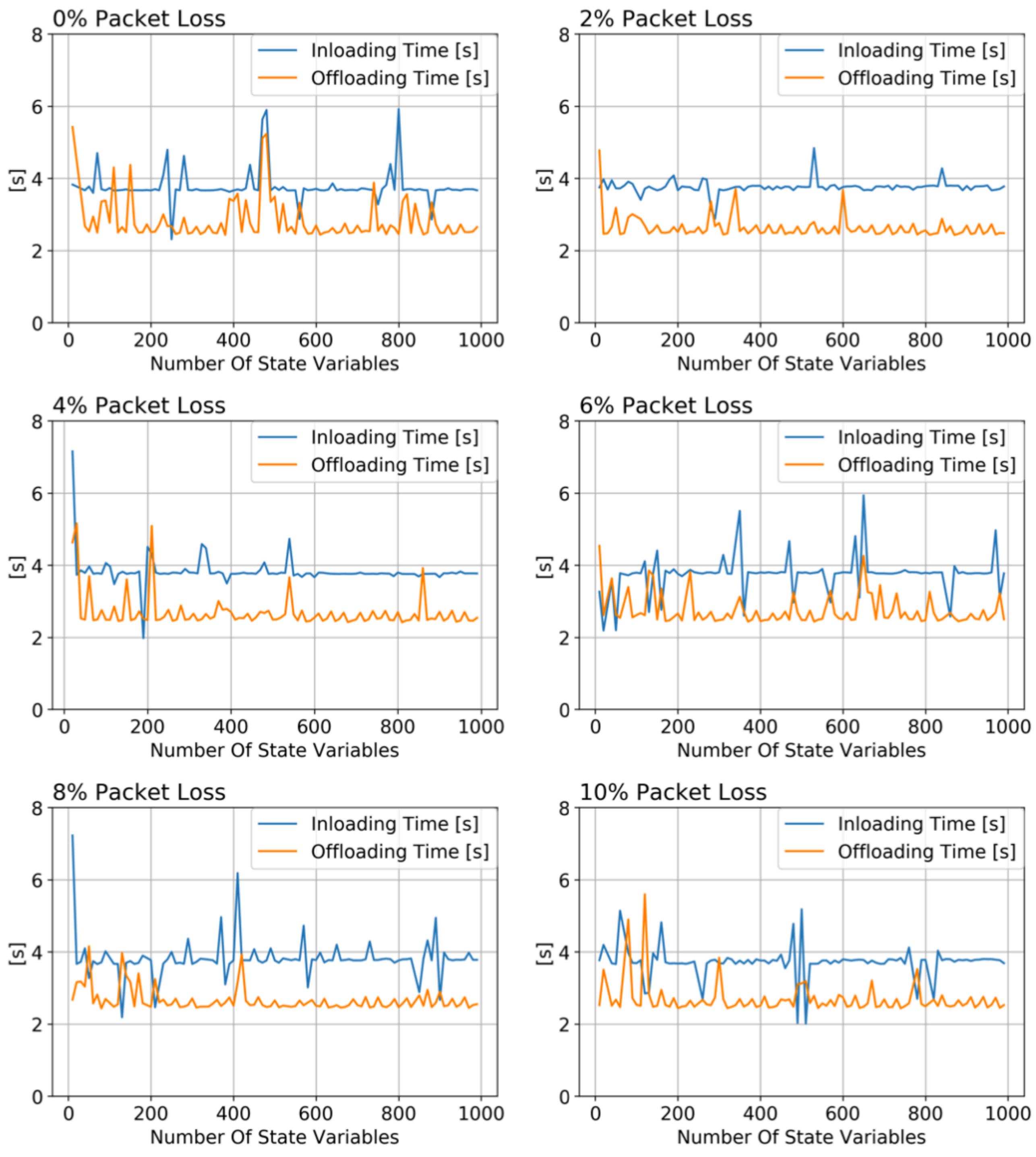
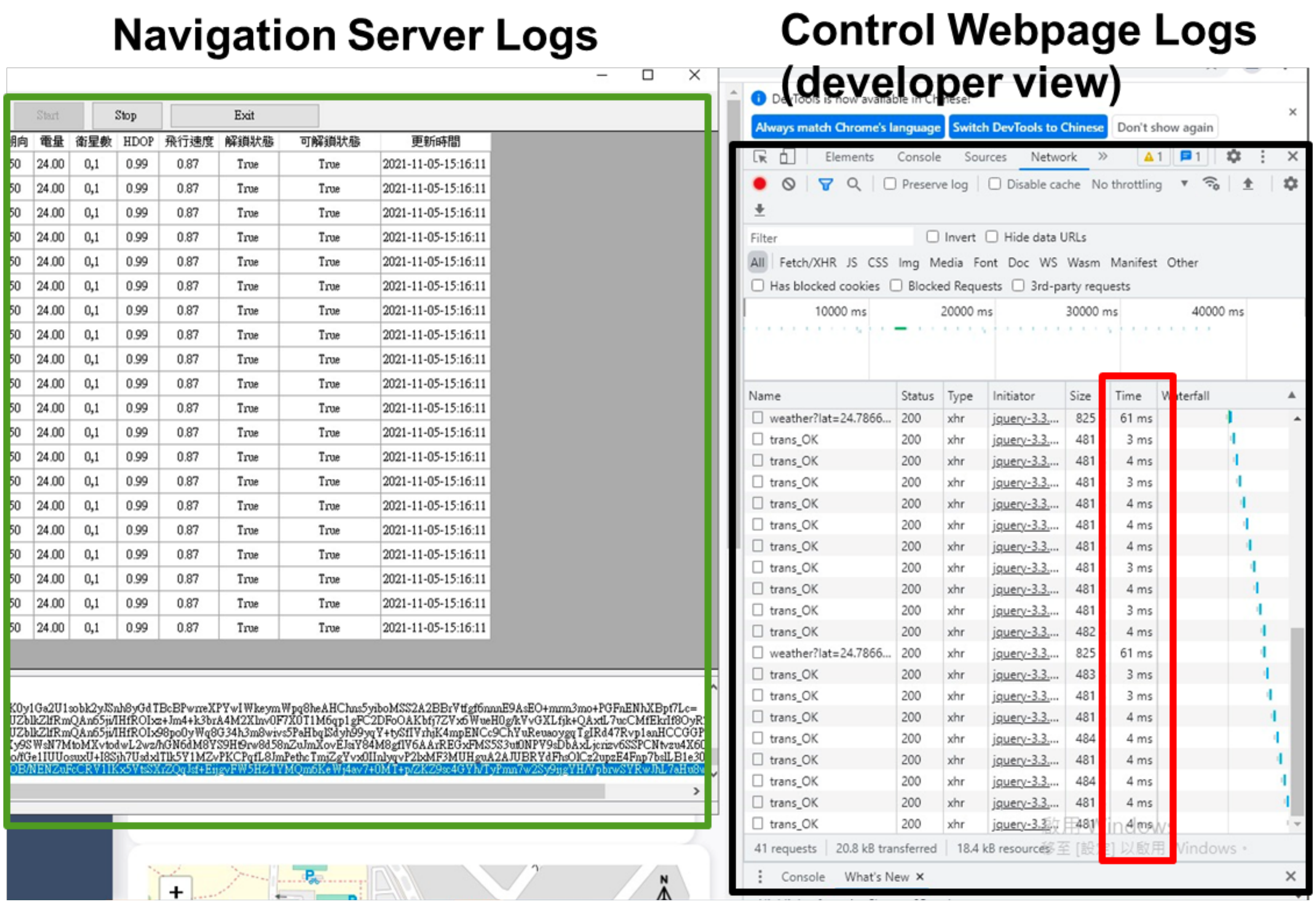


FIGURE 6-10 IDRoS migration performance

6.2.3. Scalability

For the drone fleet, it is important to be able to serve multiple drones while utilizing 5G-DIVE platform and passing traffic from drone to edge with 5G-NSA connectivity. Taking that into consideration, we observe that the navigation server sends and receives information normally including DCAS messages using 20 emulated drones and 50 emulated drones as shown in Figure 6-11 and Figure 6-12, respectively. Where the time displayed in the red box is the latency from the control webpage to the navigation server. Also, the green box shows information about the navigation server interface. The results run for more than 20 minutes (i.e. expected fly time of the drones) and the drone navigation server has the connection established with all the drones during this stress test. This means the drone navigation server is scalable to perform the drone fleet navigation during the disaster mission.



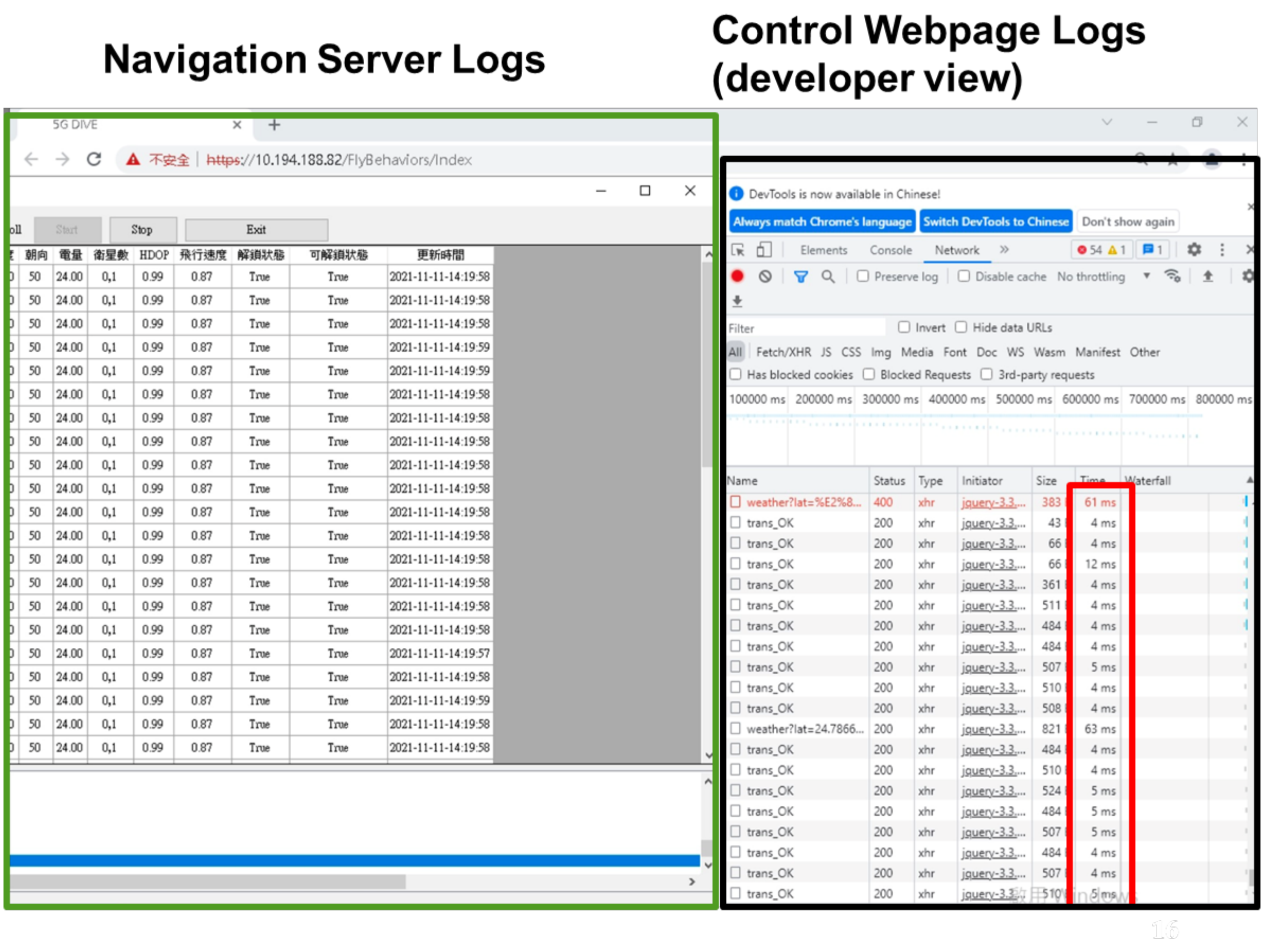
朝向	電量	衛星數	HDOP	飛行速度	解鎖狀態	可解鎖狀態	更新時間
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11
50	24.00	0,1	0.99	0.87	True	True	2021-11-05-15:16:11

K0y1Ga2U1sobk2yJSnh8yGdTBcBPwrreXPYwIWkeymWpq8heAHChns5yiboMSS2A2BBrVtfgf6nnnE9AsEO+mrm3mo+PGFnENhXBpf7Lc=
UZblkZlfRmQAn65ji/lHfROIxz+Jm4+k3brA4M2Xlnv0F7X0T1M6qp1gFC2DFoOAKbfj7ZVx6WueH0g/kVvGXLfjk+QAxtL7ucCMfEkrIf8OyR
UZblkZlfRmQAn65ji/lHfROIx98po0yWq8G34h3m8wivs5PaHbqlSdyh99yqY+tySfIVrhjK4mpENCc9ChYuReuaoygqTgIRd47Rvp1anHCCGGP
Ky9SWsN7MtoMXvtodwL2wz/hGN6dM8YS9Ht9rw8d58nZuJmXovEJsiY84M8gfIV6AArREGxFMS5S3ut0NPV9sDbAxLjcrizv6SSPCNtvzu4X60
o/fGe1IUUosuxU+I8Sjh7UsdxlTlk5Y1MZvPKCPqfL8JmPethcTmjZgYvx0IInlyqvP2bxMF3MUHguA2AJUBRYdFhsOlCz2upzE4Fnp7bslLB1e30
OB/NENZuFcCRV1IKx5VtsSXrZQqJsf+EijgvFW5HZTYMQm6KeWj4av7+0MT+p/ZKZ9sc4GYh/TyPmn7w2Sy9ijgYH/VpbrwSYRwJhL7aHu8w

Name	Status	Type	Initiator	Size	Time	Waterfall
weather?lat=24.7866...	200	xhr	jquery-3.3....	825	61 ms	
trans_OK	200	xhr	jquery-3.3....	481	3 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	3 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	3 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	3 ms	
trans_OK	200	xhr	jquery-3.3....	482	4 ms	
weather?lat=24.7866...	200	xhr	jquery-3.3....	825	61 ms	
trans_OK	200	xhr	jquery-3.3....	483	3 ms	
trans_OK	200	xhr	jquery-3.3....	481	3 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	
trans_OK	200	xhr	jquery-3.3....	481	4 ms	

41 requests | 20.8 kB transferred | 18.4 kB resources

FIGURE 6-11 Scalability testing with 20 emulated drones

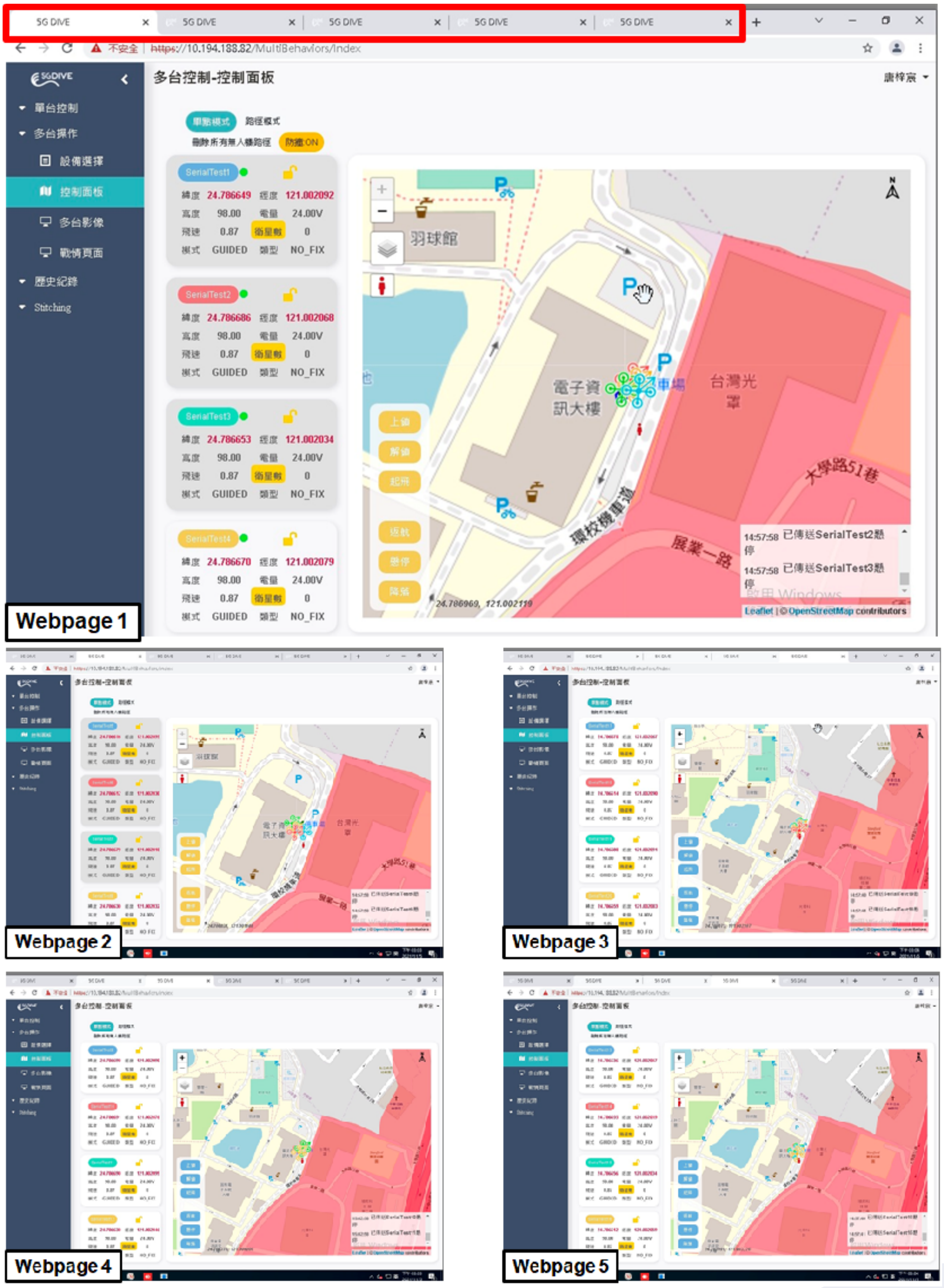


朝向	電量	衛星數	HDOP	飛行速度	解鎖狀態	可解鎖狀態	更新時間
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:59
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:59
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:57
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:59
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58
50	24.00	0,1	0.99	0.87	True	True	2021-11-11-14:19:58

Name	Status	Type	Initiator	Size	Time	Waterfall
weather?lat=%E2%8...	400	xhr	jquery-3.3....	383	61 ms	
trans_OK	200	xhr	jquery-3.3....	43	4 ms	
trans_OK	200	xhr	jquery-3.3....	66	4 ms	
trans_OK	200	xhr	jquery-3.3....	66	12 ms	
trans_OK	200	xhr	jquery-3.3....	361	4 ms	
trans_OK	200	xhr	jquery-3.3....	511	4 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	507	5 ms	
trans_OK	200	xhr	jquery-3.3....	510	4 ms	
trans_OK	200	xhr	jquery-3.3....	508	4 ms	
weather?lat=24.7866...	200	xhr	jquery-3.3....	821	63 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	510	4 ms	
trans_OK	200	xhr	jquery-3.3....	524	5 ms	
trans_OK	200	xhr	jquery-3.3....	484	5 ms	
trans_OK	200	xhr	jquery-3.3....	507	5 ms	
trans_OK	200	xhr	jquery-3.3....	484	4 ms	
trans_OK	200	xhr	jquery-3.3....	507	4 ms	
trans_OK	200	xhr	jquery-3.3....	510	5 ms	

FIGURE 6-12 Scalability testing with 50 emulated drones

Besides, during this scalability test, the control webpage can be displayed. In particular, the webpage can control up to 4 drones at a time. Each control page controls 4 drones taking into account the safety (elaborated in D3.1 (D3.1: Definition and setup of vertical trial sites, 2020)) considerations for the naked eye of the drone operator. The 20 drones are controlled in multiple pages as shown in Figure 6-13.



SerialTest1
緯度 24.786649 經度 121.002092
高度 98.00 電量 24.00V
飛速 0.87 衛星數 0
模式 GUIDED 類型 NO_FIX

SerialTest2
緯度 24.786686 經度 121.002068
高度 98.00 電量 24.00V
飛速 0.87 衛星數 0
模式 GUIDED 類型 NO_FIX

SerialTest3
緯度 24.786653 經度 121.002034
高度 98.00 電量 24.00V
飛速 0.87 衛星數 0
模式 GUIDED 類型 NO_FIX

SerialTest4
緯度 24.786670 經度 121.002079
高度 98.00 電量 24.00V
飛速 0.87 衛星數 0
模式 GUIDED 類型 NO_FIX

14:57:58 已傳送SerialTest2暫停
14:57:58 已傳送SerialTest3暫停

24.786969, 121.002119

FIGURE 6-13 Control webpage of 20 emulated drones

6.3. ADS-UC2: Intelligent Image Processing for Drones

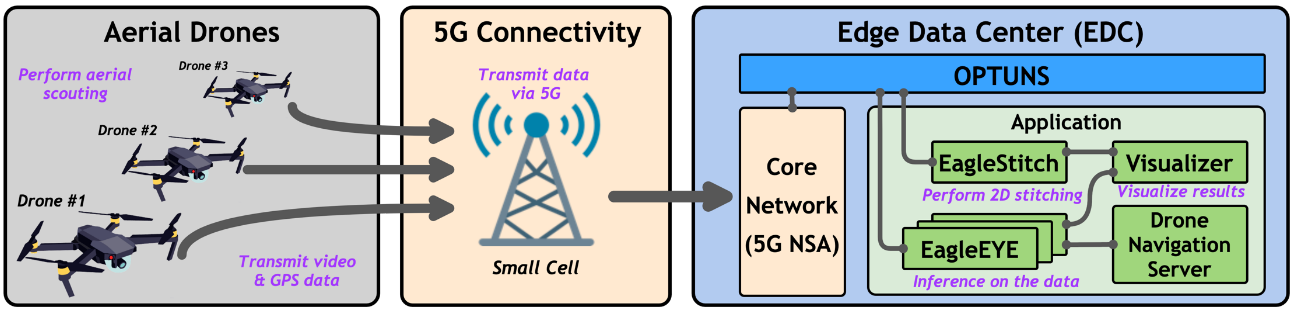


FIGURE 6-14 Overview of ADS-UC2 system

Figure 6-14 above provides an overview of the disaster relief response system for ADS-UC2. The system consists of three main components, which are the aerial drones, 5G connectivity solution, as well as edge. In the edge is where all of the intelligent application is deployed and run. In this section, the mission scenario and flow (Section 6.3.1), results on the key modules (Section 6.3.2), DEEP integration experiences (Section 6.3.3), as well as key modules scalability (Section 6.3.4) are of focus. Results on 5G connectivity solution can be found in Section 2.2. The key modules involved in the disaster relief response system involved are the Drone Data Processor system, the EagleEYE system, as well as the EagleStitch System.

6.3.1. Mission Scenario and Flow

In ADS-UC2, the goal of the disaster relief response system is to detect and localize Persons in need of Help (PiH), as well as to provide a view of the surrounding area. In the mission, 3 aerial drones are utilized to perform aerial scouting around the MIRC high-rise building located inside NCTU. In the mission, the PiH is located on the 6th floor and the right side of the MIRC building.

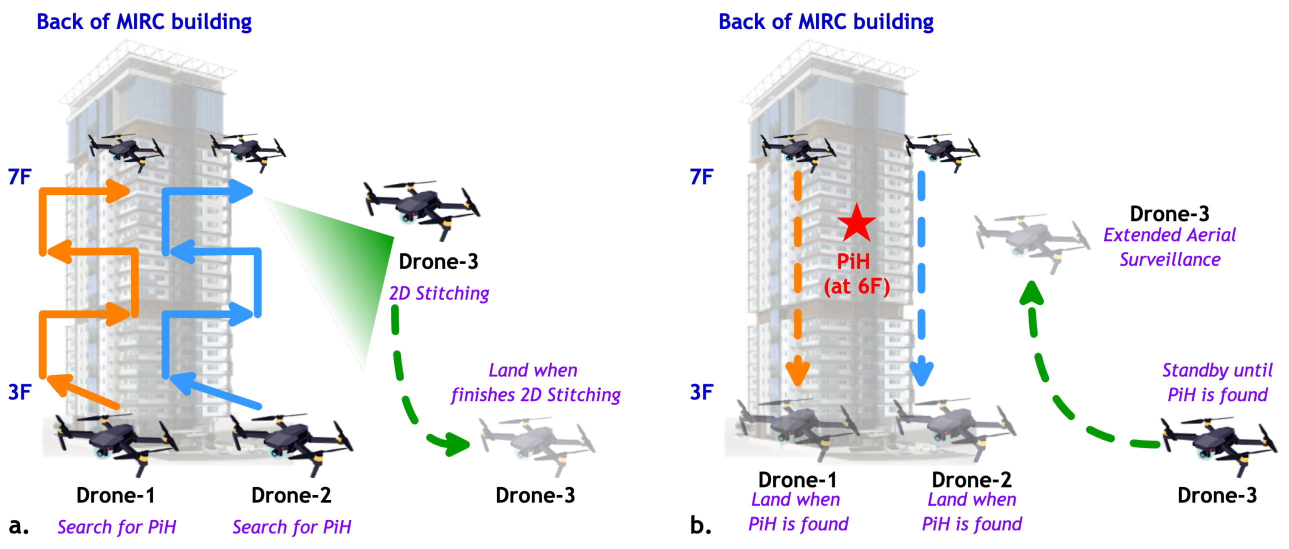


FIGURE 6-15 ADS-UC2 mission scenario and flow

When the mission starts, 3 drones are dispatched to the area. Drone-1, and Drone-2 are dispatched to perform aerial building surveillance. Drone-1 will perform aerial surveillance on the left side of the building, while Drone-2 performs aerial scouting on the right side of the building. The scouting is performed by continuously flying in a zig-zag pattern, floor-by-floor. Drone-1, and drone-2 continuously perform surveillance until the key modules running at the edge (EagleEYE) detect and localize PiH in the building (Figure 6-15a). When a PiH is detected, drone-1 and drone-2 will be instructed to update their trajectory and then perform landing or continue performing PiH detection and localization at other parts of the building. During the surveillance mission, drone-3 is also dispatched at the same time, but with a different goal. The drone-3 goal is to perform panorama image 2D stitching of the surrounding mission area, as well as to perform extended aerial surveillance. Providing a better view of the surrounding area for the rescue team. The 2D stitching mission is executed first, and then when drone-3 finishes this mission it will land and is set on standby mode. In standby mode, drone-3 waits for PiH location from the edge. When a PiH is successfully located, waypoints will be sent to drone-3 by the drone navigation server running at the edge. Drone-3 will then autonomously fly to the location of drone-3 to perform extended aerial surveillance (Figure 6-15b). This is to provide the rescue team with live streaming of the PiH to further assess the situation, to communicate with the PiH, or to deliver a first aid kit to the PiH.

During the runtime of the mission, aerial surveillance video will be streamed directly from the drone to the edge via the 5G wireless communication medium. The resolution of the streamed video varies from 848x480 up to 1920x1080, depending on the wireless network quality as well as the drone camera. In the trial itself, the 5G gNB is installed on the 6th floor of the building. The gNB itself is directly connected to the edge located in the basement of the building. In the real world, the gNB will be the one located at the perimeter of the area, while the edge can be any edge located closest to the disaster impacted area. In general, the mission usually lasts around 15-20 minutes. Figure 6-16 below showcases the breakdown of the trial mission performed by drone-1, drone-2, and drone-3 as well as its timing sequence. Note that the timing numbers are for simulation purposes and this number will change during the execution in the real world.

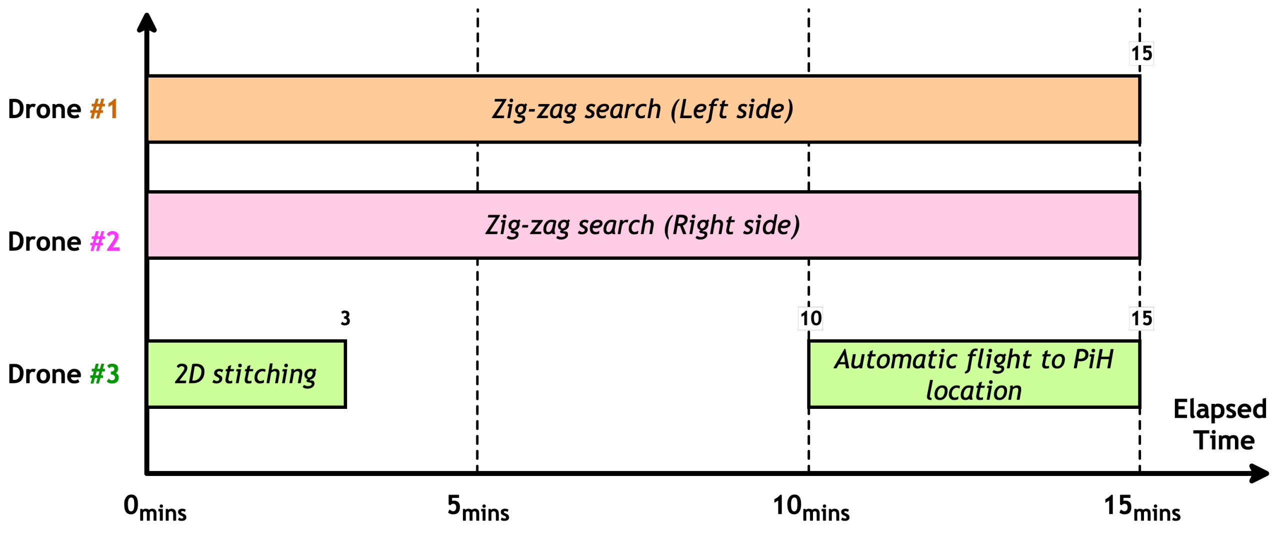


FIGURE 6-16 ADS-UC2 mission timing sequence

6.3.2. Experimental Results

In this section, the experimental results for the Drone Data Processor system (Section 6.3.2.3), EagleEYE system (Section 6.3.2.1), as well as EagleStitch System (Section 6.3.2.1) are reported. A brief overview of each system is also provided in the respective sub-section.

6.3.2.1. Experiment A: Processing Latency of EagleEYE System

In Figure 6-17, an overview of EagleEYE key modules is shown. EagleEYE system is used to perform Person in need of Help (PiH) detection. The goal of the PiH detection is to detect and locate PiH in a disaster-impacted area. A short overview of each of the EagleEYE key modules are the following:

1. **Data Offloader:** A round-robin based algorithm used to offload incoming image frames to any available Dual Object Detection module.
2. **Dual Object Detection:** CNN-based algorithm to detect 'Person' & 'Flag' objects
3. **PiH Candidate Selection:** Heuristic algorithm to check if the correlation between 'Person' & 'Flag' objects meets a set of criteria of a PiH object.
4. **Sorter:** Sort data according to frame sequence and drone ID
5. **PiH Persistence Validation:** Sliding window-based algorithm to determine if PiH object appear across a consecutive number of frames persistently.

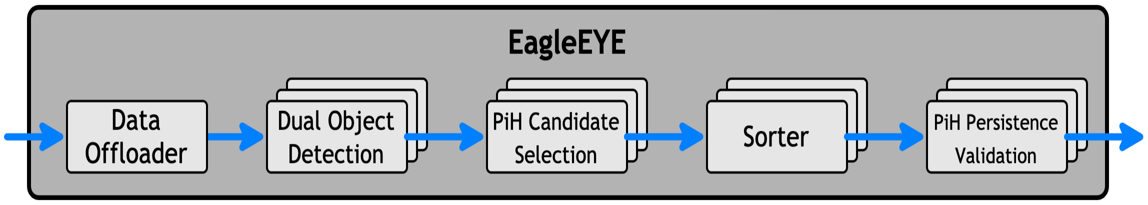


FIGURE 6-17 Overview of EagleEYE key modules

Further details on each of the EagleEYE key modules have been previously reported in D2.3 (D3.1: Definition and setup of vertical trial sites, 2020). The processing latency of each key module is summarized in Table 6-3. The processing latency is for the inference of a single image frame. In summary, 47.14 ms is needed to perform inferencing to detect and localize for PiH in a single image frame. Note that during system deployment, several modules in EagleEYE (e.g.: the Dual Object Detection module) are run in parallel to reduce the overall processing latency and to achieve real-time performance (<33 ms per image frame processing latency). Apart from that, the pipelining technique is also used to further reduce the processing latency. Figure 6-18 below showcases the output of EagleEYE system processing 2 drone input streams at once in real-time.

TABLE 6-3: EagleEYE key modules processing latency

EagleEYE Key Modules	Processing Latency
1. Data Offloader	1.873ms; std dev 2.008ms
2. Dual Object Detection	34.184ms; std dev 8.425ms
3. PiH Candidate Selection	6.778ms; std dev 3.764ms
4. Sorter	0.273ms; std dev 0.526ms
5. PiH Persistence Validation	4.032ms; std dev 1.984ms
Total	**47.14 ms**

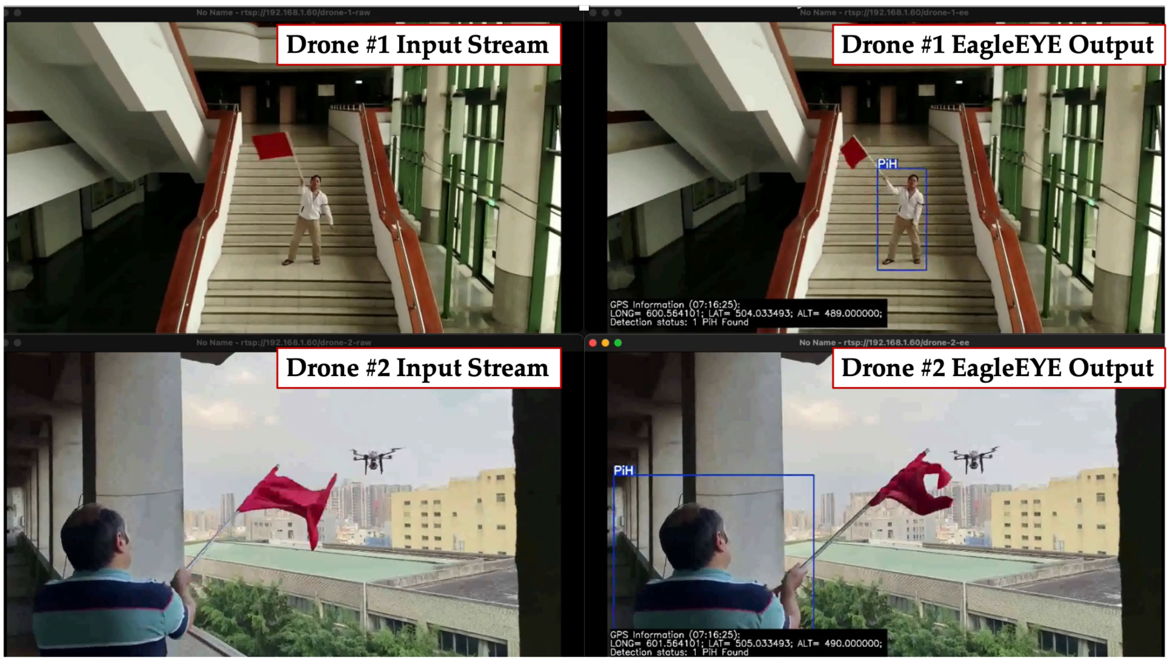

No Name - rtsp://192.168.1.60/drone-1-raw
Drone #1 Input Stream
No Name - rtsp://192.168.1.60/drone-1-ee
Drone #1 EagleEYE Output
PiH
GPS Information (07:16:25):
LONG= 600.564101; LAT= 504.033493; ALT= 489.000000;
Detection status: 1 PiH Found
No Name - rtsp://192.168.1.60/drone-2-raw
Drone #2 Input Stream
No Name - rtsp://192.168.1.60/drone-2-ee
Drone #2 EagleEYE Output
PiH
GPS Information (07:16:25):
LONG= 601.564101; LAT= 505.033493; ALT= 490.000000;
Detection status: 1 PiH Found

FIGURE 6-18 EagleEYE output

6.3.2.2. Experiment B: Processing Latency of EagleStitch System

In Figure 6-19, an overview of EagleStitch key modules is shown. EagleStitch system is used to perform 2D stitching of drone imagery gathered around the mission area. A short overview of each of the EagleStitch key modules are the following:

1. **Feature Matching:** To detect features in an image (e.g.: corner, curves).
2. **Image Matching:** To match for images that have the same features.
3. **Bundle Adjustment:** To bundle all images with the same features.
4. **Panorama Straightening:** To align the bundled images so that they are not slanted or rotated.
5. **Blending:** To adjust and correct the gain (brightness) of the images being stitched as well as to remove the seams (edges) in the stitched images.

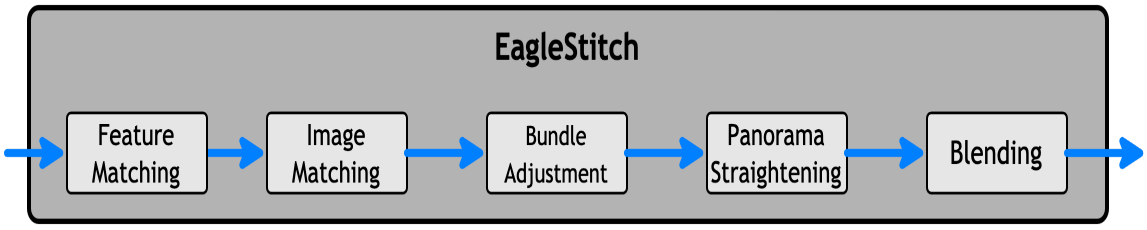


FIGURE 6-19 Overview of EagleStitch key modules

Further details on EagleStitch key modules can be found in D2.3 (D2.3: Final Specification of 5G-DIVE Innovations, 2021). The processing latency of each key module is summarized in Table 6-4.

TABLE 6-4 EagleStitch key modules processing latency

EagleStitch Modules	Processing Latency (ms)
1. Feature Matching	329
2. Image Matching	33
3. Bundle Adjustment	11
4. Panorama Straightening	95
5. Blending	289
Total	**757 ms**

The processing latency results are for the stitching of 2 image frames. In summary, 0.757s is needed to stitch for 2 image frames. Figure 6-20 below showcases the EagleStitch system output stitching 2 images taken by the drone of the surrounding trial area.

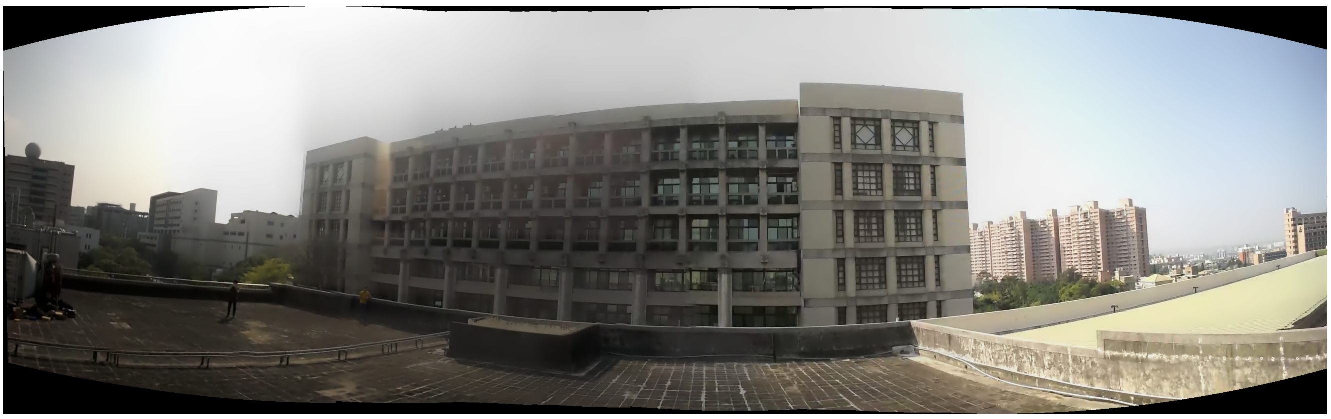


FIGURE 6-20 EagleStitch output

6.3.2.3. Experiment C: Processing Latency and Network Performance Metrics for Drone Data Processor System

The drone data processing system is used to inject metadata onto the captured images before sending them to the edge. This metadata information is used to differentiate between data coming from multiple drones. Apart from that, the drone data processor is also used to compress the captured images, reduce image file size, and save bandwidth. Table 6-5 provides a breakdown of the processing latency as well as network bandwidth consumption metrics for the drone data processor. The data transmission is done using Zenoh net via the pub/sub-publishing method.

TABLE 6-5 Drone data processor system performance metrics

Performance Metrics	Results
Processing latency (tagging, compression)	31.63 ms / image frame
Bandwidth consumption	1.1 MB / image frame

6.3.2.4. Experiment D: Stress Test of EagleEYE System and 5G-NSA solution

For stress testing the EagleEYE system, multiple tests of non-stop 2 hour PiH detection, and localization mission are executed. In the tests, drones footage of different locations is fed to the EagleEYE system to perform the PiH detection and localization. During the tests, EagleEYE system can work reliably and is able to detect and locate PiH. The stress testing is done with the complete setup as shown in Figure 6-14. In the real world, a drone-related mission can only last for about 20 minutes as the drone is limited by the battery capacity and the payload that the drone is carrying. In the stress test, we make sure that the complete system can run for far longer than 20 minutes and with that, a 2 hour test duration is picked. Figure 6-21 shows the visualization of EagleEYE, as well as the visualization from the drone navigation server during the 2 hour stress test. Notable, the results prove that EagleEYE is capable of detecting the PiH without any service interruption. Also, 5G-NSA solution is reliable to deliver HD images of the video stream without any interruption.

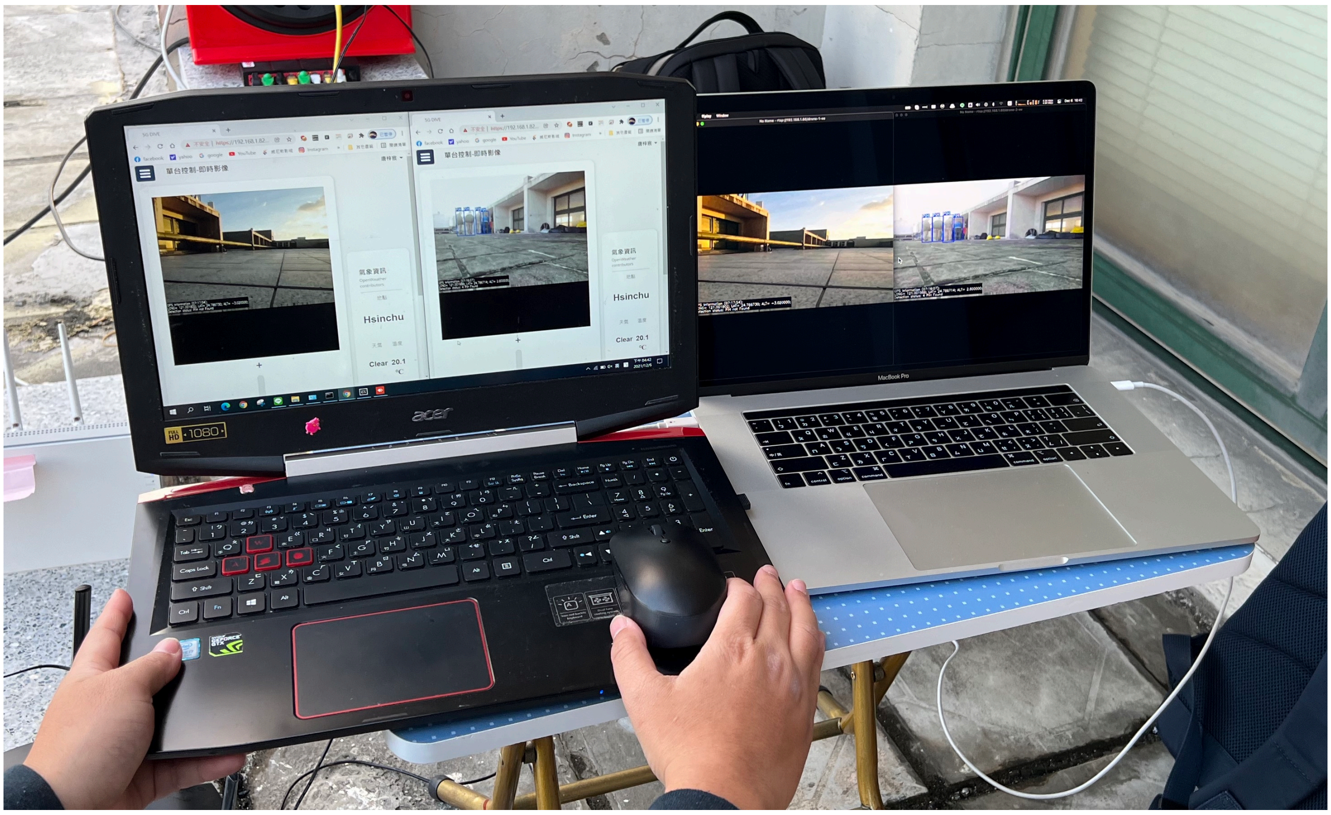


FIGURE 6-21 Visualization output of ADS-UC2 during two hours stress test

Furthermore, we also test the end-to-end solution in ASKEY labs for two hours with the setup as shown in Figure 6-22. The speed test runs for two hours without any interruption. In this setup, a downlink speed of up to 740Mbps, and an uplink speed of up to 88Mbps is recorded.

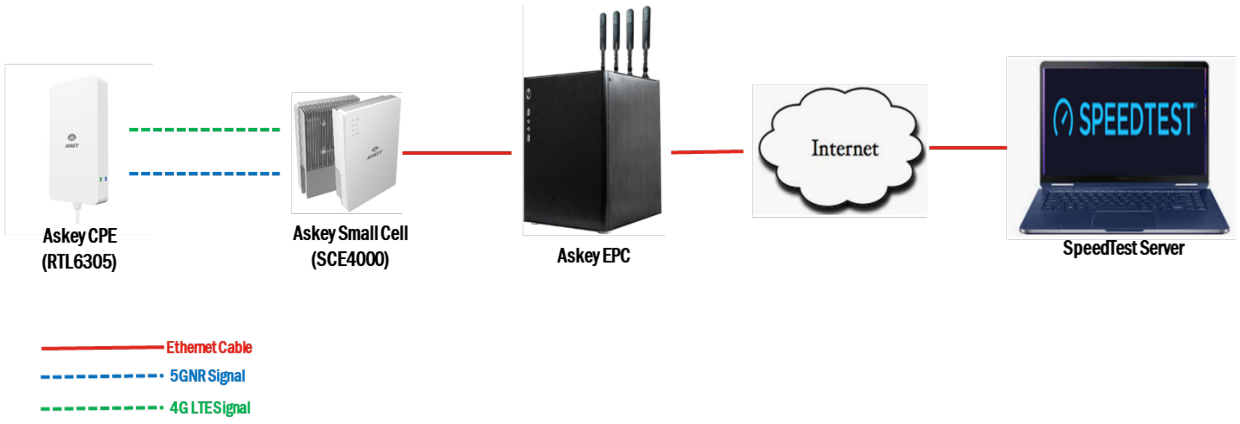


FIGURE 6-22 Stress test setup at ASKEY's lab

6.3.3. DEEP Integration Validation

In this section, a description on the DEEP platform integration experiences is detailed. The integration experiences for DASS, BASS, and IESS are detailed in Section 6.3.3.1, Section 6.3.3.2, and Section 6.3.3.3 respectively.

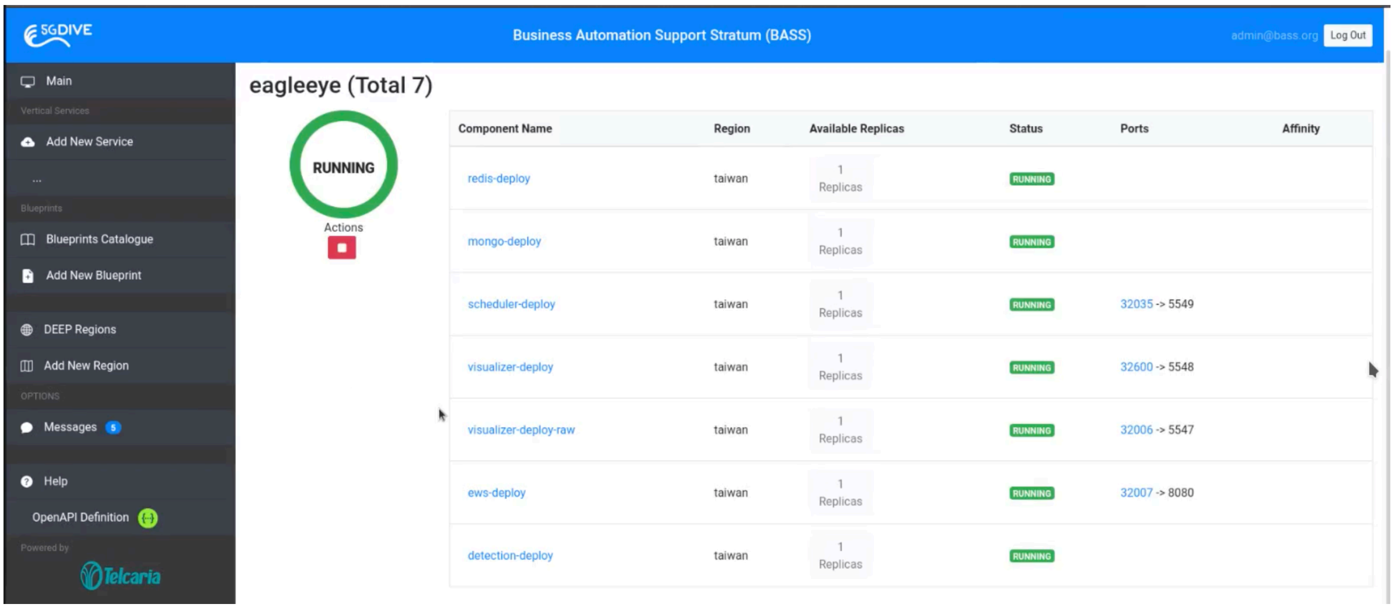
6.3.3.1. DASS

In general, with the integration of DASS, enhanced functionality in terms decentralized data dispatching and collection can be achieved. The DASS provides for a lightweight and efficient data transmission protocol for streaming data from the drones to the edge. With the DASS, the data collection process, posting, getting and subscribing to the collected data is simplified. In ADS-UC2, the DASS is extensively used by the Drone Data Processor system to perform data publishing from the drone to the edge. With results for the network bandwidth consumption listed in Table 6-5. In the ADS-UC2, the network bandwidth consumption will be limited by the frame rate of the camera mounted on the drone, as well as the image resolution of the camera. More details on the advantages of each feature of the DASS can be found in Section 3.2.

6.3.3.2. BASS

In general, with the integration of BASS, enhanced functionality in terms of application lifecycle management can be achieved. The BASS enables automatic and rapid service deployment of the ADS vertical service in the field. As a plus, deployment of the service in multiple different regions is also possible. Apart from that the BASS also provides for a common general user interface that is compatible across different resource orchestrators while also simplifying the descriptor file used to deploy for multiple system modules. And finally, the BASS provides for easy and ready-to-use monitoring probes that if combined with SLA enforcer will allow for automatic scaling of system modules. More details on the advantages of each feature of the BASS can be found in Section 3.1.

Figure 6-23 below captures the deployment of EagleEYE system via the BASS. The BASS is running at the 5TONIC lab in Spain, and it is deploying the EagleEYE system at the edge infrastructure that we have in Taiwan.



eagleeye (Total 7)

Component Name	Region	Available Replicas	Status	Ports	Affinity
redis-deploy	taiwan	1 Replicas	RUNNING		
mongo-deploy	taiwan	1 Replicas	RUNNING		
scheduler-deploy	taiwan	1 Replicas	RUNNING	32035 -> 5549	
visualizer-deploy	taiwan	1 Replicas	RUNNING	32600 -> 5548	
visualizer-deploy-raw	taiwan	1 Replicas	RUNNING	32006 -> 5547	
ews-deploy	taiwan	1 Replicas	RUNNING	32007 -> 8080	
detection-deploy	taiwan	1 Replicas	RUNNING		

FIGURE 6-23 EagleEYE System Deployment with BASS

6.3.3.3. IESS

At this moment, the integration with IESS is not yet complete. However, with the integration of IESS, we can expect for enhanced functionality in terms of automatic AI model training. The IESS, via AutoML, allows for automatic training of AI models for use in vertical service specific missions. For example, in ADS-UC2, the AI model is used to perform object detection. During the mission, there is a possibility that the object to be detected can change. And with that, the IESS can come in handy to help with automatic retraining of AI model to suit the mission needs. Apart from that, IESS also able to package the trained model for easy transport across regions for use in a similar mission in different regions. Previously trained models are also stored in a catalogue that future missions can utilize. More details on the advantages of each feature of the IESS can be found in Section 3.3.

6.3.4. Scalability

For EagleEYE, an architecture design that is capable of supporting multiple drone input streams is proposed. This proposed architecture is designed based on the incapability of the previous design to handle multiple drone streams. Two main concerns are observed in this work, first, when multiple streams are being offloaded and processed, the edge system was unable to distinguish the stream sources of the drones. Second, the processing is working asynchronously, while the system requires to visualize the processed image frames in order. As described in Figure 6-24, the blue blocks represent the new component in EagleEYE design. First, we introduced a data tagging approach on the drone side, where the image frames are pre-processed and are tagged with insightful information before being sent to the edge, such as drone ID, frame ID, and other important information. On the edge, we introduce the Data Offloader service to capture and pre-process the incoming image frames, while keeping the tuple of information sent by the drones. As for the Dual Object Detection (DoD) service, a new sub-module is added to map each image frame from different drones and make sure to process them to the correct next coming PiH Candidate Selection services. Afterward, for each processed image frame, it will be sorted out as a batch into Sorter service before finally are being sent to the Visualizer service. Finally, the end-users will be able to monitor the real-time results of each drone stream.

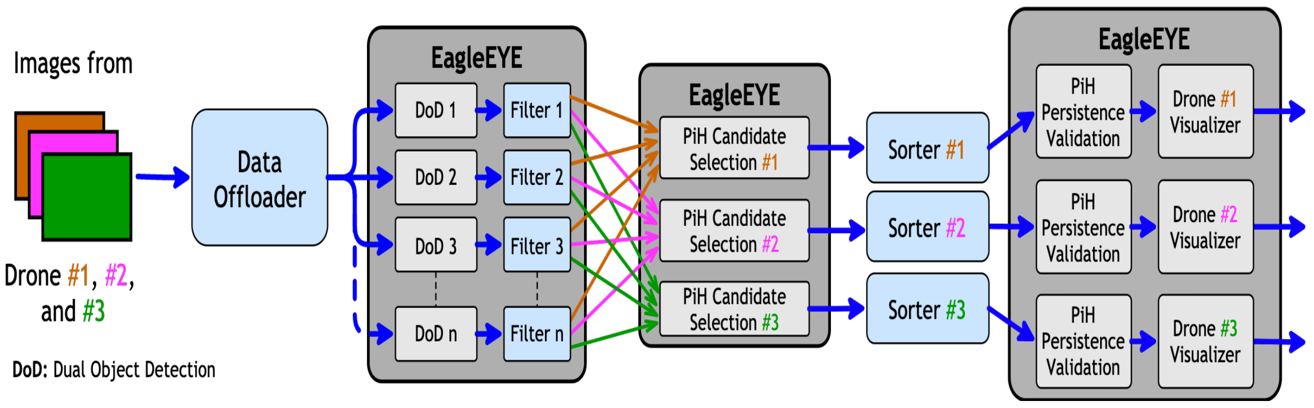


FIGURE 6-24 EagleEYE scalable architecture for multiple drone inputs

To validate our design, we performed a scalability test with six drone streams in an indoor environment. In this test, we pre-initialized EagleEye system with multiple Dual Object Detection services ready and prepared multiple JetsonNano which has a camera attached. Each camera will stream the pre-recorded video to visualize see whether the system can handle a real-time input stream or not. As shown in Figure 6-25, our new proposed EagleEYE design can smoothly handle multiple drone streams with no overlapping image frames from different drone sources.

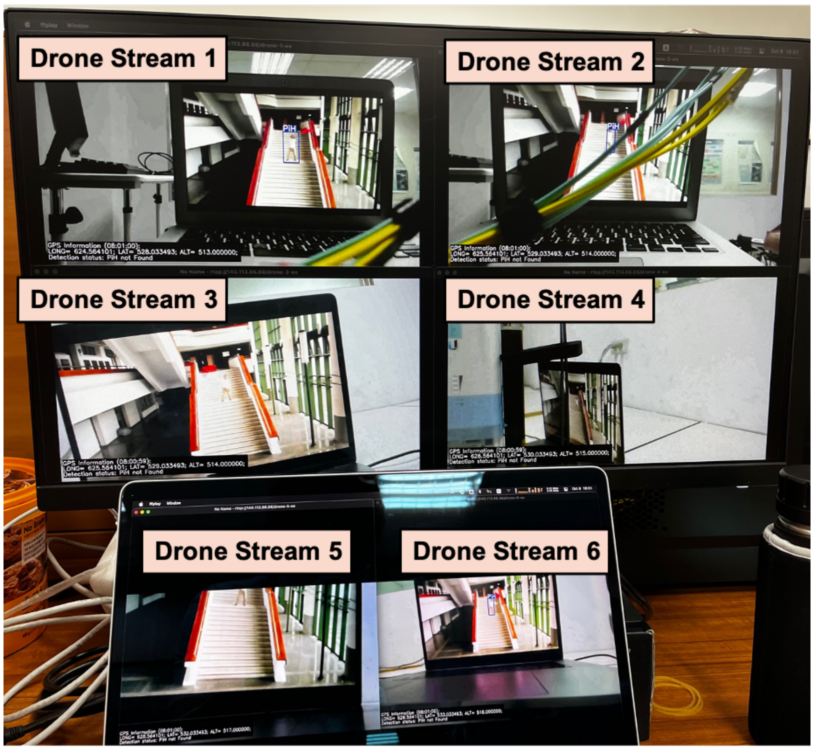


FIGURE 6-25 EagleEYE scalability test with 6 drones Input

7. Conclusions

This deliverable has presented a detailed view of the final validation results of different 5G-DIVE use cases. We have elaborated on the setup of Digital Twin (DT), Zero Defect Manufacturing (ZDM) and Massive Machine-Type-of-Communication (mMTC) use cases for Industry 4.0 (I4.0) trial. Also, the set of ADSUC1 and ADSUC2 remain unchanged. Then, mission scenarios and flows are presented for all the aforementioned use cases of I4.0 and ADS pilot. Indeed, the experiment results depicted how I4.0 and ADS pilots utilize the DEEP, fog and edge computing platform, and 5G connectivity in different levels toward better and reliable services. In addition, the integration for a single platform (in particular, BASS of DEEP) has been achieved. In summary, the achievements are summarised in Table 7-1.

TABLE 7-1: Final achievement per use case

Use Case	Main Achievements
DT	Developed a Movement Prediction AI model that recovers Digital Twin control commands. The Digital Twin use case was integrated with BASS of the DEEP for exploiting the SLA enforcement feature in order to allocate the optimal minimum amount of resources for the service. The use case was also integrated with the IESS for auto-packaging and auto-deployment of the Movement Prediction inference application. Finally, the use case was installed and deployed in the 5TONIC 5G trials where measurements were performed regarding application performance, Movement Prediction performance for commands predictions and vertical resource scaling.
ZDM	Integrated the testbed with 5G SA network, as well as other elements of the DEEP platform, namely the DASS and the BASS. Designed, developed and tested of an edge intelligent for the ZDM use case. A new object detection engine has been integrated with AWS Wavelength, Amazon's Edge Computing services (telco-Edge), where the ZDM object detection runs. An ATSSS xApp has been developed and performance results show 45% gains in achieved throughput at the network layer.
mMTC	Integrated the mMTC testbed with BASS of DEEP, further development and optimization of mMTC service implementation for trial measurements, installed and deployed the mMTC trial setup in 5TONIC integrated with DEEP, performed long-term mMTC trial regarding application layer performance with Ethernet and 5G, RF fingerprinting performance for intruder detection, orchestration and automation performance for enhanced system robustness, and scalability with resource

	utilization. The trial results show that the development of the mMTC use case achieves the objectives set in the project.
ADS-UC1	Integrated 5G-NSA solution for wireless communication from drone to edge. Utilized DCAS in the edge with newly developed drone navigation server. Executed drone fleet navigation with multiple drones using DCAS and considering a drone charging spot for the multiple drone trial. Demonstrated IDrOS software mobility features.
ADS-UC2	Integrated 5G-NSA solution for wireless communication from drone to edge. Integrated Zenoh as data transmission protocol for drone-to-edge data streaming. Integrated BASS for automatic service deployment and lifecycle management. Updated Drone Data Processor Module to better support multiple drone trials. Updated EagleEYE processing pipeline to better support scalability in handling multiple drone inputs. Completed EagleStitch development to provide panorama 2D stitching of the mission area.

8. Annex A- VSD in YAML format for mMTC

Representative example of the definition of a Vertical Service in the BASS. Can be codified both in JSON and YAML. This is the Vertical Service Descriptor for the mMTC use-case in YAML format.

```
name: mmtc
region: 5tonic-mmtc
components:
  - name: contiki-deployment
    numReplicas: 1
    imageRepository: docker.io/ericssonsics/5g-dive:contiki
    maxWaitTime: 1200
    exposedPorts:
      - 52001
      - 52002
      - 8080
    driverSpecific:
      type: KUBERNETES
      env:
        podIP: $ComponentIP
      networkPrivileged: true
  - name: contiki-phy
    numReplicas: 1
    imageRepository: eabsics/5g-dive:contiki_phy_test2
    maxWaitTime: 3600
    exposedPorts:
      - 52001
      - 52002
      - 53001
      - 53002
    driverSpecific:
      type: KUBERNETES
      env:
        contikiclusterip: contiki-deployment
        IID: '1'
        PUB_PORT: '53001'
        CONTIKI: contiki-deployment
        PULL_PORT: '53002'
  - name: fingerprinting
    numReplicas: 1
    imageRepository: ericssonsics/5g-dive:finger
    maxWaitTime: 1800
    exposedPorts:
      - 55002
    driverSpecific:
      type: KUBERNETES
      env:
        contikiclusterip: contiki-deployment
        THRESHOLD: '0.7'
```

9. References

GUEANT, V. (2021). *Iperf - The TCP, UDP And SCTP Network Bandwidth Measurement Tool.* Retrieved from https://iperf.fr/.

O. Liberg etal. (2017). *Cellular Internet of Things: Technologies, Standards, and Performance.* Elsevier Science Publishing Co Inc.

3GPP. (n.d.). *TS 22.104 v17.2.0: Service requirements for cyber-physical control applications in vertical domains, 2019-12.*

GNURadio webpage. (n.d.). Retrieved from https://www.gnuradio.org/

LoRa webpage. (n.d.). Retrieved from https://www.semtech.com/lora

Pycom Fipy webpage. (n.d.). Retrieved from https://docs.pycom.io/index.html

Vijayendra Walvekar, P. (2019). *Virtualizing LoRa baseband functionalities to the Edge.*

GitHub/gr-lora. (2021, 01 17). Retrieved from https://github.com/rpp0/gr-lora

Docker documentation. (2021, 01 17). Retrieved from https://docs.docker.com/get-started/

ZMQ documentation. (2021, 02 01). Retrieved from https://zeromq.org/get-started/

Telegraf documentation. (2021, 01 17). Retrieved from https://docs.influxdata.com/telegraf/v1.17/

InfluxDB documentation. (2020, 02 10). Retrieved from https://docs.influxdata.com/influxdb/v2.0/get-started/

Grafana documentation. (2020, 02 06). Retrieved from https://grafana.com/docs/grafana/latest/getting-started/

5G-DIVE Innovations Specification. (2020, 06 10). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/01/D2.1-5G-DIVE-innovations-specification_v1.0_compressed.pdf

Contiki-NG webpage. (2021, 02). Retrieved from https://github.com/contiki-ng/contiki-ng

Bloessl, B. (2021). *GNURadio IEEE 802.15.4 Source Code*. Retrieved from https://github.com/bastibl/gr-ieee802-15-4

Ettus Reasearch. (2021). Retrieved from https://www.ettus.com/

Zoleria Firefly. (2021). Retrieved from https://zolertia.io/product/firefly/

D1.1: 5G-DIVE architecture and detailed analysis of vertical use cases. (2020). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/01/D1.1_Final.pdf

D3.1: Definition and setup of vertical trial sites. (2020). Retrieved from https://5g-dive.eu/wp-content/uploads/2020/05/D3.1.pdf

Big Data IoT WebPage. (n.d.). Retrieved from https://5g-dive.bigdataiot.com.tw/

D2.1: 5G-DIVE Innovations Specification. (2020). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/01/D2.1-5G-DIVE-innovations-specification_v1.0_compressed.pdf

Mongo Database. (n.d.). Retrieved 9 23, 2020, from https://www.mongodb.com/

Redis Database. (n.d.). Retrieved 9 23, 2020, from https://redis.io/

EasyDarwin. (n.d.). Retrieved 9 23, 2020, from https://github.com/EasyDarwin/EasyDarwin

NetPipe. (n.d.). Retrieved 12 1, 2020, from https://www.ameslab.gov/scl/netpipe.

DroneKit. (n.d.). Retrieved 12 3, 2020, from https://dronekit-python.readthedocs.io/en/latest/develop/sitl_setup.html

Zenoh. (n.d.). Retrieved 12 2, 2020, from http://zenoh.io/

Microelectronics and Information System Research (MIRC-NCTU). (n.d.). Retrieved 12 5, 2020, from https://eic2.nctu.edu.tw/

Ardiansyah, Muhammad Febrian and William, Timothy and Abdullaziz, Osamah Ibrahiem and Wang, Li-Chun and Tien, Po-Lung and Yuang, Maria C. (2020). EagleEYE: Aerial Edge-enabled Disaster Relief Response System. *2020 European Conference on Networks and Communications (EuCNC).*

Yuang, M., Tien, P.L., Ruan, W.Z., Lin, T.C., Wen, S.C., Tseng, P.J., Lin, C.C., Chen, C.N., Chen, C.T., Luo, Y.A. and Tsai, M.R. (2020). OPTUNS: Optical intra-data center network architecture and prototype testbed for a 5G edge cloud. *Journal of Optical Communications and Networking, 12*(1), A28--A37.

Redmon, Joseph and Farhadi, Ali. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767.*

Kuznetsova, Alina and Rom, Hassan and Alldrin, Neil and Uijlings, Jasper and Krasin, Ivan and Pont-Tuset, Jordi and Kamali, Shahab and Popov, Stefan and Malloci, Matteo and Duerig, Tom and others. (2018). The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982.*

Lin, Tsung-Yi and Maire, Michael and Belongie, Serge and Hays, James and Perona, Pietro and Ramanan, Deva and Doll{\'a}r, Piotr and Zitnick, C Lawrence. (2014). Microsoft coco: Common objects in context. *European conference on computer vision.*

5TONIC Lab. (n.d.). Retrieved from https://www.5tonic.org/

5TONIC Lab, Madrid, Spain. (n.d.). Retrieved from https://www.5tonic.org/

DELL PowerEdge R430 Rack Server. (n.d.). Retrieved from https://www.dell.com/dm/business/p/poweredge-r430/pd

Redmon, J. (n.d.). *YOLO: Real-Time Object Detection*. Retrieved from https://pjreddie.com/darknet/yolo/

D2.3: Final Specification of 5G-DIVE Innovations. (2021). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/06/D2.3.pdf

H. Zhao, S. D. (2020). Distributed Redundancy Scheduling for Microservice-based Applications at the Edge. *IEEE Transactions on Services Computing*.

D1.3 . (2021). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/11/D3-DIVE.pdf

D3.2. (2021). Retrieved from https://5g-dive.eu/wp-content/uploads/2021/05/D3.2-final.pdf

Kuma Uptime. (n.d.). Retrieved December 2021, from https://github.com/louislam/uptime-kuma

Nielsen, J. (1993). Chapter 5. In *Usability Engineering.*

InfluxDB. (n.d.). Retrieved December 2021, from https://www.influxdata.com/products/influxdb/

Telegraf plugins. (n.d.). Retrieved December 2021, from https://docs.influxdata.com/influxdb/v2.0/reference/telegraf-plugins/

statsmodel. (n.d.). Retrieved December 2021, from https://www.statsmodels.org/stable/index.html

H2o.ai. (n.d.). Retrieved December 2021, from https://www.h2o.ai/

Mohammed M. Mabkhot, A. M.-A. (2018). Requirements of the Smart Factory System: A Survey and Perspective . *Machines*.

Raspberry Pi HAT. (n.d.). Retrieved from https://www.waveshare.com/sim8200ea-m2-5g-hat.htm

RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. (n.d.). Retrieved from https://datatracker.ietf.org/doc/html/rfc6550

OMA Lightweight M2M server and client in Java. (n.d.). Retrieved from https://www.eclipse.org/leshan/

Kaggle. (n.d.). Retrieved from https://www.kaggle.com/uciml/zoo-animal-classification

5G trial in Milano. (n.d.). Retrieved from https://vodafone5g.it/en/sperimentazione-milano.php

Python libarary Pyro4. (n.d.). Retrieved from https://pyro4.readthedocs.io/en/stable

Mahimahi. (n.d.). Retrieved from http://mahimahi.mit.edu

Psutil cros-platform python library . (n.d.). Retrieved from https://pypi.org/project/psutil/

OpenWRT. (n.d.). Retrieved from https://openwrt.org/start

OpenWRT. (n.d.). Retrieved from https://openwrt.org/start

5Gdataset. (n.d.). Retrieved from https://github.com/uccmisl/5Gdataset