H2020 5G Dive Project
Grant No. 859881

5G-DIVE Innovations Specification

Abstract

This deliverable D2.1 provides a baseline specification of the 5G-DIVE solution. An overall framework governing the solution specification is presented first. This is then followed with detailed specification for each of the targeted vertical pilots, namely industry 4.0 and autonomous drones. Early evaluation results are also presented based on initial implementations reported in the deliverable D2.2.

Document properties

Number	D2.1
Title	5G-DIVE Innovations Specification
Responsible	Alain Mourad (IDCC)
Editors	Alain Mourad (IDCC), Milan Groshev (UC3M), Samer Talat (ITRI)
Contributors	IDCC: Alain Mourad, Ibrahim Hemadeh, Jani-Pekka Kainulainen UC3M: Milan Groshev, Carlos Guimarães ITRI: Samer Talat, Andee Lin, Chenhao Chiu EAB: Chenguang Lu, Gyanesh Patra TELCA: Sergio Fernandez Rubio, Ángel Segovia Fernández, Aitor Zabala Orive ADLINK : Gabriele Baldoni, Luca Cominardi, Ivan Paez NCTU: Osamah Ibrahiem, Muhammad Febrian Ardiansyah, Timothy William, Abebe Belay TID: Luis M. Contreras, Alberto Solano, Jesús Folgueira III: Zora Wang ASKEY: KJ Liu, June Liu RISE: Saptarshi Hazra AAU: Hergys Rexha, Sebastien Lafond ULUND: Chao Zhang, Per Ödling
Reviewers	Antonio De La Oliva (UC3M), Luca Cominardi (ADLINK), Bengt Ahlgren (RISE)
Target dissemination level	Public
Status of the document	Final
Version	1.0
Publication date	30 September 2020

Disclaimer

This document has been produced in the context of the 5G-DIVE Project. The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement Nº H2020-859881. All information in this document is provided “as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors’ view.

Contents

List of Tables

List of Figures

List of Acronyms

3GPP	Third Generation Partnership Project
5GC	5G Core
ASIC	Application Specific Integrated Circuit
ADS	Autonomous Drones Scout
AF	Application Function
AI	Artificial Intelligence
AMF	Access Management and Mobility Function
API	Application Programming Interface
BASS	Business Automation Support Stratum
BBox	Bounding Box
BBU	Baseband Unit
BSS	Business Support System
CA	Carrier Aggregation
CPE	Customer Premise Equipment
C-RAN	Cloud RAN
CU	Central Unit
CNN	Convolutional Neural Network
COCO	Common Objects in Context
COTS	Common Off The Shelf
CPU	Central processing Unit
CSS	Chirp Spread Spectrum
DASS	Data Analytics Support Stratum
DC	Dual Connectivity
DCAS	Drone Collision Avoidance System
DCN	Data Centre Network
DEEP	5G-DIVE Elastic Edge Platform
DL	Downlink
DLT	Distributed Ledger Technology
DNN	Deep Neural Network
DNS	Domain Name System
DPU	Deep neural network Processing Unit
DRL	Deep Reinforcement Learning
DTMC	Discrete Time Markov Chains
DU	Distributed Unit
eMBB	Enhanced Mobile Broadband
EagleEYE	End-to-end Object Detection application
EN-DC	Evolved-Universal Terrestrial Radio Access-New Radio Dual Connectivity
EPC	Evolved Packet Core
EFS	Edge and Fog System
FDD	Frequency Division Duplex
FIFO	First In First Out

FPGA	Field Programmable Gate Array
FQDN	Fully Qualified Domain Name
GPS	Global Positioning System
GPU	Graphics Processing Unit
GTP-U	GPRS Tunnelling Protocol User Plane
HW	Hardware
KPI	Key Performance Indicators
IEEE	Institute of Electrical and Electronics Engineers
IESS	Intelligence Engine Support Stratum
IETF	Internet Engineering Task Force
IL	Imitation Learning
iMEC	Intelligent Mobile Edge Computing
IoT	Internet of Thigs
KVM	Kernel virtual machine
LCM	Life Cycle Management
LTE	Long Term Evolution
LTE-NR	Long Term Evolution-New Radio
LoRa	Long Range
LPWAN	Low Power Wide Area Network
LSTM	Long-Short Term memory
LWM2M	Lightweight Machine-to-Machine
LXC	LinuX Containers
MAC	Medium Access Control
MEC	Mobile Edge Computing
ML	Machine Learning
MIMO	Multiple Inputs Multiple Outputs
mMTC	Massive Machine Type Communications
MOD	Merged Object Detection
MQTT	Message Queuing Telemetry Transport
NBI	North Bound Interface
NEF	Network Exposure Function
NF	Network Function
NFV	Network Function Virtualization
NR	3GPP 5G New Radio
NSA	Non-standalone 5G mode
OCS	Orchestration and Control System
OPTUNS	Optical Tunnel Network System
OSS	Operation Support System
PC	Personal Computer
PiH	Person in need for Help
PHY	Physical layer
QoE	Quality of Experience
RAN	Radio Access Network

RAM	Random Access Memory
RBS	Radio Base Station
RDS	Radio Dot System
RESTful API	Web service implemented using HTTP and the principles of REST
RF	Radio Frequency
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTT	Round Trip Time
RRH	Remote Radio Heads
RU	Radio/Remote Unit
S1	External interface between the LTE eNodeB and the LTE S-GW
SA	Standalone 5G mode
SBA	Service Based Architecture
SBI	Service Based Interface
SDN	Software Defined Network
SDR	Software Defined Radio
SLA	Service Level Agreement
SMF	Session Management Function
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TMs	Trained Models
TSC	Transport Slice Control
SSD	Solid State Drive
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UPF	User Plane Function
URLLC	Ultra-Reliable Low Latency Communication
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VIM	Virtualization Infrastructure Manager
vRAN	Virtualized RAN
vSGW	virtualized Serving Gateway
VSD	Vertical Service Descriptor
VSS	Vertical Support System
X2	External interface defined between two-neighbour eNodeBs
YOLO	You Only Look Once
ZDM	Zero-Defect Manufacturing

Executive Summary

This deliverable provides a baseline specification of the 5G-DIVE solution for the targeted use cases in the Industry 4.0 (I4.0) and Autonomous Drones Scout (ADS) vertical pilots. The main achievements of this deliverable include:

1) Developed the overall design framework governing the 5G-DIVE solution including the underlying 5G connectivity and edge computing infrastructure strata, and the DEEP platform with its three supports systems namely BASS (Business Automation Support System), DASS (Data Analytics Support System), IESS (Intelligence Engine Support System).
2) Applied the design framework to the targeted I4.0 use cases, namely i) digital twinning, ii) zero defect manufacturing (ZDM), and iii) massive machine type communications (mMTC), resulting into the design customization for each use case. Each customized design is applied in the context of specific intelligence engines such as predictive maintenance in digital twinning, object defect detection in ZDM, and radio security for massive MTC.
3) Applied the design framework to the targeted ADS use cases, namely i) drones fleet navigation and ii) intelligent image processing for Drones, resulting into the design customization for each use case. Each customized design is applied in the context of specific intelligence engines such as geolocation and image analytics and object detection.
4) Where available, reported preliminary results from the ongoing implementations of the specifications, such as for the 5G RAN connectivity, Zenoh-based DASS, and intelligence engine for radio security in massive MTC.

The specification in this deliverable already served as a basis for first implementations reported in the deliverable D2.2. It is noteworthy that not all specifications in this deliverable are or will be implemented. The implementation plan is done jointly between WP2 and WP3.

1. Introduction

5G-DIVE aims at designing and validating through experimental trials 5G-based solutions augmented with business automation, data analytics and intelligence deployed at the edge, for two vertical pilots, namely Industry 4.0 (I4.0) and Autonomous Drones Scout (ADS). The targeted use cases in I4.0 and ADS together with their requirements have been defined in a previous deliverable D1.1 [12]. These requirements have been accounted for in D1.1 by defining a baseline architecture with focus on the three support strata targeted in 5G-DIVE, namely Business Automation Support Stratum (BASS), Data Analytics Support Stratum (DASS), and Intelligence Engine Support Stratum (IESS).

This deliverable D2.1 builds on the work from D1.1 and targets the baseline specification of the 5G-DIVE solution in the I4.0 and ADS use cases. This specification has already served as a basis for first implementations reported in an accompanying software deliverable D2.2 and will be further refined in line with the evolving implementations and corresponding experimental results. The structure of this deliverable includes three main sections with a first section (Section 2) describing the overall solution design framework followed by two sections (Section 3 and Section 4) detailing the solution specification for each of the use cases targeted in I4.0 and ADS verticals, respectively.

Section 2 presents the overall design framework for the solution targeted in 5G-DIVE. It first presents an overview of the solution concept and moves next to describe the different layers the 5G-DIVE solution is building upon. These include: 1) the 5G connectivity stratum with its RAN, Transport and Core; 2) the Edge infrastructure stratum including edge and fog computing nodes in line with H2020 5G-CORAL project; and 3) the DEEP stratum introduced by 5G-DIVE including the three support systems targeted namely DASS, BASS, and IESS. Section 2 serves as a common reference design framework for the bespoke solutions detailed in Section 3 and Section 4 for the Industry 4.0 and ADS use cases, respectively.

Section 3 takes the design framework from Section 2 and applies it to the I4.0 use cases, namely i) digital twinning, ii) zero-defect manufacturing, and iii) massive MTC. For each use case, Section 3 delves into the detailed specification starting with a mapping to the 5G-DIVE architecture from D1.1 and moving next to present the detailed specification of key components. Where available, some preliminary experimental results are also presented.

Like Section 3, Section 4 comes next to apply the same design framework from Section 2 to the ADS use cases, namely i) drones fleet navigation and ii) intelligent image processing for Drones. These two use cases are addressed in a scenario of public safety where drones are commissioned to scout a disaster area utilizing intelligently suitable communication (5G) and computing means.

The conclusions from this baseline specification and the next steps are outlined in a final section 5.

Appendixes are also provided at the end of this deliverable covering a range of topics including:

- Section 6: Appendix – ETSI MEC Reference Architecture
- Section 7: Appendix – Zenoh-based Data Management
- Section 8: Appendix – Survey of Relevant Intelligence Techniques
- Section 9: Appendix – Survey of AutoAI/ML Tools

The above appendixes provide the results of surveys and complementary information deemed of value for the reader to read in conjunction with the innovation specifications in the main body of this deliverable.

2. Solution design framework

Figure 2-1 illustrates the overall framework of the solution targeted in 5G-DIVE. The framework consists of three logical layers, namely the network layer, computing layer, and intelligence layer.

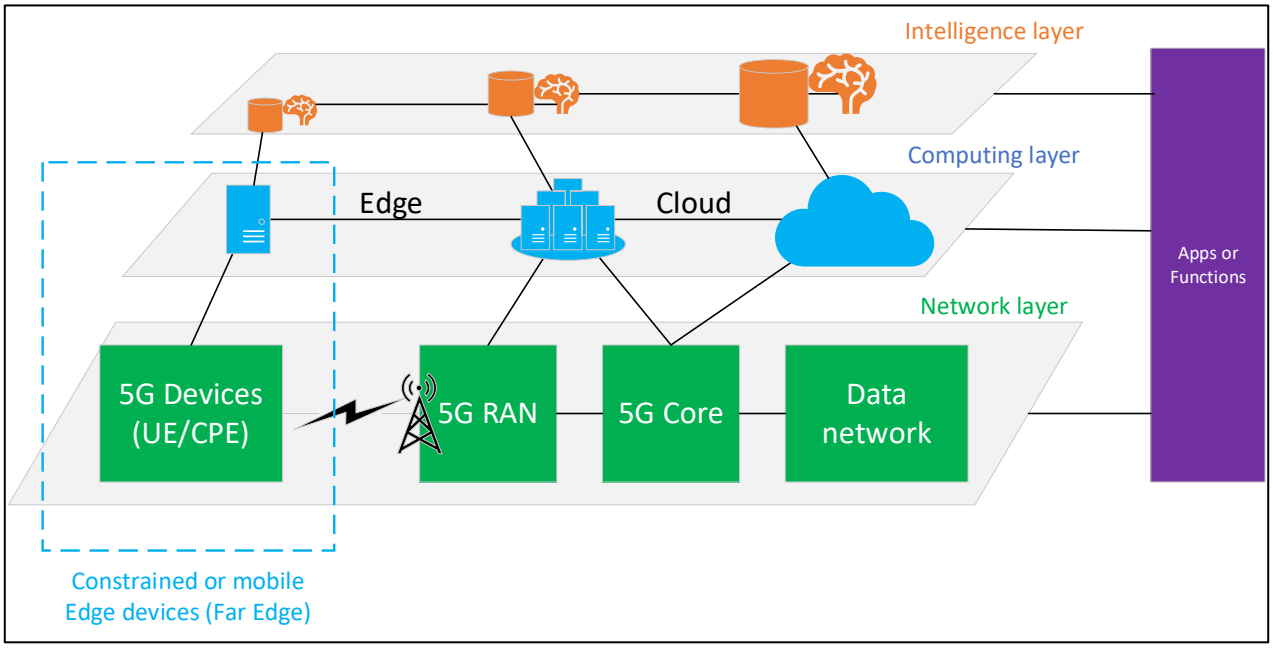


FIGURE 2-1: OVERALL SOLUTION DESIGN FRAMEWORK.

The network layer is depicted using an end-to-end 5G network. 5G devices (e.g. User Equipment (UE) or Customer Premise Equipment (CPE)) connect via 5G NR (New Radio) connectivity to the 5G RAN (Radio Access Network). The 5G RAN includes all gNBs and the CUs (Central Units), DUs (Distributed Units), RUs (Radio Units) where a gNB functional split is considered. The fronthaul network connecting RUs to DUs and CUs is therefore included in the 5G RAN. The backend of the 5G RAN including CUs and gNBs is connected through a backhaul transport network to the 5G Core. The 5G Core includes both options of i) a 4G EPC (Evolved Packet Core) in the non-standalone (NSA) mode, and ii) a next generation core in the standalone mode (SA). The 5G Core represents the egress of the 5G connection towards the external data network.

The computing layer is composed of different computing tiers, namely, the central cloud, the edge cloud (e.g. Telco Edge) connecting to the RAN or Core and far edge (also called as Fog) capabilities attached to the constrained devices (e.g. UEs or CPEs). Far edge capabilities may be embedded in the constrained terminal devices or provisioned externally. Constrained devices may be battery-powered, mobile, or volatile, as compared to the traditional edge clouds connecting to the RAN or Core. The constrained devices can provide federated services involving more than one constrained device.

The intelligence layer includes telemetry, training and inference functionalities which are envisioned distributed across the different computing tiers. Applications and functions may be hosted anywhere

in the computing stratum (cloud, edge cloud or far edge devices). These may include functions or applications residing inside the end-to-end 5G network or external to/over the top of the 5G network.

In the following, we first present the 5G connectivity framework including the main three domains of Core, Transport and Access. Next, the underlying edge infrastructure is presented taking reference in the 5G-CORAL edge and fog computing infrastructure augmented with an enhanced edge data centre. The 5G-DIVE DEEP stratum follows next with its three support systems targeted namely DASS, BASS, and IESS, and the framework of their use by the intelligence engines targeted. A preliminary study on the potential application of Distributed Ledger Technologies (DLT) into DEEP is also presented.

2.1. 5G Connectivity

5G networks are a key piece for supporting the digital transformation of the society. New foreseen services, not only residential ones but also enterprise services represented by vertical industry use cases, can demand extreme requirements in terms of data rate access or low latency, as well as a large number of active sessions originated by permanently connected devices. Thus, 5G services can be generally grouped in three main categories, namely enhanced Mobile Broadband (eMBB), ultra-Reliable and Low Latency Communications (uRLLC), and massive Machine Type Communications (mMTC). Figure 2-2 [1] graphically represents the heterogeneous requirements of these services.

Those characteristics enable the simultaneous support of multiple types of services in different areas: machine communications (e.g., IoT, industrial applications, etc.), new value added services (e.g., telemedicine), mobile broadband at high throughput, advanced vehicular technologies (e.g., connected car, drones), etc.

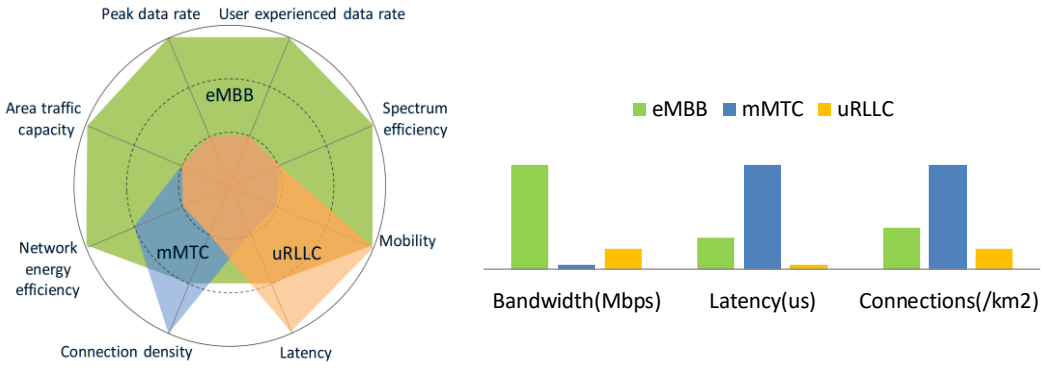


FIGURE 2-2: ITU-R CATEGORIZATION OF 5G SERVICES.

This section focuses on characterizing 5G-DIVE underlying 5G connectivity including the three key parts of core, transport, and access.

2.1.1. 5G Core Network

2.1.1.1. Service-Based Architecture

The 5G Core Network (5GC) has been specified in 3GPP with the aim to increase operational efficiency and support various new advanced services for industries and consumers. 5GC embraced service-based architecture (SBA) using web-scale internet protocols like HTTPv2, cloud-native implementations and deployments, and network automation for service provisioning and service assurance. Figure 2-3 depicts the SBA-based 5GC architecture [2].

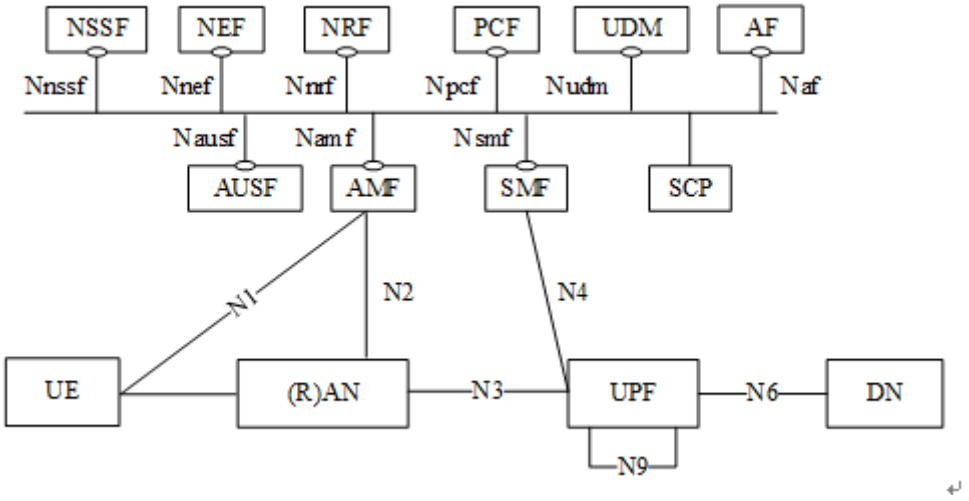


FIGURE 2-3: 5G CORE SERVICE BASED ARCHITECTURE [2].

SBA provides a modular framework enabling the composition of 5GC from different vendors. The control plane and common data repository are delivered by way of a set of interconnected Network Functions (NFs), each with authorization to access each other's services. All the 5GC NFs register themselves and subscribe to services from other NFs, or from Application Functions (AFs) via the Service Based Interface (SBI). Figure 2-4 depicts the 5GC service exposure scheme where NFs expose through NEF (Network Exposure Function) their services to AFs and vice-versa.

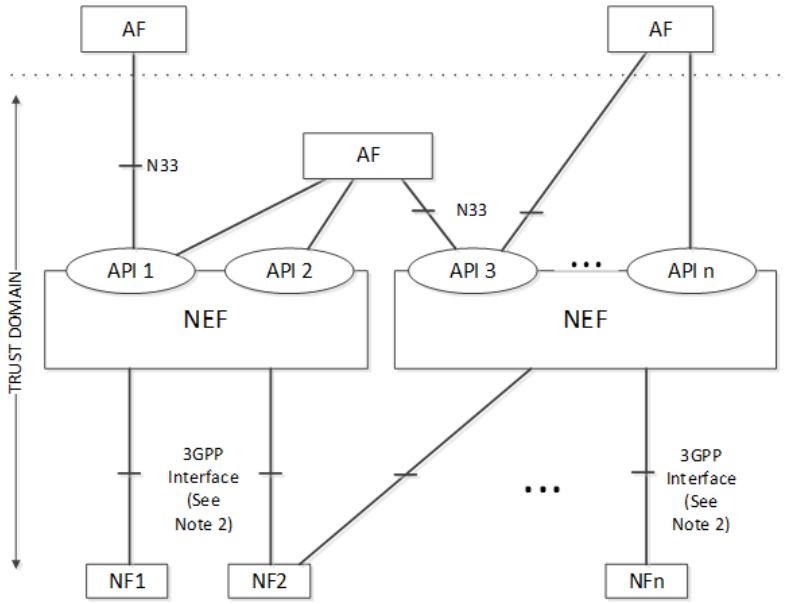


FIGURE 2-4: NETWORK SERVICE EXPOSURE SCHEME [2].

Today's 5GC SBA is evolving in the direction of finer decomposition of the NFs into smaller sub-functions providing micro services to one another, where some micro services may even be reused for different NFs and have independent life-cycle management from one another.

2.1.1.2. Network Slicing

Network slicing provides an advance in service offerings, adding new service characteristics on top of existing service offerings, especially for industry verticals. Aspects such as guaranteed latency, high throughput, isolation, etc., are made available for the verticals consuming the network, permitting sophisticated services to gracefully co-exist on top of the same infrastructure.

Network slicing allows multiple virtual networks to be created on top of a common shared physical infrastructure where each slice is tailored to a given service profile such as eMBB, URLLC or mMTC. Each slice may therefore have its own network capabilities and characteristics like protocols, quality of service and security settings for defined business purposes of a customer.

5GC manages the end-to-end slices configuration and deployment in accordance with services' traffic characteristics and service level agreements (SLA). In particular, the access and mobility management function (AMF) establishes the UE context and PDU resource allocation via slice assistance information (S-NSSAI) provided by the UE. The S-NSSAI is set up per PDU session for the policy management in the PDU session level. This is depicted in Figure 2-5. The session management function (SMF) controls the user plane function (UPF) and therefore directs and redirects the service flows as required for the applications.

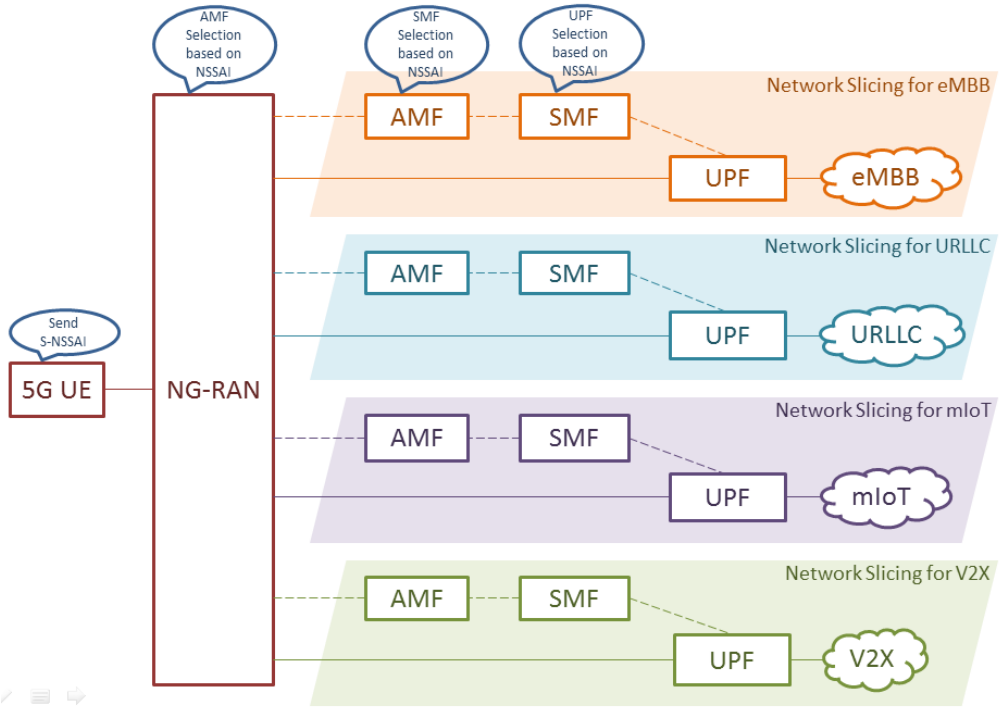


FIGURE 2-5: NETWORK SLICE IN 5G CORE.

2.1.2. 5G Transport Network

5G transport networks accommodate simultaneously a mix of different logical service typologies with very distinct needs on top of the same physical infrastructure. The deployment of those services, although expected to be gradual, stress the capabilities of existing backhaul and aggregation networks, requiring the evolution of the transport networks in terms of capacity, data processing capabilities and even some relevant architectural changes, especially for guaranteeing very low latencies. All these evolutionary aspects in the transport part are accompanied by some other additional challenges emerging in parallel and impacting the underlying transport segments, linked to two new technological paradigms: network virtualization and network programmability.

On the one hand, virtualization, which is being adopted by the radio access solutions, presents diverse architectural options of protocol stack disaggregation, allowing the centralization of some of the radio processing functions. The original proposition of radio disaggregation was fueled by the Cloud RAN (C-RAN) architecture [3], promoting the disaggregation of the radio stack, which traditionally has been deployed in a monolithic manner within a radio base station. Such disaggregation proposes to divide the radio stack into parts that can be hosted and distributed along the access and aggregation segments of the network, leveraging on cloud computing capabilities. In this way, what was previously shipped as a monolithic functionality of a Radio Base Station (RBS), is now decomposed as a Radio Unit (RU) comprising the radio antenna head, a Distributed Unit (DU) that performs the real time processing of the radio stack, and the Centralized Unit (CU) which concentrates the higher radio layer stack performing the non-real time processing tasks. Figure 2-6 illustrates these three components. Such

architecture leverages on virtualization of the architectural components leading to what is called the virtual RAN (vRAN) approach.

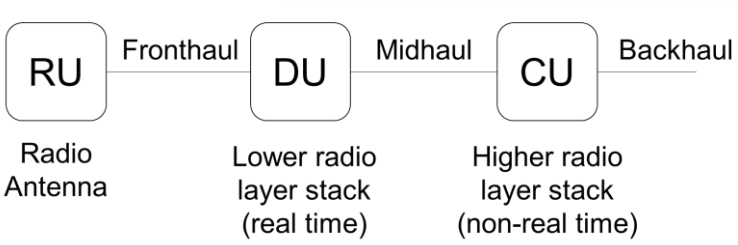


FIGURE 2-6: RADIO FUNCTIONAL SPLIT COMPONENTS.

The programmability of the network facilitates an easier reconfiguration of the transport substrate, making possible the automation of operations in the network and its integration with intelligent mechanisms of management and control, including autonomic AI-driven network automation. The interplay of virtualization and programmability permits to go a step further enabling the partition and allocation of virtual resources for a specific service purpose, in an isolated manner, facilitating to accommodate the available resources for different needs along the time, in a flexible and automated manner. This concept is also referred as network slicing. These paradigms, if implemented properly, have the potential of enabling the generation of important economic efficiencies in terms of CapEx and OpEx for service providers, as well as enabling new service offerings, especially for vertical industries, as mentioned before.

2.1.2.1. Transport Network Scenarios and Requirements

The radio software stack, once disaggregated, can be deployed either on purpose-built devices or on x86 COTS servers. Certainly, the latter can provide more flexibility in the instantiation and update of functionalities and features in the aggregation network, so getting traction in the industry as a more efficient mean of network roll-out.

The lower part of the radio protocol stack, focused on real-time processing tasks, is deployed in the proximity of the radio antennas reducing the energy consumption and size. Complementary to that, the upper part, dedicated to non-real time processing, is concentrated and located in a central point of the network, providing a more efficient utilization of the baseband processing resources.

Four different scenarios can be foreseen depending on the separation of RU, DU and CU:

1. Full integration of RU, DU and CU: this case corresponds to the conventional backhaul.
2. RU and DU integrated: the RU is deployed in proximity to the DU and no transport equipment is needed among them. In this case, there are midhaul (between DU and CU) and backhaul networks (from the CU).
3. DU co-located with CU: the RU is distant from the DU/CU, existing fronthaul transport infrastructure, and also backhaul from the CU.

4. Full separation of RU, DU and CU: in this scenario, there are fronthaul, midhaul and backhaul network segments.

The distinct alternatives of functional split of the radio stack impose different requirements on the transport side. The lower the split, the higher is the throughput required, depending on factors such as the number of antennas, MIMO layers, etc. In terms of latency (and jitter), the more stringent requirements also correspond to low layer splits, which can be as low as tens to several hundreds of microseconds. Table 2-1 from [4] summarizes some of these requirements[1].

TABLE 2-1: TRANSPORT NETWORK REQUIREMENTS DEPENDING ON THE FUNCTIONAL SPLIT.

Protocol Split option	Required bandwidth	Max. allowed one way latency [ms]	Comment
Option 1	[*DL: 4Gb/s*] [*UL: 3Gb/s*]	[*10ms*]	
Option 2	[*DL: 4016Mb/s*] [*UL:3024 Mb/s*]	[*1.5~10ms*]	[16Mbps for DL and 24Mbps for UL is assumed as signalling
Option 3	[*lower than option 2 for UL/DL*]	[*1.5~10ms*]	
Option 4	[*DL:4000Mb/s*] [*UL:3000Mb/s*]	[*approximate 100us*]	
Option 5	[*DL: 4000Mb/s*] [*UL: 3000 Mb/s*]	[*hundreds of microseconds*]	
Option 6	[*DL: 4133Mb/s*] [*UL:5640 Mb/s*]	[*250us*]	[133Mbps for DL is assumed as scheduling/ control signalling. 2640Mbps for UL is assumed as UL-PHY response to schedule]
Option 7a	[*DL:10.1~22.2Gb/s*] [*UL:16.6~21.6Gb/s*]	[*250us*]	[713.9Mbps for DL and 120Mbps for UL is assumed

[1] Calculation done according to the following parametrization:
Channel Bandwidth: [100MHz(DL/UL)]
Modulation: [256QAM(DL/UL)]
Number of MIMO layer: [8(DL/UL)]
IQ bitwidth: [2*(7~16)bit(DL), 2*(10~16)bit(UL)] for Options 7a, 7b and 7c, and [2*16bit(DL/UL)] for Option 8
~~Number of antenna port: [32(DL/UL)] for Options 7b, 7c(UL) and 8~~

			as MAC information]
Option 7b	[*DL:37.8~86.1Gb/s*] [*UL:53.8~86.1 Gb/s*]	[*250us*]	[121Mbps for DL and 80Mbps for UL is assumed as MAC information]
Option 7c	[*DL:10.1~22.2Gb/s*] [*UL:53.8~86.1Gb/s*]	[*250us*]	
Option 8	[*DL:157.3Gb/s*] [*UL: 157.3Gb/s*]	[*250us*]	

The virtualization trend also applies to the mobile packet core entities. Through virtualization, entities of the mobile packet core can be instantiated in different locations (e.g., edge nodes, central cloud, etc) allowing to adapt the deployment for each of the aforementioned services. For instance, some mMTC services could not require the support of mobility management at all, allowing for a very extreme centralization of the deployment. Other kind of services, such as uRLLC services, could require very extreme latency reduction pushing for a very capillary distribution of the core entities. This versatility implies the necessity of establishing in an easy way multiple paths in the network simultaneously, with different transport needs associated (e.g., need for traffic engineering) to them and for several different locations, pushing for the need of programmability. If we additionally consider the potential coexistence of different access technologies, like 3G and 4G together with 5G, then the existing transport network should be able to carry multiple and different logical interfaces for the mobile service, as shown in Figure 2-7.

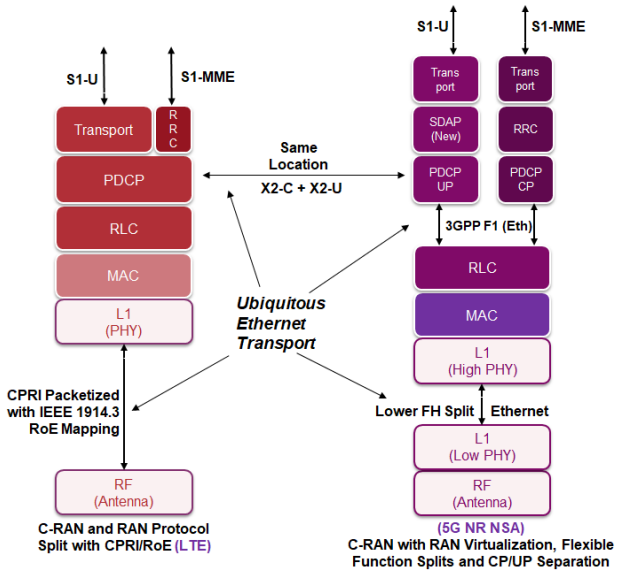


FIGURE 2-7: MULTIPLE LOGICAL MOBILE NETWORK INTERFACES SUPPORT.

2.1.2.2. Transport Network Slicing

A key component of the end-to-end network slicing is precisely the slicing of the transport network which provides the necessary connectivity to the elements and functions constituting the final end-to-end communication service. Thus, the transport network is an essential component in the end-to-end delivery of services and, consequently, it is necessary to define the mechanisms in which the transport network is consumed as a slice.

There is an on-going activity in IETF to define the scope of transport slicing [5]. From an operator perspective, it is important to understand what the attributes to be supported in the northbound interface (NBI) are. In this respect, taking as the starting point the developments in GSMA and 3GPP with respect to the definition of network slices, there is an on-going effort on identifying what of those characteristics impact the transport network [6]. Through such NBI, as shown in Figure 2-8, the TSC (Transport Slice Controller) will be integrated with external entities, such as 3GPP Management Systems, requesting such kind of slices.

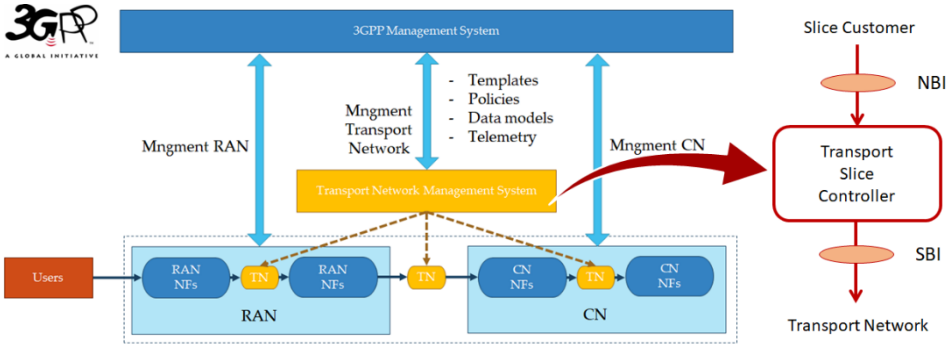


FIGURE 2-8: INTERACTION OF 3GPP MANAGEMENT SYSTEM WITH IETF TRANSPORT SLICE CONTROLLER.

It should be noted that the concept of TSC does not imply the existence of a separate entity at transport level. It is just a functional characterization of the entity that is required to support transport slices. Transport slicing should be integrated in a smooth manner with the existing network programmable capabilities in carrier networks. One potential example is the SDN architecture defined by Telefónica [7]. A possible integration is illustrated in Figure 2-9.

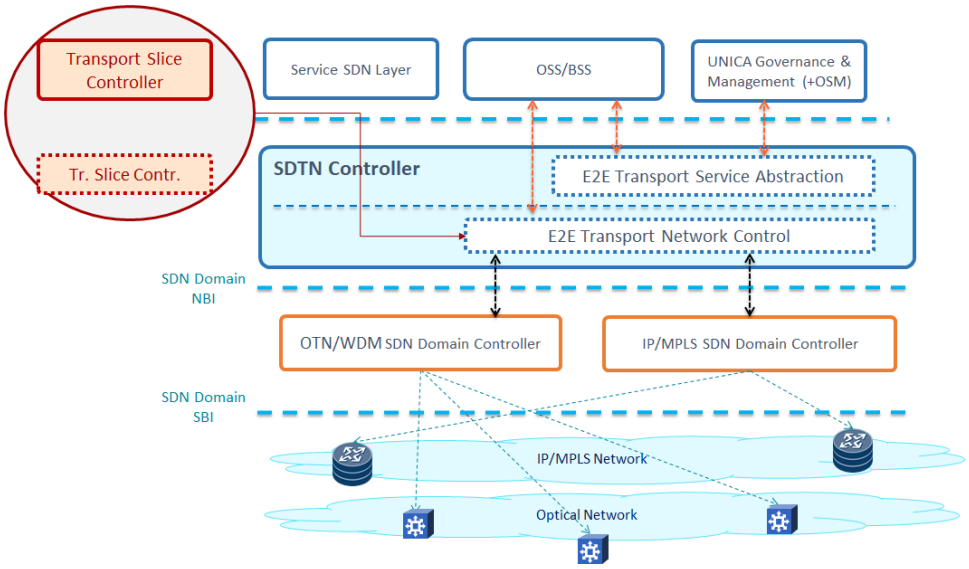


FIGURE 2-9: POTENTIAL POSITIONING OF TRANSPORT SLICE CONTROLLER FUNCTIONALITY WITHIN SDTN ARCHITECTURE.

2.1.3. B5G Radio Access Network

Here, we focus on the RAN part of 5G connectivity. In the following, we will first give an overview regarding 5G RAN (especially on air-interface) key features in section 2.1.3.1. A deployment example for the Industry 4.0 pilot in 5G-DIVE is described in section 2.1.3.2. Then section 2.1.3.3 presents the lab measurement results obtained in 5G-DIVE regarding latency and peak throughput.

2.1.3.1. Overview

The first release of the 3GPP 5G NR radio-access technology [8] was specified in 2018. The spectrum range supported by 5G is extended up to 52.6 GHz from around 3.5 GHz in 4G. New spectrum offers much more bandwidth to support the ever-increasing high traffic volumes in 5G era and enable much higher end-user data rate. Two frequency ranges, i.e. FR1 (450 MHz – 7.125 GHz, referred also as sub-6GHz including both low-band and mid-band) and FR2 (24.25 GHz – 52.6 GHz, referred to as mmWave band), are supported with flexible OFDM numerology. For example, higher subcarrier spacing, e.g. 120 kHz, is used for mmWave, while 15 kHz is used for low-band and 30 kHz is used for mid-band. The maximum carrier bandwidths supported are for subcarrier spacings of 15/30/60/120 kHz are 50/100/200/400 MHz, respectively. Even larger bandwidth can be supported by carrier aggregation.

Massive MIMO is another key 5G feature to improve the system performance with a large number of antenna elements. The size of antenna elements operating on higher frequency bands are smaller following the theory of physics. Using more antennas becomes feasible without increasing too much the size of the overall antenna system. For example, in mid-band, advanced antenna system with 32/64 antenna elements are usually deployed in dense urban areas to boost cell capacity and spectrum efficiency with SU-MIMO and MU-MIMO, while mmWave radios usually have 256 or 512 antenna elements to counter the increased path losses in high band frequencies by forming narrow beams towards UEs. Furthermore, the system design of 5G is beam-centric, in which all channels and signals

including synchronization and control channels are designed to support beamforming. UEs are required to support more MIMO layers, e.g. supporting 4 layers in DL and 2 layers in UL, which further increases the data rate.

5G also features with low latency, which is especially important to some new vertical use cases such as factory automation. First, thanks to the flexible numerology, higher subcarrier spacing results in shorter slot duration and therefore reduce the latency. Second, to further reduce latency, several new MAC (medium access control) and PHY (physical layer) features are specified in 5G, enabling new capabilities such as faster scheduling, smaller transmissions (mini slots), repetitions, faster retransmissions, pre-emption and packet duplication. For use cases requiring high reliability, these new features also provide the tools to manage the trade-off between latency, reliability and capacity. High reliability of a packet can be achieved by using very low code rate, achieving very low packet error rate with very low data rate. However, if the latency requirement allows for a few retransmissions, less robust code rate can be used to achieve higher data rate. In addition, multi-antenna, carrier aggregation (CA) and dual connectivity (DC) can be used to improve the transmission reliability.

Ultra-lean air-interface design in 5G has significantly reduced always-on transmissions, such as reference signal transmissions, system information broadcast etc. For example, cell reference signal (CRS) which is transmitted all the time in 4G is removed in 5G, while the demodulation reference signals (DMRS) are only transmitted when there is a data transmission. Several procedures, such as cell search and random access, have also been redesigned to adapt for this change. Furthermore, minimizing the always-on transmissions creates more opportunities for base stations to save energy, paving the way to a more energy-efficient network to achieve energy consumption scaled with traffic demand dynamically.

Given these key features of 5G RAN briefly described above, 5G radio technologies bring significant improvements against 4G, in the areas of spectrum efficiency, system capacity, lower latency, higher reliability, energy consumption, scalability and flexibility. There are other advancement areas of 5G, such as dynamic spectrum sharing (DSS), integrated access backhauling (IAB) for mmWave, NR for unlicensed spectrum (NR-U), V2X etc. More details can be referred to in [8][9][10] and the references therein.

2.1.3.2. Deployment example: indoor radio system for Industry 4.0

The use cases of Industry 4.0 take place mostly indoors, like in factory plants, warehouses etc. Indoor radio systems like Ericsson Radio Dot System (RDS) [11] are suitable to such scenarios, which are specially designed to fulfil the requirements of indoor environment.

As an example, we present more details of Ericsson RDS, which is the world's smallest, lightest, and highest performing 5G mid-band small cell indoor radio, making simple and flexible deployment. It will be used for 5G-DIVE Industry 4.0 trials in Taiwan. As shown in Figure 2-10, RDS is comprised of the following 3 parts.

- Radio Dot: it is like an active antenna unit, featuring with a small form-factor, light weight, low power consumption and elegant design, supporting easy installation, e.g. on ceiling and on

walls. Radio Dots are remotely powered over the standard LAN cable with PoE (Power over Ethernet) technology. Radio signal are compressed with advanced algorithms, supporting 800 MHz total bandwidth over one link of 10 Gigabit Ethernet.

- Indoor radio unit (IRU): IRU acts as a radio aggregator, which aggregates multiple Radio Dots as one cell. Thereby, UEs within the coverage of one IRU have seamless mobility between Radio Dots. The seamless mobility feature can be even extended to multiple IRUs.
- Digital unit (DU): DU is also refers as baseband unit (BBU), which performs the baseband processing. DU and IRU can be co-located or located at a separate location. When they are not co-located, they are connected via fiber links. DU runs the same Ericsson baseband software as used for macro base stations, bringing the same high performance and feature support to indoors.

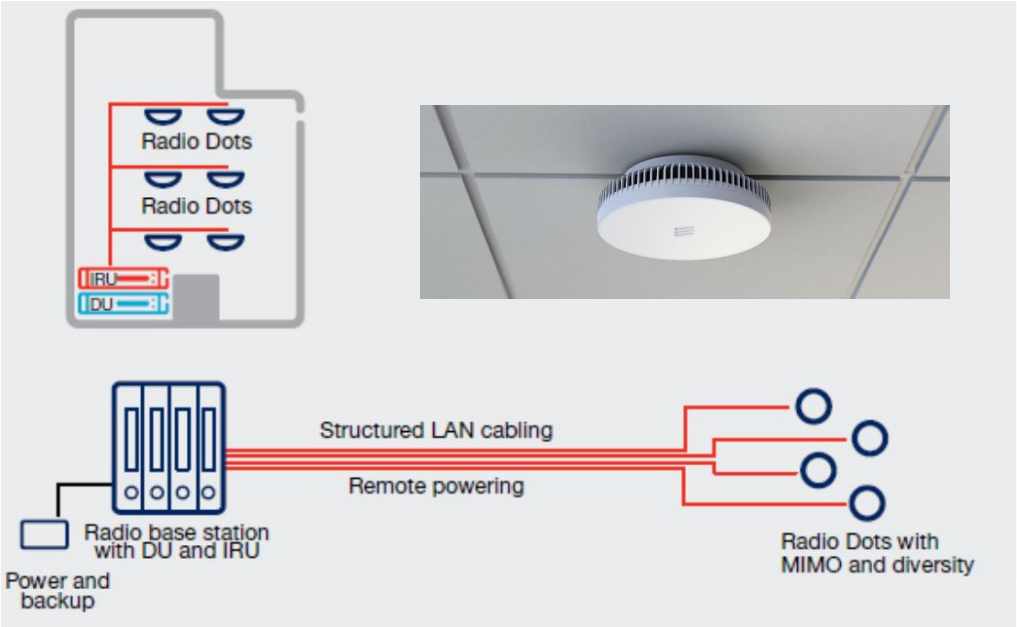


FIGURE 2-10: ILLUSTRATION OF ERICSSON RADIO DOT SYSTEM.

The new sub-6GHz 5G bands (e.g. 3.5 GHz, 4.9 GHz) are higher than the existing 4G bands (e.g. 2.1 GHz, 2.3 GHz). From propagation point of view, the pathloss of higher frequency bands are higher, which can cause smaller coverage. However, in 5G RDS design, the additional pathlosses are well compensated by increased transmit power and increased number of antennas. As shown by the simulation results in Figure 2-11, 5G RDS at 3.5 GHz with 4T4R and 100 MHz bandwidth achieves the same coverage as 4G RDS at 2.3 GHz with 2T2R and 20 MHz bandwidth, where the Radio Dots are deployed at the same locations. With the same coverage, 5G RDS can achieve much higher data rate thanks to a wider carrier bandwidth and more MIMO branches used.

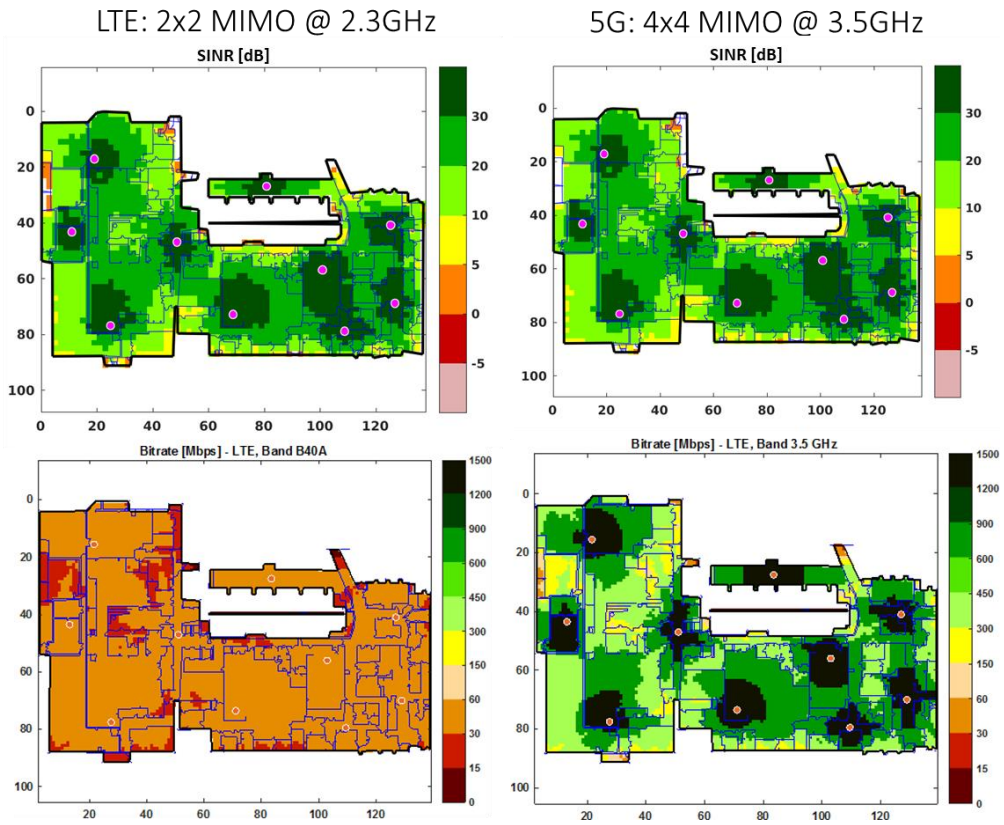


FIGURE 2-11: SIMULATION RESULTS OF A TYPICAL INDOOR SCENARIO.

2.1.3.3. Lab testbed and measurement results

A lab testbed has been built in Ericsson Research lab in Kista, Sweden. The testbed currently supports 5G NSA with 5G NR on n78 band (i.e. 3.5 GHz) and 4G LTE on B3 band (i.e. 1800 MHz). The setup is based on Ericsson RDS consisting of Radio Dots, IRU, BBU (DU) and 5G EPC, as shown in Figure 2-12. Two RDS chains are used to support 5G NR and 4G LTE at the same time for NSA, where user plane goes through 5G NR and control plane is done via 4G LTE. The 5G EPC used in the lab setup is a remote Core deployed at a different location from the RDS and BBU setup with a dedicated fiber link in between. This will be replaced with a local Core co-located with other equipment in later stage for Taiwan field trial. Furthermore, the system is planned to be upgraded to 5GC with support to SA deployment. The UE used is a 5G CPE from Wiston NeWeb Corporation (WNC) supporting NSA. It is based on Qualcomm's Snapdragon S50 5G modem. In the next step, we plan to order new WNC SKM-5X CPEs with a newer S55 5G modem supporting SA and 2x2 MIMO in UL.

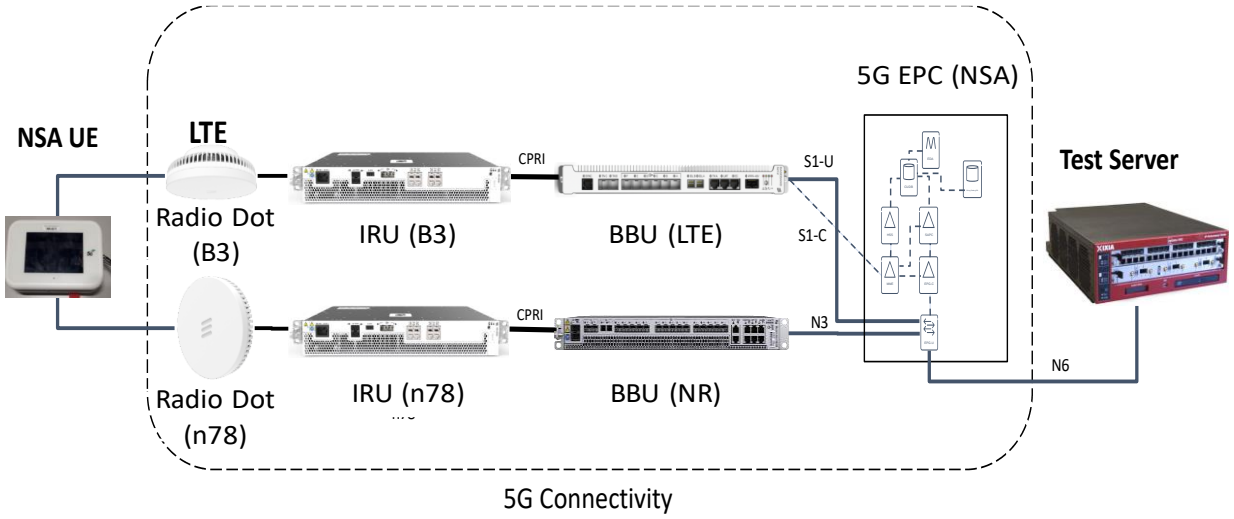


FIGURE 2-12: ILLUSTRATION OF LAB SETUP.

Lab measurements have been performed to compare the performances between 5G NR and 4G LTE. In the measurements, 5G NR is TDD having 100 MHz bandwidth with 4x4 MIMO in downlink and 4x1 MISO in uplink, while LTE is FDD having 20 MHz bandwidth with 2x2 MIMO in both downlink and uplink.

Table 2-2 shows the measured round-trip time (RTT) results. It shows that the latency of 5G NR is much shorter than 4G LTE. For example, the maximum RTT for pinging from server to UE is reduced from 35.7 ms to 15.9 ms by 55%. The main contributor for the RTT reduction is the numerology of 30 KHz subcarrier spacing, which is twice of 15 kHz used in LTE and thus results in a short slot, only a half slot time of 4G.

Table 2-3 shows the measured peak throughputs. In DL, the throughput of 5G NR is more than 10 times better than 4G LTE, mainly due to 5 times increased bandwidth and 2 times more MIMO layers. In UL, the improvement is smaller because TDD DL/UL pattern always allocates more resources to DL and the UE currently used doesn't support 2x2 MIMO. It is expected to achieve doubled peak throughput with the new CPE supporting 2x2 MIMO.

In the experiment, the LTE mode used was 20 MHz FDD. This mode has 20 MHz dedicated for UL transmission. NR mode used was 100 MHz TDD. The TDD pattern is 3:1 for DL to UL ratio. Effectively, it is around 25 MHz bandwidth for UL if other overhead is not considered. The UE used in the experiment only supports SISO (1T4R) and 64-QAM. In these conditions, it is reasonable that LTE and NR UL throughputs are similar in this case, and that is reflected in the table. When carrier aggregation is used between NR and LTE, the peak throughput is around 100 Mbps, although this was not measured. The 5G system is expected to be upgraded to SA at a later point in time. With new UEs supporting 2T4R, 2-layer UL MIMO can also be supported. With 2 layers (and maybe also 256-QAM), the peak throughput is expected to be increased to more than 100 Mbps.

TABLE 2-2: PING RTT RESULTS.

		MIN (ms)	AVG (ms)	MAX (ms)	MEDIAN (ms)
5G NR	*UE -> Server*	7.96	10.85	23.5	9.65
	Server -> UE	7.96	9.67	15.9	9.03
4G LTE	*UE -> Server*	19.4	25.28	36.9	25
	Server -> UE	18.4	24.38	35.7	23.8

TABLE 2-3: PEAK THROUGHPUT RESULTS.

	Uplink (Mbps)	Downlink **(Mbps)**
5G NR	53	1268
4G LTE	46	96

2.2. Edge Infrastructure

The 5G-DIVE solution uses Edge Computing Infrastructure enabling support for an end-to-end Platform-as-a-Service (PaaS) service model. This approach allows verticals to develop, run, and manage applications and services without the complexity of building and maintaining the infrastructure typically associated with the delivery of the application. As presented in 5G-DIVE deliverable D1.1 [12], this is particularly relevant given the heterogeneous and distributed nature of edge and fog environment. It is therefore of paramount importance to provide the necessary tools for a streamlined and flexible management, thus concealing the underlying PaaS complexity.

The Customer Provided Equipment (CPE) used for the Industry 4.0 pilots are commercial ones. For the first year validation at the 5TONIC lab, we used 4G and 5G CPEs because the robotic arm does not have a 4G/5G interface. So, the robotic arm is connected via Ethernet to two CPEs that provide connectivity towards the 4G and 5G networks available at 5TONIC. In particular, for 4G router was the Huawei's B315s-22 [45], while for the 5G we used the 5G CPE PRO Baloong 5000 from Huawei [46].

2.2.1. Reference 5G-CORAL Architecture

5G-DIVE edge infrastructure builds on the H2020 EU-Taiwan 5G-CORAL project. The reference 5G-CORAL architecture specified in [13] is based on ETSI NFV [14] and ETSI MEC [15] frameworks and is composed of two major building blocks, namely the Edge and Fog Computing System (EFS) and the Orchestration and Control System (OCS). This is depicted in Figure 2-13.

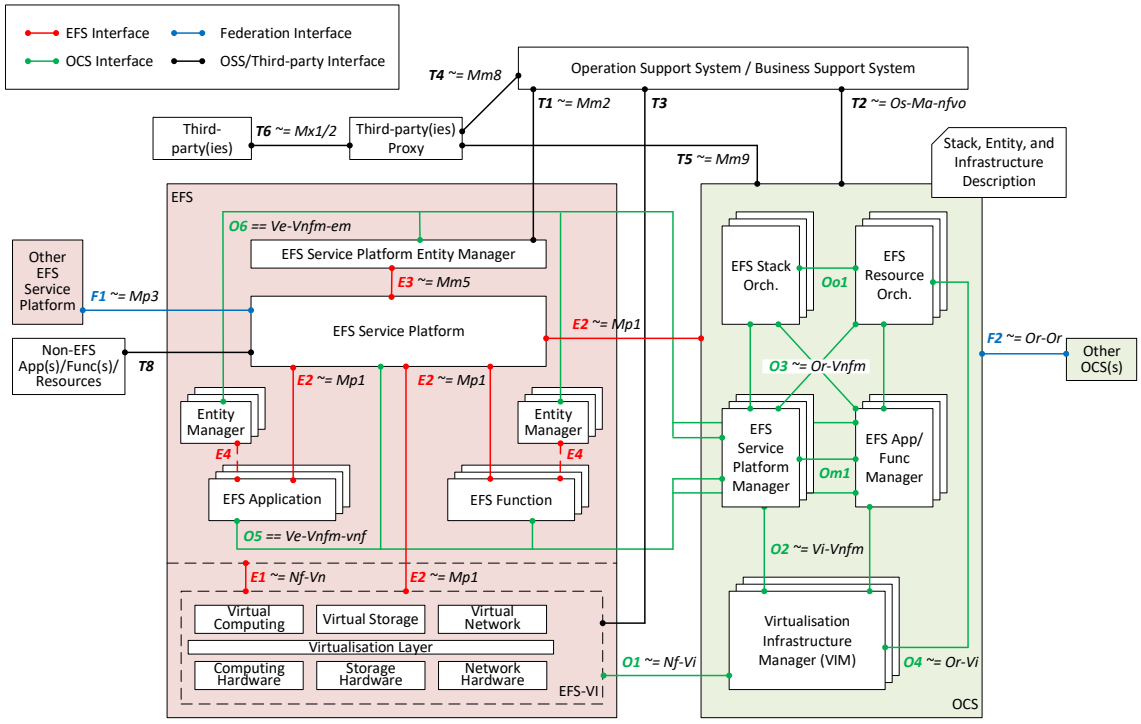


FIGURE 2-13: 5G-CORAL SYSTEM ARCHITECTURE.

2.2.1.1. Edge and Fog Computing System

The EFS is composed of a set of Edge and Fog resources that belong to a single organisation, thus it represents a single administrative domain. Inside an EFS, apart from the aforementioned resources, software components are present including the Service Platform, Functions and Applications. Such components run on top of the EFS resources leveraging virtualisation for decoupling, isolation, and density.

While the overall EFS architecture is compliant with ETSI MEC-in-NFV [15], it introduces a differentiation between user-plane applications and network functions. EFS Applications map to MEC Apps in the ETSI MEC framework, while EFS Functions map to VNF in the ETSI NFV framework. Examples of EFS Apps include web servers and application backend, while examples of EFS Functions include Firewalls, DNS Servers, etc. On the other hand, the EFS Service Platform is analogue to the MEC Platform, thus providing an environment for Applications and Services to discover, communicate and store relevant information.

2.2.1.2. Orchestration and Control System

The OCS provides orchestration, management and control over one or more EFSs. It comprises an EFS Orchestrator, for both applications and resources, an EFS manager for both Service Platform and Application/Functions, and Virtual Infrastructure Managers (VIMs).

Such components are designed to support heterogeneity and dynamicity in the resources and applications. The EFS stack orchestrator provides orchestration, validation and control functionalities for the software components of a given EFS, while the EFS resource orchestrator keeps track of resource

utilisation and allocation for a given EFS. The Service Platform manager provides configuration and management capabilities to a specified EFS Platform, which comprises verification of running App/Func or registration of new services and the platform lifecycle management. The App/Func manager provides the possibility to manage single applications and functions inside a given EFS, including lifecycle management.

The VIM comprises functionalities that are used to manage the virtualisation infrastructure and provide virtual resources to service platform, applications and functions. Multiple VIMs may be deployed to control and manage different virtualisation substrates (e.g. VMs and Containers) as well as different administrative domains.

2.2.2. Enhanced Edge with OPTUNS

5G-DIVE project aims at taking advantage of an enhanced edge infrastructure (OPTUNS) available to the project through the work of NCTU partner. OPTUNS (optical tunnel network system) is an Edge Data Center (EDC) network architecture and prototype testbed designed to deliver massive bandwidth and ultralow latency [16] for 5G edge cloud. OPTUNS consists of a set of optical switching subsystems that operate collectively to interconnect racks of servers/storage systems by means of wavelength-based optical tunnels. These optical tunnels are proactively governed by an SDN-based intelligent tunnel control system resulting in a continuous availability of optical tunnels. OPTUNS boasts five significant features that are crucial to meeting the needs of a 5G edge cloud. They include scalable/modular architecture, massive wavelength reuse (yielding high bandwidth), proactive optical tunnel control (yielding ultralow latency), fault tolerance, and high energy efficiency. These are five features that OPTUNS can provide which are not always prevalent in existing cloud data center network (DCN). Existing cloud DCNs employ the use of electrical spine-leaf data center (ESDLC) to deliver ultralow latency and high bandwidth requirement of an edge cloud. However, experiment results showed that ESDLC is incapable of meeting the aforementioned demands for an edge cloud. Specifically, under load ≥0.85 [17], an ESDLC exhibits overloaded phenomena at a few leaf/spine switch ports, while others remain underloaded.

OPTUNS presents itself as a promising DCN solution to support for the rising needs of compute intensive vertical applications. One crucial feature of OPTUNS is that it is able to provide for a high bandwidth and ultralow latency communication network between rack-to-rack and server-to-server via wavelength reuse in optical tunnels [17]. These features are key in supporting for vertical applications that require high bandwidth and intensive computations. Operations such as video processing and artificial intelligence model training in 5G-DIVE use cases require high bandwidth for video transmission and computational power for training complex models. Furthermore, OPTUNS enables ultralow latency between servers for distributed computing processes across all available computing nodes in the EDC.

2.3. DEEP design framework

This section presents 5G-DIVE DEEP suite of support systems including DASS, BASS, and IESS. It also presents a few intelligence engines that have been developed in the first year of 5G-DIVE. Preliminary considerations to the application of distributed ledger technologies in 5G-DIVE solution are also briefly presented here.

2.3.1. Data Analytics Support System

2.3.1.1. DASS Architecture

The DASS architecture was initially presented in D1.1 [12] is composed of three main building blocks, the data dispatcher, the data pre-processing, and the data storage, as depicted in Figure 2-14.

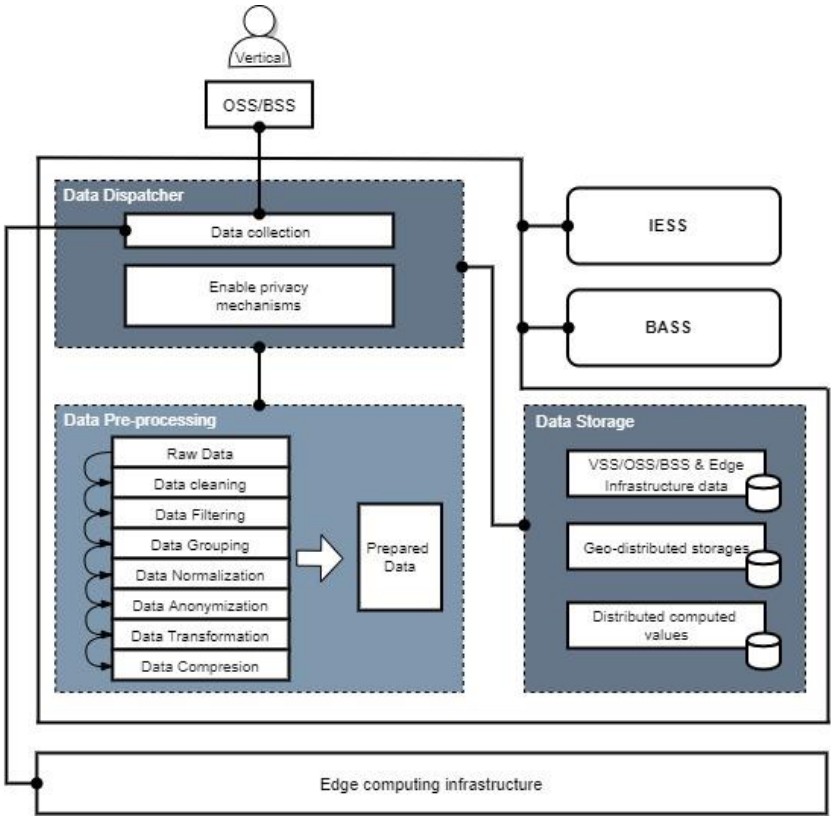


FIGURE 2-14: DASS ARCHITECTURE.

The data dispatcher functionality implements a networking layer capable of running above a Data Link, Network or Transport Layer. This provides primitives for efficient pub/sub and distributed queries. It supports fragmentation and ordered reliable delivery. The data dispatcher has been the primary focus of the work carried out during the first reporting period. The data dispatcher is implemented by the Zenoh.net layer, whose motivation, concept, architecture and design are detailed in Appendix 8.

The data pre-processing and data storage functionalities provide a high-level API for pub/sub and distributed queries, data representation transcoding, an implementation of geo-distributed storage and

distributed computed values. The design and implementation of data pre-processing and the data storage will be addressed in the second reporting period. This is envisioned to be based on Zenoh as described in Appendix 8. Table 2-4 describes DASS functionalities of each building block.

TABLE 2-4: DASS BUILDING BLOCKS.

Proposed WP2 Building Blocks	Functionalities
Data Collection	Subscriber: is usually an entity (e.g., an application) subscribing to a given resource (e.g., */house/room/temp*) that has interest in any new value associated to it. A subscriber can be either **push** or **pull**. A push subscriber receives data as produced. For a pull subscriber, the infrastructure caches the data, as close as possible to allow the subscriber to consume it effectively when ready to pull.
	Queryable: issues a distributed query and returns a stream of results. The query target, coverage and consolidation depend on configured policies.
Enable Privacy Mechanisms	Enables the creation of a properly designed NDN topology, organized hierarchical by a meaningful key space, making it possible to achieve the desired level of privacy for the data.
Data Pre-processing activities	Responsible for preparing the data for both analytics and machine learning purposes. Once the data is collected, it is time to assess the condition of it looking for trends, outliers, exceptions, inconsistent, missing or skewed information. We have identified eight interchangeably steps that can be applied over to different type of variables e.g. continuous, nominal, ordinal, etc. The expected output is to have data that can be split into two sets: one for training the ML algorithms, and another for evaluation purposes.
Data storage	Vertical and Edge infrastructure data: responsible for storing the processed OSS/BSS data as well as the edge computing infrastructure
	Geo-distributed storages: responsible for enabling a storage system that spans many geographic locations while providing location transparent access to it.
	Distributed computed values: A computation registered at a specific path. This computation can be triggered by a get operation on a selector

	matching this path. The computation function will receive the selector's properties as parameter

Table 2-5 provides a mapping between the above DASS building blocks and the 5G-DIVE baseline architecture presented in D1.1 [12]. Only the data dispatcher has been specified and implemented in the first year of the project.

TABLE 2-5: MAPPING OF DASS BUILDING BLOCKS TO 5G-DIVE ARCHITECTURE.

WP1 Components	WP2 Building Blocks	Main Functionalities
Data Dispatcher	Data collection	It is designed as the main part of the DASS. It is responsible for managing the data collection of the business processes, operation support systems and the verticals services operation. It has an interface with the Edge computing infrastructure as well to leverage current context information from the local environment and the virtualization infrastructure. Interacts internally with the Data pre-processing functionality and the data storage functionality.
	Privacy mechanisms enabler	
Data Pre-processing	Set of pre-processing data activities e.g. data cleaning, data filtering, data grouping, data normalization, data anonymization, data transformation, data compression	It is responsible for the different tasks required to transform the raw input data into a proper understandable common format that can be used for ML models for training and evaluation purposes in the IESS.
	Prepared data	
Data storage	VSS/OSS/BSS & Edge infrastructure prepared data storage	It is responsible for storing the Vertical, Operational and Business support system's information as well as the edge computing infrastructure. The geographically distribute storage should provide location transparent access to the data to the different applications. Additionally, it supports distributed computations store in a given path, making it possible to analyse sensitive data where they are generated.
	Geo-distributed storage	
	Distributed computed values	

2.3.1.2. DASS Workflow Example

Figure 2-15 depicts an exemplary DASS workflow to help better understand the DASS functionalities and their interactions:

1. The Publisher e.g. a running application generates some raw-data regarding its utilization and its current state, this raw-data is transferred to the Data Dispatcher.
2. The Data dispatcher receives the raw data coming from a Publisher and then performs information collection regarding the source of data, it then forwards it to the Data pre-processing
3. When the Data Pre-processing receives the raw-data it starts performing a data cleaning which involves removing or correcting records with corrupted or invalid values from raw-data, as well as removing records that are missing a large number of columns.
4. Next the Data Pre-processing performs data filtering or feature extraction which means reducing the number of features by creating lower-dimension, more powerful data representations using techniques such as PCA, embedding extractions and hashing.
5. Following is the data grouping which involves selecting a subset of the input features for training the model and ignoring irrelevant or redundant ones. This can also involve simply dropping features if the features are missing a large number of values.
6. Other Data pre-processing operations includes data normalization, data anonymization, data transformation and data compression. Once the data is ready to be used it is sent to the Data Storage component.
7. The Data dispatcher receives a notification that the prepared data is ready from the Data Pre-processing and enforces authorization mechanisms to make sure the information is delivered to the interested subscriber e.g. the IESS.

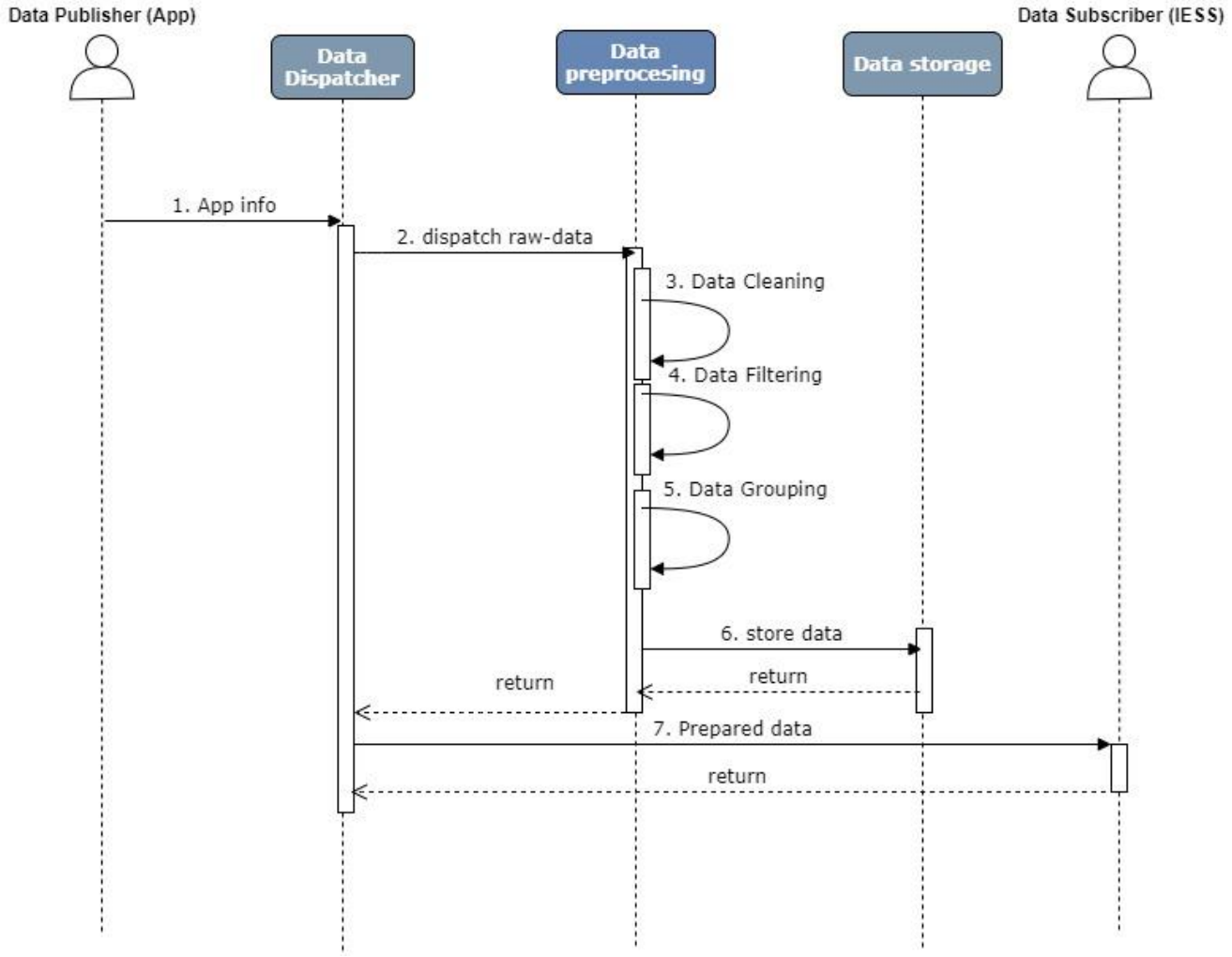


FIGURE 2-15: DASS WORKFLOW EXAMPLE.

2.3.1.3. DASS Baseline implementation

The DASS baseline implementation shown in Figure 2-16 has been contributed to the Eclipse Zenoh Project as open source [18]. At the time of writing this deliverable, the implementation covers the Data Dispatcher functionality, which is provided by the zenoh.net protocol as detailed in Appendix 8. The code is written in the Rust [19] programming language due to its portability, low memory footprint, and safe memory model that prevents a series of common bugs like uninitialized pointers and concurrent memory access. Moreover, the code is written adopting the asynchronous programming paradigm, resulting in better scalability in terms of number of subscribers and publishers. Current Zenoh implementation supports several operating systems such as: Linux, Windows and Mac OS X.

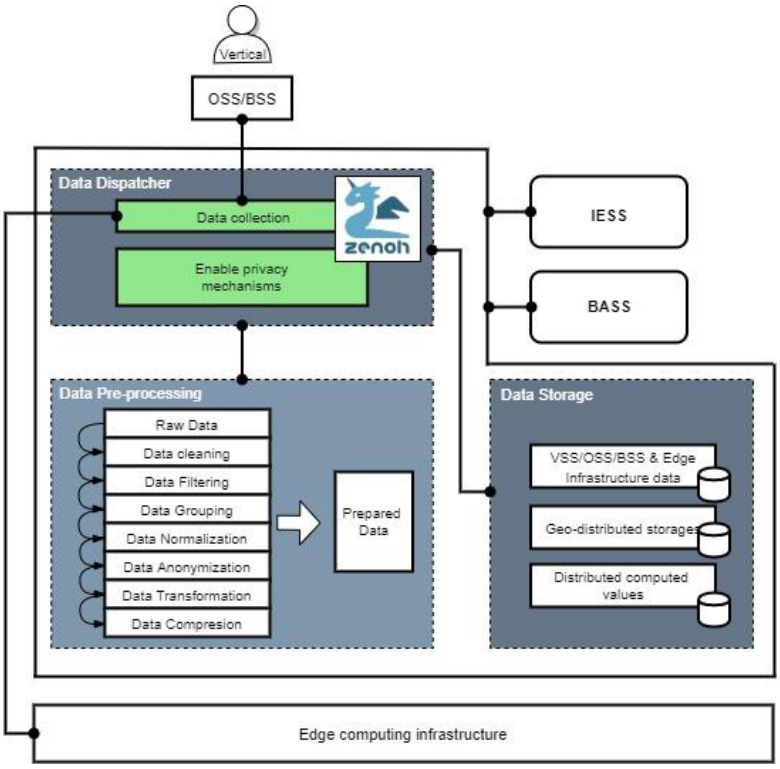


FIGURE 2-16: DASS BASELINE IMPLEMENTATION.

Initial performance tests for 1-to-1 communication were carried out in two different configuration scenarios: (1) peer-to-peer and (2) routed/brokered. These tests are run on a Linux Laptop equipped with an Intel Core i5 and 2 GB of RAM. All communications occur in localhost.

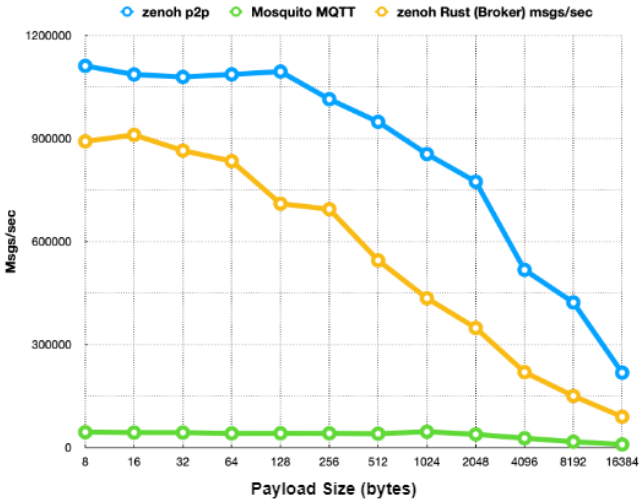


Figure 2-17 Figure 2-17 shows the results in terms of messages per second for different payload sizes (from 8 to 6384 bytes). The zenoh.net results (p2p and routed/brokered) are then compared against Mosquitto MQTT [20], a widespread pub/sub protocol. As it can be seen, zenoh.net significantly outperforms MQTT in terms of msg/s, exceeding 1 million msg/s for small messages.

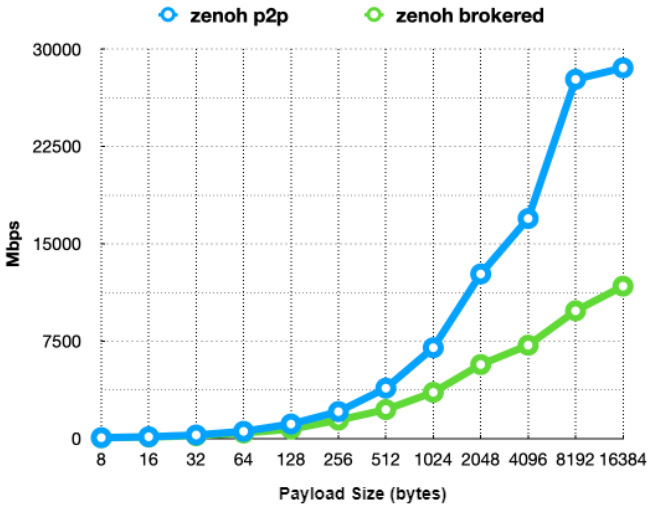


Figure 2-18 shows the throughput achieved by zenoh.net (please note that MQTT is not reported because it results to be out of scale and not appreciable). As it can be seen, as the payload size increases, the throughput increases accordingly, reaching ~29 Gbps in case of p2p communication and 16384 bytes payload.

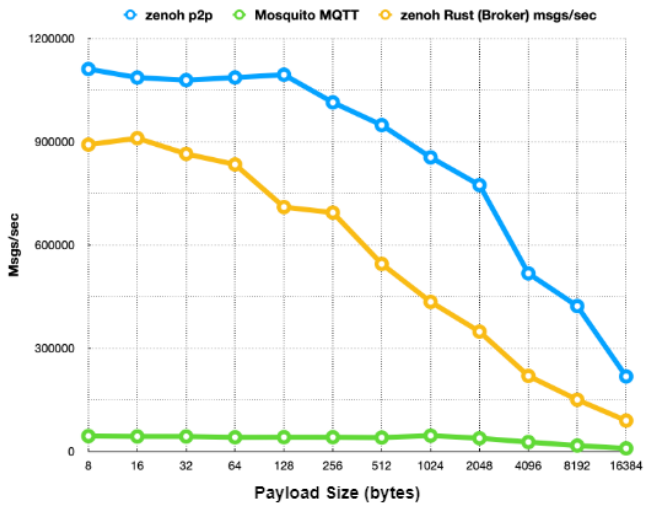


FIGURE 2-17: ZENOH.NET MSG/S.

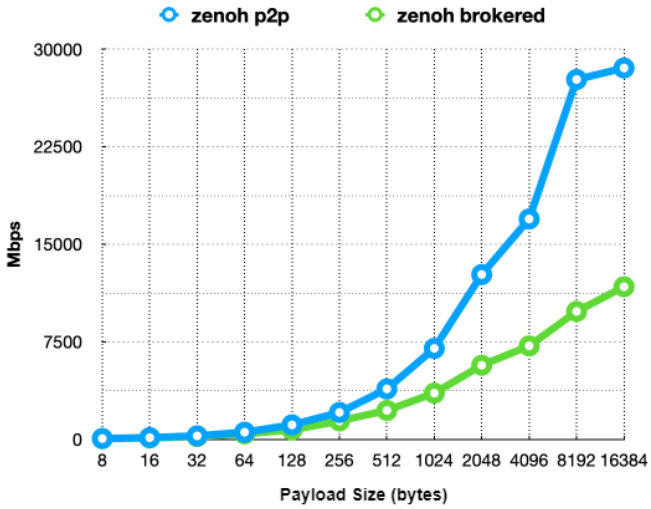


FIGURE 2-18: ZENOH.NET THROUGHPUT.

A second test has been performed to test the fanout of the zenoh.net implementation, that is how the overall zenoh.net system behaves when increasing the number of publishers and subscribers. This test has been performed on a Dell server equipped with 2x Intel Xeon 6130 (64 cores in total) and 128 GB of RAM. The communication occurs in localhost and the p2p communication pattern is employed.

FIGURE 2-19: ZENOH.NET FANOUT.

Figure 2-19 shows the aggregated results in terms of msg/s for any combination of simultaneous 1, 2, 4, 8, 16, and 32 publishers/subscribers in case of 8 bytes data payload. As it can be noticed, the system is capable of handling up to 6 million messages per second depending on the configuration.

2.3.2. Business Automation Support Stratum

The Business Automation Support Stratum (BASS) has been conceived in the 5G-DIVE project as an evolution of the current control systems where an operator oversees the business processes administration. This enhancement provides streamlined processes to facilitate the interaction with the verticals while coordinating all the interactions with the resources and systems underneath.

The BASS will automate the orchestration of the resources and their lifecycle. Besides, the BASS will verify the end-to-end business process KPIs identifying anomalies to minimize the business impact using the enforcement of service level agreements (SLAs). These SLAs are going to be ensured in most cases by leveraging the AI/ML capabilities of the Intelligence Engine Support Stratum (IESS).

2.3.2.1. BASS Architecture

The BASS architecture depicted in Figure 2-20 includes different building blocks which are described in Table 2-6.

TABLE 2-6: BASS BUILDING BLOCKS.

Proposed WP2 Building Blocks	Functionalities
Business Translator	Receives the requests from the verticals, which contain the service descriptor and the service level agreement descriptors for a service.
	Translates from the business to the technical domain, this translation is used internally in the BASS to manage the lifecycle of the vertical service.

	Recommends the SLAs for the given service.
Vertical Service Manager	Manages the lifecycle of the vertical service, namely, deploying, deleting, and updating services using the Orchestrator Driver (OD).
	Triggers the SLA Enforcement for the service and communicates with the IESS in case the service requires some AI/ML functionality.
Orchestrator Driver	Driver used by the BASS to communicate with the Resource Orchestrator creating, destroying, or scaling the resources associated with the service.
SLA Enforcement Manager	Responsible for the lifecycle management of the SLA Enforcement Closed-Loops.
SLA Enforcement Closed-Loop	Makes auto-scaling decisions over a service.
	Enforces the defined SLAs based on the monitoring information gathered via the DASS passively and the Active Monitoring.
Active Monitoring	Responsible for identifying the relevant information to assess the performance associated with a given vertical service, as well as for actively retrieving the identified information from DASS.
Vertical Service Blueprints Catalogue	Repository containing vertical service blueprints, which provides the vertical template with the specific parameters for the service

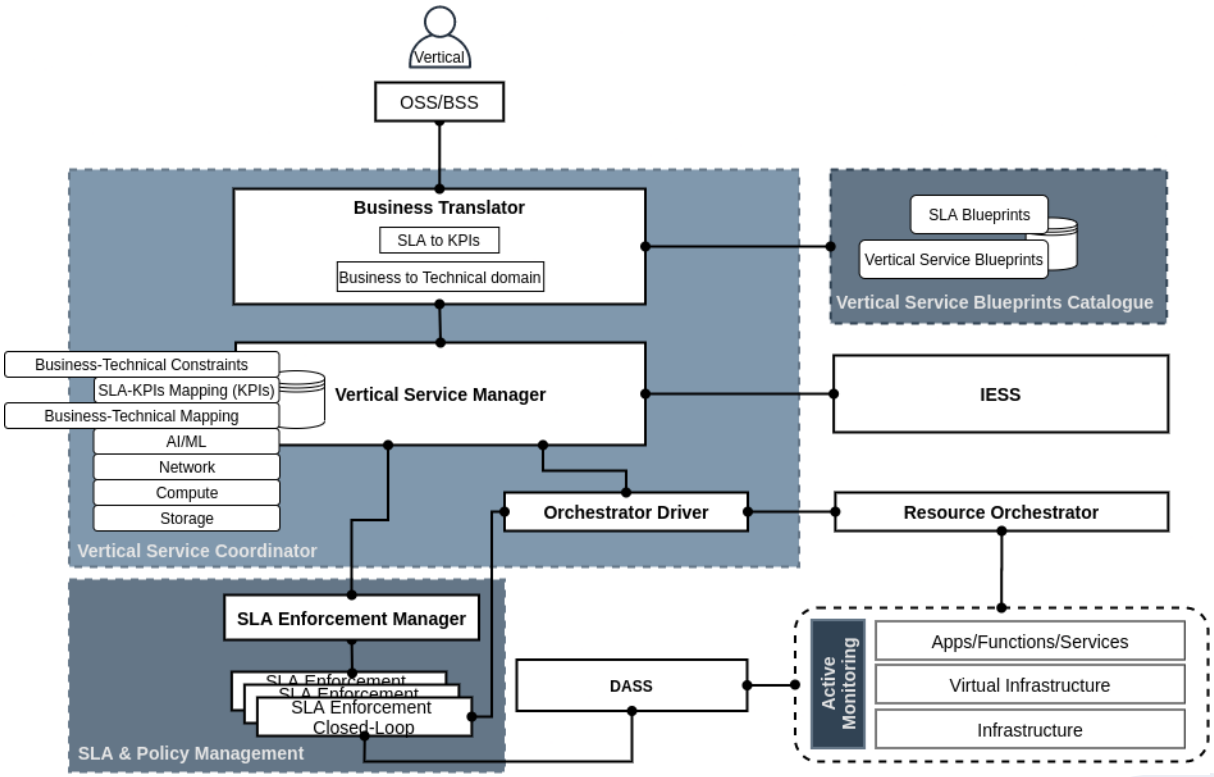


FIGURE 2-20: BASS ARCHITECTURE.

Table 2-7 provides a mapping between the above BASS building blocks and the 5G-DIVE baseline architecture presented in D1.1 [12]. Except for the External Resource Federation Support functionality, all the building blocks have been designed in the first year of the project.

TABLE 2-7: BASS MAPPING TO 5G-DIVE ARCHITECTURE.

WP1 Components	WP2 Building Blocks	Main Functionalities
Vertical Service Coordinator	Business Translator	It is designed as the main part of the BASS. It will be responsible for managing the lifecycle of the business processes, listening to the requests from the verticals, translating them from the business domain to the technical domain. In addition, the VSC will communicate with the IESS in order to leverage the IESS functionalities regarding AI/ML, in case the vertical provides a request describing AI-based components.
	Vertical Service Manager	
	Orchestrator Driver	
SLA & Policy Management	SLA Enforcement Manager	It is designed to provide the necessary functionalities to guarantee that the end-to-end vertical service is compliant, reacting if necessary, before service violations occur.
	SLA Enforcement Closed-Loop	
Vertical Service Blueprints Catalogue	Vertical Service Blueprints Catalogue	In order to assist the verticals to comply with the different descriptors needed by the BASS, a Vertical Service Blueprints Catalogue (VSBC) has been designed. It will contain the templates and blueprints required to be able to interact with the BASS. Both the templates and blueprints should be completed by the vertical, which will be further translated by the VSC to a deployment and guarantee of a business process.
Active Monitoring	Active Monitoring	It is the component responsible for probing the resources and publishing the monitoring information in the DASS pipeline. Making it accessible to the BASS and, to any other component in the 5G-DIVE platform.

2.3.2.2. BASS Exemplary Workflow

Figure 2-21 depicts an exemplary BASS workflow in order to help gain a better understanding about the BASS functionalities and corresponding actions and interactions:

1. The Vertical first generates a request for a service using a Vertical Service Descriptor (VSD), and expresses its constraints through the SLAs descriptor, explaining what SLAs to enforce and measure for a given service.
2. The BT translates business-oriented VSDs into technical-oriented service descriptors (Technical Domain Descriptors), and also simultaneously recommends to the vertical the SLAs and KPIs.
3. The Vertical Service Life-Cycle Manager (LCM) manages the services using the technical-oriented service descriptors through the Orchestrator Driver.
4. When a Vertical Service is deployed by the Orchestrator, the Vertical Service LCM requests for SLA Monitoring and Enforcement, according to the specified vertical SLAs.
5. Using the request from the Vertical Service LCM, an SLA Enforcement LCM instantiates a closed loop to guarantee the SLAs defined for the Service.
6. The SLA Enforcement closed loop is responsible for requesting and retrieving monitoring information from the DASS. With this valuable information, it can forecast, detect, and make decisions to prevent SLA violations. This guarantees the adequate performance of the service. If an SLA violation is forecasted, the closed-loop will auto-scale the service, such as scaling up or down, scaling in or out, and it will be able to migrate the service, too. In case the closed loop fails to enforce the SLAs, the vertical will be notified.

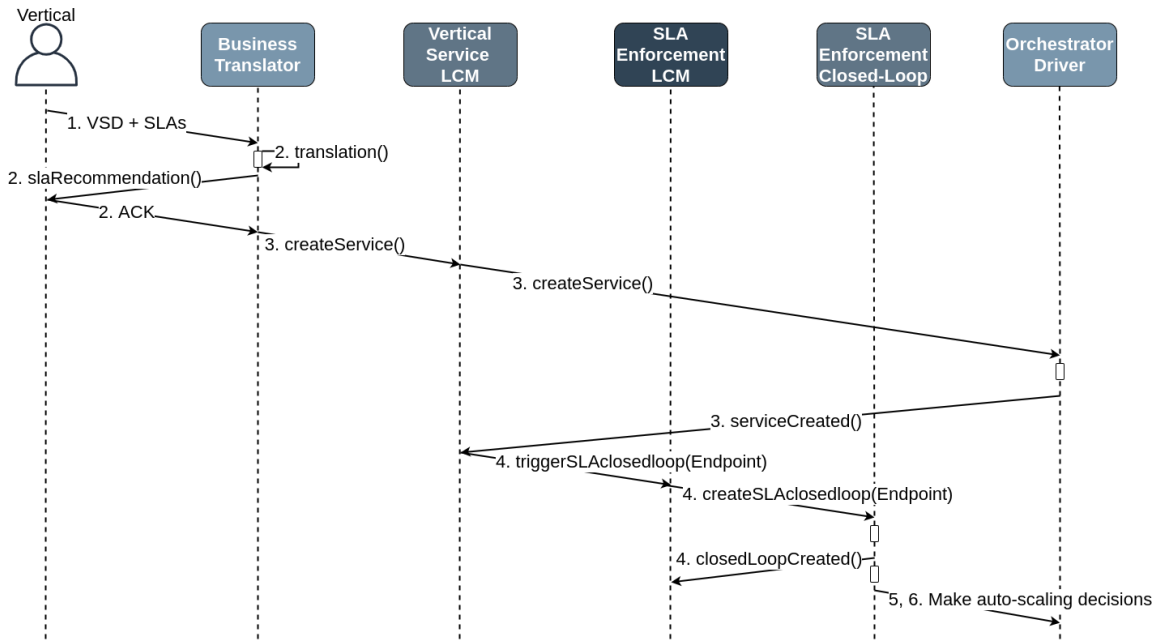


FIGURE 2-21: BASS EXEMPLARY WORKFLOW.

2.3.2.3. BASS Baseline Implementation

Figure 2-22 highlights the building blocks implemented in the first year of the project. These include: the Business Translator, the Vertical Service Manager, the Orchestrator Driver, and the Resource Orchestrator. In this first release of the implementation, the Spring Framework [21] has been selected to develop the Business Translator, the Vertical Service Manager, and the Orchestrator Driver due to its integration with other Java frameworks, its powerful database transaction management capabilities, and its native service-based architecture integration. Also, the Business Translator and the Vertical Service Manager will make use of Jinja [22] templates to interact through predefined data models with other entities.

MongoDB [23] is going to be used to store the different descriptors provided by the verticals and the internal data model of the BASS, making this storage function one of the main functionalities of the Vertical Service Manager. For the Resource Orchestrator, a certified lightweight distribution of Kubernetes (K3s) [24] has been selected, as F0rce [25] will not be available until the second year of the 5G-DIVE project. K3s offers a highly available solution designed for production workloads in unattended, resource-constrained, and remote edge locations.

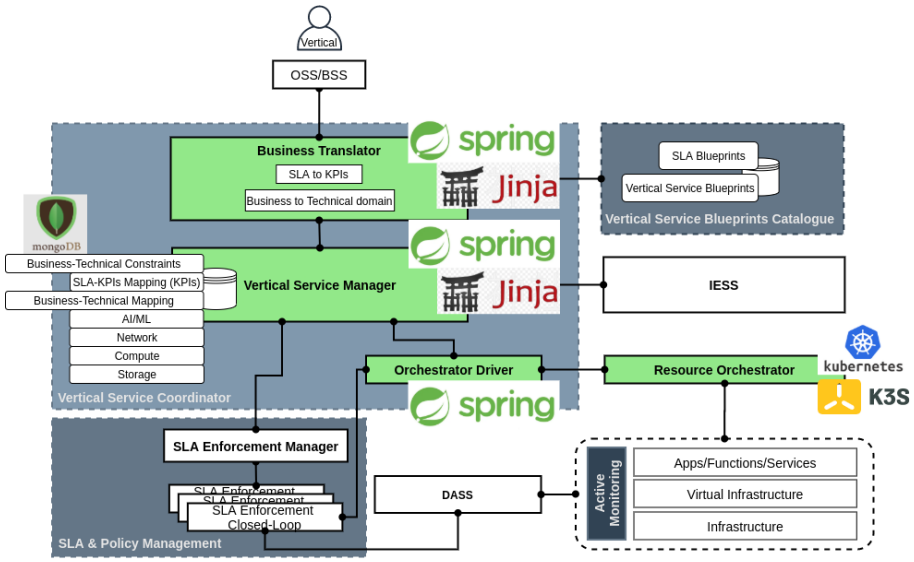


FIGURE 2-22: BASS BASELINE IMPLEMENTATION.

2.3.2.4. Intelligent SLA Enforcement Closed Loop

The SLA Enforcement Closed-Loop is part of the SLA and Policy Management component contained inside the BASS. As shown in Figure 2-23, the SLA Enforcement Closed-Loop makes use of the Active Monitoring component included in the BASS. However, the SLA Enforcement Closed-Loop data will be routed to its destination through the DASS. Once the data is collected, it will be processed and fed to the SLA enforcement algorithms to assist during the SLA enforcement.

After the monitoring information is collected, the SLA Enforcement Closed-Loop takes decisions and executes actions according to the current state of the underlying infrastructure and services. These decisions are taken by a deep reinforcement learning (DRL) algorithm that is trained by IESS.

The implementation of the intelligent SLA Enforcement Closed Loop is planned for the second year of the project.

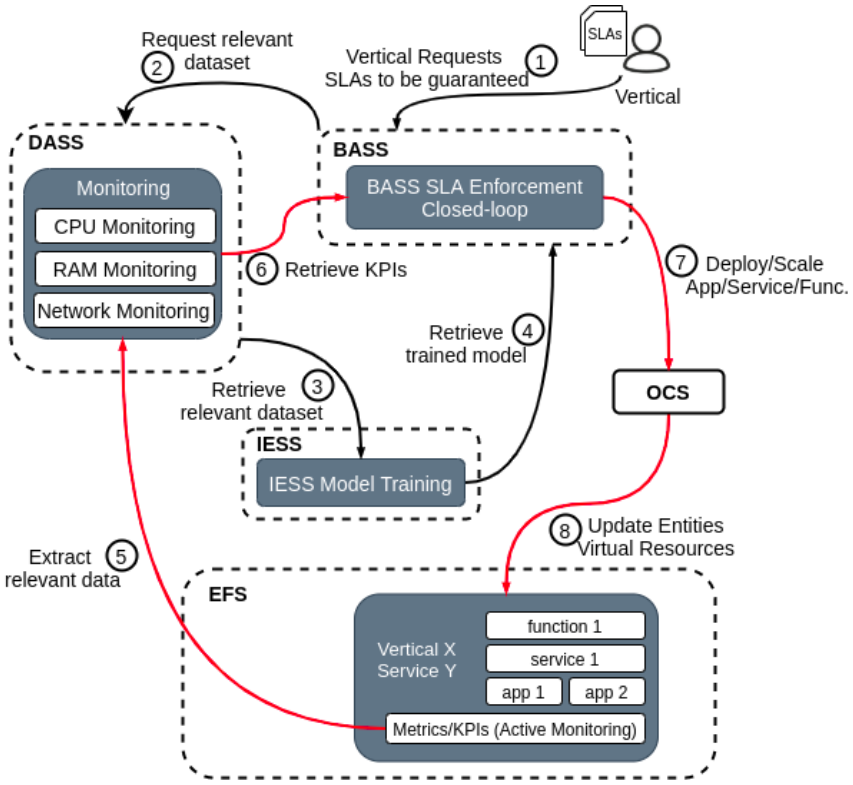


FIGURE 2-23: SLA ENFORCEMENT CLOSED-LOOP.

2.3.3. Intelligence Engine Support Stratum

The Intelligence Engine Support Stratum (IESS) is envisioned as an Artificial Intelligence Platform which uses data-driven algorithms to make predictions, classifications, and decisions. This allows to provide a tool kit to develop and train intelligent models at the Edge/Fog.

The IESS selects an optimal algorithm to train a given model based on context information using AutoAI/AutoML Engines. These engines automate the selection of the optimum algorithm, the hyperparameters, and the training process. When the IESS chooses the algorithm, data is requested to train the model. When the model is trained, the model can then predict, classify, and decide from new inputs.

In 5G-DIVE, IESS supports the application of AI/ML to three main areas as follows:

- Applications: Vertical applications can make use of the intelligence capabilities provided by the IESS, thus adding desired functionalities, and enhancing their performance, such as in an object detection application.
- Orchestration: The goal of the SLA enforcer is to intelligently identify and forecast SLA violations, therefore requesting the IESS to provide trained models when the deployed services can be scaled up or down, scaled in or out, and migrated.

- Networking: Intelligent networking policies will need to be deployed to guarantee certain SLAs are met, such as response time or capacity.

2.3.3.1. IESS Architecture

The IESS architecture depicted in Figure 2-24 includes different layers as described in Table 2-8.

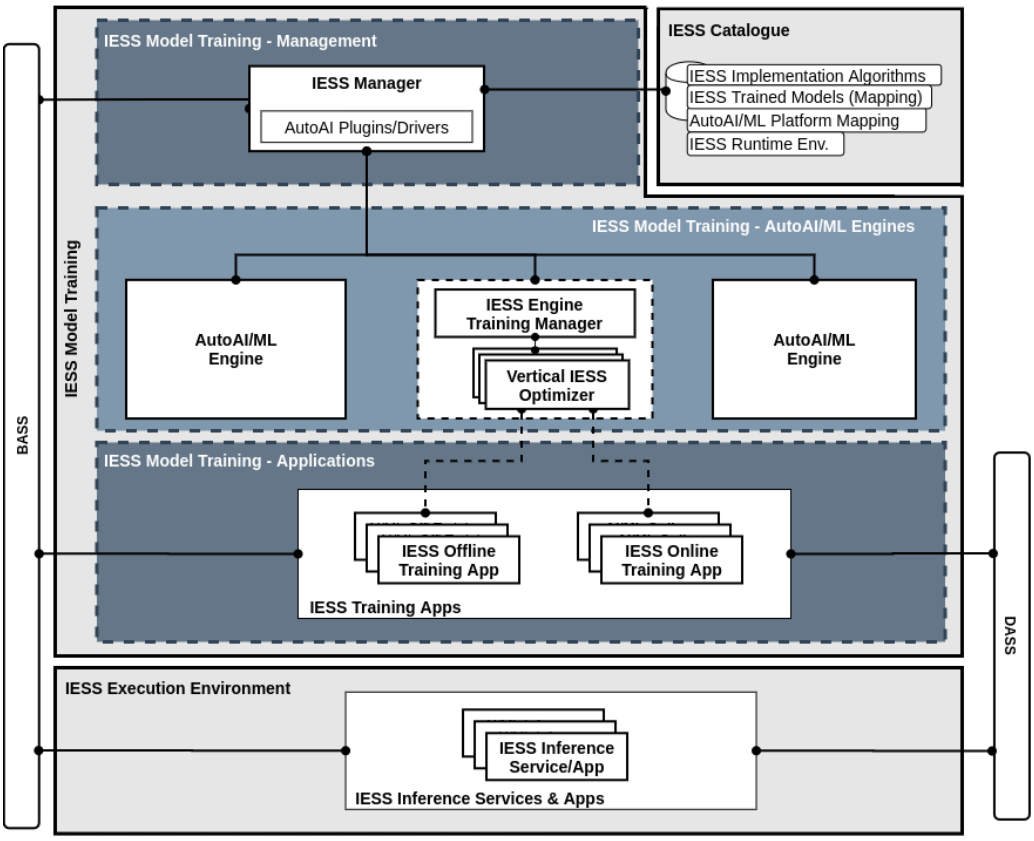


FIGURE 2-24: IESS ARCHITECTURE.

TABLE 2-8: IESS LAYERS.

Proposed WP2 Layers	Functionalities
IESS Model Training – Management	This layer manages the life-cycle of the training stage, is the entry point to request AI/ML training and contains the main component of the IESS, the IESS Manager.
IESS Model Training – AutoAI/ML Engines	This layer contains different AutoAI/ML engines and is responsible for the training stage itself.
IESS Model Training – Applications	This layer contains the resources where the model will be trained as applications.

Table 2-9 describes the detailed building blocks which are contained into the proposed IESS design. Note that not all the building blocks are contained inside a layer.

TABLE 2-9: IESS BUILDING BLOCKS.

Proposed WP2 Building Blocks	Functionalities
IESS Manager	This is the entry point for the BASS to request AI/ML training and manages the life-cycle and triggers the different AutoAI/ML engines for each AI/ML request.
AutoAI/ML Engine	Responsible for selecting the optimum dataset and also the optimum AI/ML algorithm, optimizing its hyperparameters, and verifying that the trained model results strictly match the AI/ML requirements received by the vertical. Note that in the IESS there exists the possibility for the vertical to plug in a distinct AutoAI/ML engine in the IESS Manager. The AutoAI/ML engines are composed of (i) the IESS Engine Training Manager, which is responsible for the lifecycle of (ii) the Vertical IESS Optimizers, which train the algorithms, verify the trained models results and select the hyperparameters previously stated.
IESS Training Apps	Applications where the training algorithms will train the models.
IESS Inference Services & Apps	Loads the trained model, perform the actual inference.
	Receives the data inputs to make the inference
	Publishes the inference results to the DASS pipeline.
IESS Catalogue	There is no extension from WP1.

Table 2-10 provides a mapping between the above IESS building blocks and the 5G-DIVE baseline architecture presented in D1.1 [12].

TABLE 2-10: IESS MAPPING TO 5G-DIVE ARCHITECTURE.

WP1 Components	WP2 Building Blocks	Main Functionalities
IESS Model Training	IESS Manager	It is designed as the main component of the IESS, responsible for training the different optimization algorithms to comply with the requirements provided by the verticals in terms of losses or accuracies. Besides, the IESS Model Training could be used to meet the requirements in terms of orchestration or networking such as availability, reliability or capacity that are requested by the 5G-DIVE platform to guarantee the SLAs.
	AutoAI/ML Engine	

IESS Execution Environment	IESS Inference Services & Apps	Responsible for consuming the trained model and publishing the inference results.
IESS Catalogue	IESS Catalogue	In this catalogue, the trained models, algorithms, training runtimes, or mappings between AutoAI/ML platforms and their suitable purpose are stored.

2.3.3.2. IESS Exemplary Workflow

Figure 2-25 represents an exemplary workflow to help better understand the IESS functionalities and how they work:

1. Firstly, the IESS Manager receives an AI-based request from the BASS.
2. The IESS Manager extracts all relevant information such as AI/ML purpose, final accuracy, losses, etc.
3. The IESS Manager chooses the optimum AutoAI/ML engine(s) to train a model based on the requirements and triggers an IESS optimizer using different plugins/drivers of the distinct platforms.
4. When the IESS Manager triggers the closed-loop, the IESS Training Manager is responsible for the IESS closed-loop life-cycle. In addition, the IESS Training Manager instantiates an IESS closed-loop per selected algorithm. Note that if there exists more than one selected AutoAI/ML engine, there will be more than one IESS Training Manager.
5. The IESS optimizer is responsible for training a model, verifying the requirements, and retraining if needed (modifying the hyperparameters).
6. When the training stage is finished, the IESS Manager stores the trained model and notifies the BASS to deploy the inference app.

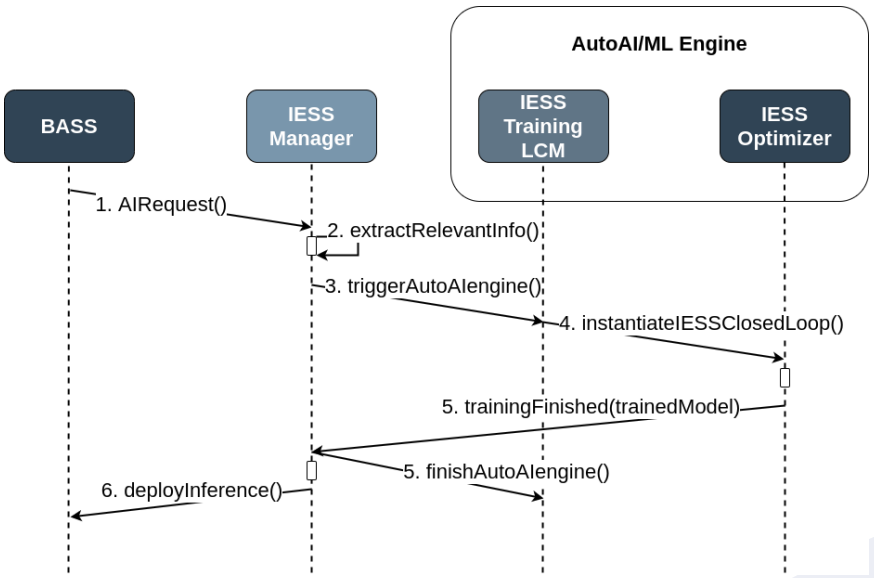


FIGURE 2-25: EXEMPLARY IESS WORKFLOW.

2.3.3.3. IESS Baseline Implementation

Figure 2-26 depicts the first-year implementation for the IESS, where the IESS Manager and the AutoAI/ML Engine are primarily targeted.

The IESS Manager, which is the main component of the IESS architecture, is implemented using the Spring Boot framework. Spring has been selected as it is light weight and it has many integrations with other relevant frameworks. The latter is the most important feature as IESS Manager will need to integrate with different AutoAI/ML engines such as H2O.ai or TPOT, and most of their clients are released in Java [26].

For the AutoAI/ML engine, there are some top-notch and near state-of-the-art implementations of AutoAI/AutoML engines such as H2O.ai [27], TPOT [28], AutoGluon [29], NNI [30] and Ray [31]. Some of them offer a wide variety of interfacing options, automatic feature engineering and data cleaning, integrated dashboard, ARM support, GPU acceleration, all of them offer distributed computing. However, as the first engine to implement, we have selected H2O.ai as it is the most complete tool in the set of surveyed engines (See Appendix 10), and as such it will be integrated for the first release of the IESS.

Finally, we have selected several storage options for each type of data that needs to be stored in the IESS Catalogue. The first one is Docker Registry [32], for storing the runtimes for training. Then, a Git server to store the implementation of the algorithms. And the last storage option is a MongoDB database in order to store the trained models in a suitable format (e.g. ONNX [33], HDF5 [34], JOBLIB [35]), including the trained models' metadata (e.g. purpose, accuracy, etc), the algorithms implementation metadata (e.g. suitable purpose, AI/ML Platform, etc) and the mapping between the AutoAI/ML platforms and their purpose.

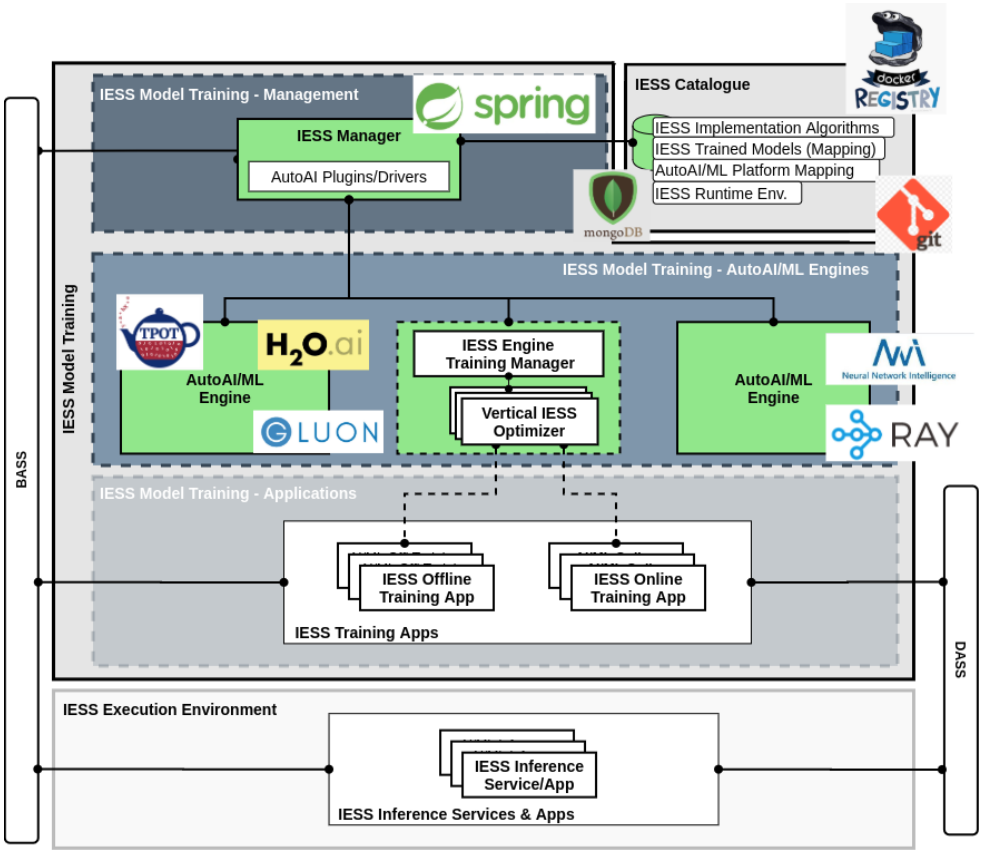


FIGURE 2-26: IESS BASELINE IMPLEMENTATION.

2.3.4. Intelligence Engine

2.3.4.1. Introduction

An intelligence engine is the machine learning algorithm that runs as part of the IESS or BASS to provide inference in the context of 5G vertical service automation. Machine learning and cognitive techniques have made impressive breakthroughs and are being used in many fields such as computer vision, speech recognition and other bioinformatics. One of the most prominent features of machine learning algorithms is dealing with complex problems. They allow systems to perform like human-being with tasks requiring prediction, classification and decision making. There are several types of machine learning systems which can be broadly classified based on if they:

- Require human supervision or not (supervised, unsupervised, semi-supervised or reinforcement learning).
- Learn on the fly or offline (online learning or batch learning).
- Infer by detecting patterns or by comparing new data to known data (model-based or instance-based learning).

This classification is not exclusive as categories can be combined to fit specific application requirements. In Appendix 9, we present a survey of several machine learning algorithms that are relevant to the

applications targeted in 5G-DIVE project including artificial neural networks, reinforcement learning and imitation learning.

Recently, machine learning has caught much more attention in the fields of communication and computing. Firstly, bringing machine learning into the network domain is not a new initiative but the work in this area is yet to be prototyped or deployed in the field. For example, a knowledge plane for the Internet is proposed in [36]. The knowledge plane relies on machine learning techniques to operate the network. This paradigm may reshape the way the network is operated and managed as it brings many benefits to network control such as automated configuration and recommendation. However, data networks are naturally distributed systems and thus deploying a knowledge plane is challenging since the learning process will be applied to a partial view of the network. With software-defined networking (SDN) emergence, the network control is logically centralized and a global view of the entire network is easily available. To this end, the work in [37] proposes to integrate machine learning, SDN and data analytics to augment network operation and control through artificial intelligence.

Furthermore, edge computing and machine learning are predestined for each other. On one hand, edge computing environment is very suitable for machine learning for three reasons: 1) machine learning depends on large amount of data for modelling, and edge computing can collect the required data, 2) machine learning requires training which is a computationally intensive process, edge computing can provide the required computing resources, 3) online machine learning requires powerful resources and low-latency inferences, edge computing provide resources at the edge. On the other hand, machine learning is appropriate for edge computing for two reasons: 1) edge computing infrastructure is very dynamic and complex, machine learning can seamlessly adapt to changes and new data, 2) edge computing orchestration requires a lot of hand-tuning tasks to adhere to service level agreements, machine learning techniques can provide insights and enable automation.

2.3.4.2. Intelligence Engines in 5G-DIVE Architecture

An intelligence engine is an integral and essential part of the IESS which transforms raw data into knowledge that can then be utilized to intelligently orchestrate and automate vertical services. The increasingly growing complexity of edge computing operations can no longer be managed by simple techniques that automate repetitive manual tasks. By leveraging the recent advances in data analytics and machine learning, true intelligent automation can be made possible. Fundamentally, 5G-DIVE augments the edge platform developed in the 5G-CORAL project by integrating DEEP layer. Among the crucial components within the DEEP is the intelligence engine of the IESS. The orchestration and automation operations relay heavily on the knowledge provided by the intelligence engine inference. To this end, once the model training is accomplished by the IESS, the intelligence engine run-time is deployed as an EFS application in the edge platform. An EFS application is defined as a computing task comprised by at least one atomic entity deployed in EFS for consumption by end users and third parties. This way, the inferences performed by the intelligence engine are considered as EFS services and consumed by other EFS applications/functions. For example, an emergency relief intelligence engine that performs victim detection could expose victim location as a service to be consumed by other applications in the system to facilitate prompt response.

In the core design of DEEP, intelligence engines share a common data flow to train models, provide inference and enable vertical service automation. Fundamentally, the data flow initiates at the source of data which is generated by real devices or simulated environments. The data flow of intelligence engines can be described during the follow logical steps also depicted in Figure 2-27:

(a) Pre-processing and Storage – The DASS receives raw data from various sources and preforms data pre-processing and storage operations. Pre-processing operations such as cleaning, scaling, transformation and compression prepare the data for training and inferencing. For example, in case of offline learning intelligence engines, datasets are pre-processed and stored in preparation for offline training requests.

(b) Model Training – The IESS subscribes to relevant datasets that are used to train models. In the case of intelligence engines that are capable of incremental learning, IESS can subscribe to a pre-processed data stream, rather than stored datasets, to learn on the fly.

(c) Trained Model Inferencing – The trained intelligence engines are deployed as applications in the edge platform (e.g., EFS Application). Here, the EFS application subscribes to a new data, stored or live pre-processed stream, to infer and generalize by detecting patterns or by comparing the new data to learned examples (model-based or instance-based learning).

(d) Measurement Reports – The trained intelligence engines can generate measurement reports that can be used to auto-scale the current deployment. The measurement reports are exposed to the DASS which acts as an anchor for telemetric data.

(e) KPI Monitoring – The BASS subscribes to the measurement reports performed by the intelligence engines to actively monitor the committed SLAs. Lastly, the BASS acts on the reports and makes decisions to auto-scale the deployment based on the infrastructure and services states.

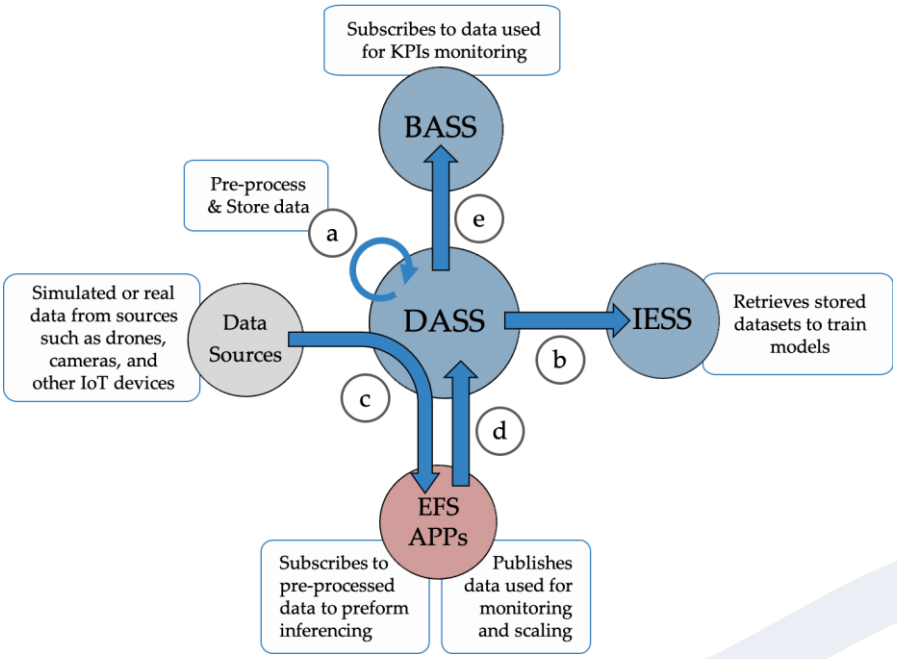


FIGURE 2-27: IE DATA FLOW.

Several intelligence engines have already been identified for each of the use cases, such as predictive maintenance in digital twinning, object defect detection in ZDM, radio security for massive MTC, geolocation and image analytics in ADS use cases. These are described in the context of each use case for the I4.0 and ADS pilots in section 3 and section 4 respectively.

2.3.5. Potential Applicability of Distributed Ledger Technologies

A distributed ledger (DLT) is a database that is consensually shared and synchronized across multiple sites, institutions, or geographies, accessible by multiple people. The participant at each node of the network can access the recordings shared across that network and can own an identical copy of it. In DLTs, there is no central administrator or centralized data storage [38][39].

Depending on the technology used, the ledger can include the signing of smart contracts, and different consensus mechanisms to reach an agreement between peers, like Proof-of-Work, Proof-of-Stake, Delegated Proof-of-Stake, Byzantine Fault Tolerance, Directed Acyclic Graphs, etc.

They can also be differentiated between Permissioned and Permissionless. Permissioned blockchains only allow a certain user to perform actions on the network, wherein permissionless everything is public and everybody is allowed to participate.

A distributed ledger is well suited in all of the I4.0 and ADS use cases, as a key part of the federation process in order to build trust. It will enable the domains to have a public ledger where the transactions of resources can be stored, consulted and validated by each peer joining the federation.

Examples of known open source DLTs solutions are Hyperledger Fabric [41], Corda [42], Quorum [43], Openchain [44], and Enterprise Ethereum [45]. Our goal for the first year will be to explore Hyperledger Fabric, examining more possibilities during the second year.

We foresee that DLT frameworks could be used in 5G-DIVE as a part of the federation process in order to be used as a common marketplace of resources between Fog and Edge. This marketplace should support the verticals to exchange resources without relying on each other. This marketplace is willing to be explored by using Hyperledger Fabric, an open-source enterprise-grade permissioned distributed ledger technology platform that offers modularity and versatility for a broad set of industry use cases. It delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms:

- First distributed ledger platform to support smart contracts authored in general-purpose programming languages such as Java, Go and Node.js, rather than constrained domain-specific languages (DSL). In 5G-DIVE, we will mainly use Java for developing the BASS and also the IESS. Therefore, we will not need to learn another domain-specific language for this particular feature.
- Permissioned ledger, so the participants are known to each other and therefore, fully trusted. Trust between identified peers is a requirement in 5G-DIVE and needs to be addressed during the federation process.

- Highly modular and configurable architecture. In 5G-DIVE all pieces are modular as it is built up on service-based interfaces. Consequently, this modularity that Hyperledger Fabric offers fits with the overall architecture.
- Does not require a native cryptocurrency to incentivize costly mining or to fuel smart contract execution. In 5G-DIVE, resources are constrained so this is an important feature.
- A novel architecture for transactions called "execute-order-validate". It addresses the resiliency, flexibility, scalability, performance and confidentiality challenges faced by the order-execute model. These features are desired in 5G-DIVE as there exists a requirement stating that the federation process should be performant, scalable and resilient.

3. Detailed Solution for I4.0 Use Cases

3.1. I4.0UC1: Digital Twin

A Digital Twin system consists of mapping the physical and virtual worlds supported by connected data that ties the two worlds together. It comprises computational models or analytic models that are required to describe, understand and predict the real-world object's operational states and behaviours, as well as models that are used to prescribe actions based on business logic and objectives about the corresponding object[61]. These models may include models based on physics, engineering or simulation models, data models based on statistics, and ML and AI capabilities. It may also include visualization models (e.g., 3D modelling, augmented reality) that mirror the physical entity and allow its remote control. Finally, a Digital Twin system contains connected data that in many cases consists of data of the full lifecycle of the object (e.g., navigation instructions, real-time and historical state, installation and configuration data, hardware status data, etc.). In general, the connected data from the physical world (e.g., sensor data) is transmitted to the virtual models to complete the simulation, validation, and dynamic adjustment. At the same time, the simulation data is fed back to the physical world to respond to the changes, improve the operation, and increase the value.

In Digital Twin, the network-assisted robotic concept is still far from being fully exploited and optimized. The distribution of the time-sensitive control logic between the physical objects and the infrastructure places a new set of stringent requirements (including low latency, high reliability and availability and high bandwidth) over the underlying communication medium. These requirements are effectively fulfilled through an end-to-end 5G wireless connectivity as described in Section 2.1.

Thanks to the distributing computing with Edge and Fog, new opportunities arise for Digital Twin to remove the time-sensitive computation algorithms (e.g., robot control algorithms) from the physical objects and place them in a remote Edge/Fog node. Moreover, context networking information (e.g., communication links status) acquired from the access network can be used to adapt the physical objects (e.g., robots, machines) operations. As a result, it is possible to build lightweight, low cost, and smarter physical objects while exploiting the Edge and Fog distributed infrastructure to share the connected data between various virtual models for cooperative operation.

Digital Twin is a key 5G-DIVE I4.0 use case where low latency and reliable communications are simultaneously required. In 5G-DIVE, the Digital Twin solution is based on 5G connectivity integrated with distributed Edge and Fog computing and augmented with automation and AI/ML capabilities provided through the DEEP platform.

5G-DIVE Digital Twin solution includes three subsystems namely, EFS, OCS and DEEP, as follows:

- EFS: The **computation models** that define the behaviour of the real objects are represented as EFS entities and can be implemented using different virtualization technologies (e.g., Docker, LXC, KVM). As an example, the Digital Twin application is a visualization oriented EFS application that gives human understanding to the connected data and enables remote control.
- OCS: As defined in [47], the OCS will compose, control, manage and orchestrate the Digital Twin models. It will be responsible for fully automated management and operation of the distinct modules.

- DEEP platform:
  - IESS: The **analytics models** defining the intelligent and automated behaviour of the Digital Twin are represented as IESS entities. These intelligence engines will analyse, optimize, monitor, and predict the operations of the physical object. More details about the intelligence engines will be provided in the next section.
  - DASS: The **connected data** of the Digital Twin will be collected from the **computation models** and will be pre-processed, stored and shared with the help of DASS.
  - BASS: Defined **SLAs will be enforced** with the help of the SLA and policy management based on the monitoring data from the DASS to ensure proper and correct work of the Digital Twin. This part of the work will be developed in the second year of the project.

3.1.1. Reference Design

Figure 3-1 presents our reference design of the Digital Twin use case. The Digital Twin solution is composed of a computation stack that implements the computation models, an analytics stack that implements the analytics models, and the connected data.

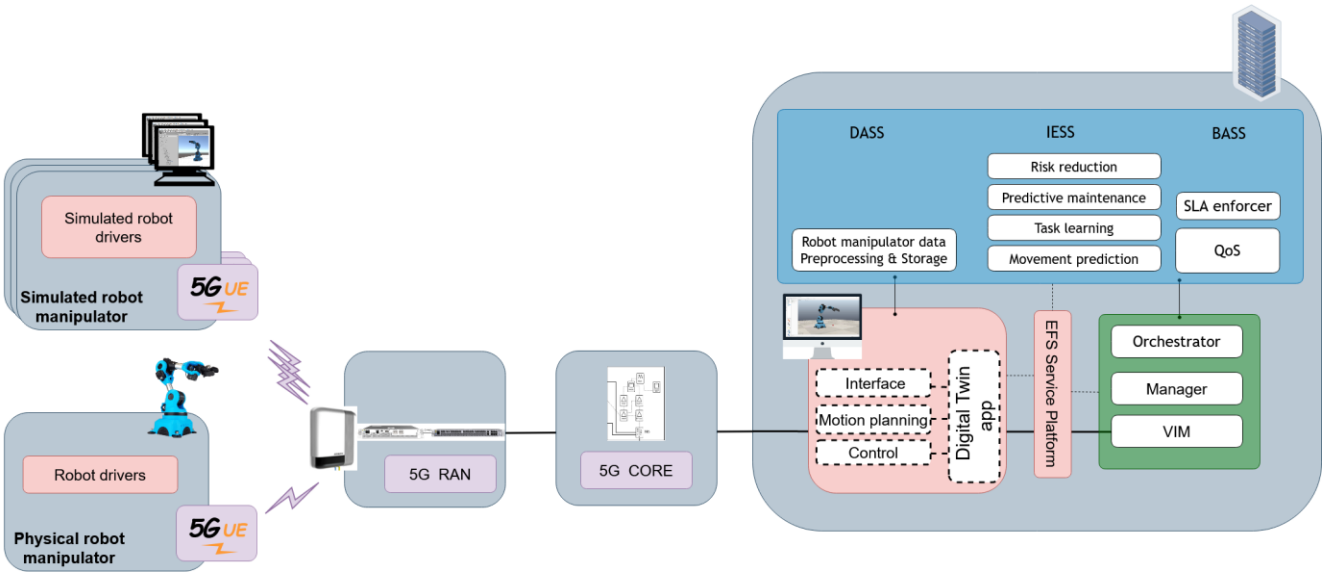


FIGURE 3-1: SYSTEM BLOCK DIAGRAM FOR DIGITAL TWIN.

The computation stack is envisioned as various computation models distributed across the physical object and the EFS:

1. **Drivers layers**: Digital Twin computation models that directly interact with the physical object hardware and are responsible for: (i) making available sensor data and operational states to the other layers, and (ii) executing instructions or navigation commands received from the Control layer.
2. **Control layer**: Is defined as an abstraction layer that allows physical object manipulation. It receives a navigation command or instructions and runs them in a control loop towards the **Drivers**. The loop is then closed by the physical object continuously sending-back the current state.

3. **Motion planning**: Is responsible for finding inverse kinematics and building a path for the robot. The path created consists of a series of navigation commands sent to the control layer.
4. **Interface layer**: Represents a high-level abstraction for the core motion planning functionalities. This layer is the gateway between the operator and the physical object.
5. **Digital Twin app**: Implements 3D models and control mechanisms to visualize the variations of the physical object while the control mechanism enables remote control and maintenance.

The **connected data** exploits the benefits of the edge and fog environment through the DASS for improving several aspects of Digital Twin such as:

1. **Privacy and security:** Characterized by locality of edge and fog, Digital Twin can improve security and privacy of the sensor data by offering restricted access within trusted private domain through the DASS.
2. **Data filtering:** The data generated by different physical objects may often be inaccurate, incomplete, inconsistent, or even duplicate. The DASS can provide new and efficient way of data filtering and sampling to prevent physical objects failures. Moreover, pre-processing, formatting and storage of the data can be performed using the DASS to prepare the data for the analytics models to use.
3. **Bandwidth savings:** The data generated by various physical objects is growing exponentially. These data are conveyed over the network to the cloud where data processing is carried out. The DASS offers bandwidth saving by sending to the cloud only the necessary data.

The analytic stack is envisioned as various intelligence engines that are part of the IESS:

1. **Movement prediction**: focuses on navigation commands predictions to cope with the variability of the wireless medium (e.g., increased jitter or packet loss).
2. **Task learning**: aims at teaching an AI-based agent how to dynamically execute tasks (e.g., sorting objects) without a human in the loop.
3. **Risk reduction**: analyses information from both the robot and sensing devices, along with other contextual information, ensure safe work environment.
4. **Predictive Maintenance**: aims at predicting hardware failures of the robot components using maintenance data and suggesting when maintenance is required.

3.1.2. Analytics and Intelligence

After the data is collected, aggregated and transformed with the help of DASS, the **analytics stack models** comprise of techniques such as statistics, AI/ML and simulations to monitor, simulate and predict the behaviour of the physical robot. As shown in Figure 3-2, the analytics stack models are represented as IESS entities and their applicability to Digital Twin includes movement prediction, task learning, automation and predictive maintenance.

Movement prediction

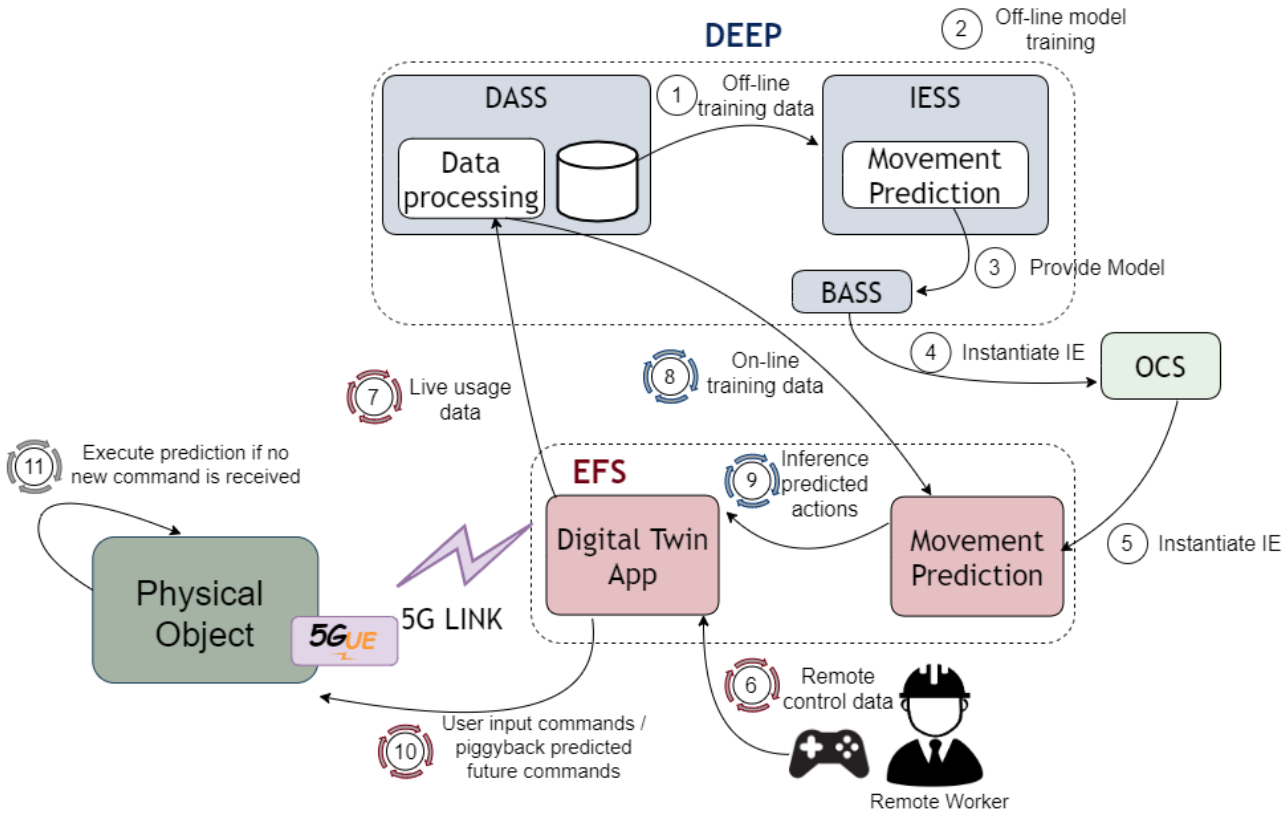


FIGURE 3-2 MOVEMENT PREDICTION MODULE WORKFLOW.

The movement prediction module is used to predict and suggest the next movement based on historical data. In remotely controlling a real object using the Digital Twin App, the prediction can be useful when the conditions of the network are not able to guarantee a correct control. In such cases, trying to remotely control the real object will lead into unwanted (sometimes catastrophic) movements.

The idea comes from the gaming industry. Similar to what has been done in[49], Discrete Time Markov Chains (DTMC) can be applied to the digital twin remote control to predict user input.

Two different types of data are used for the training: (i) real-time usage data, and (ii) historical usage data. Both are processed and stored by the DASS and can be requested on-demand by the IESS for the training. In the example of a robotic manipulator, the data is composed of time sequences of commands, while the states of the DTMC is the combination of the state of the joints of the real robotic arm.

The training (step 1 and 2) aims at finding the best transition probability matrix of the DTMC. In this way, for each state it is possible to predict the most probable next state with a certain accuracy. This computation is performed both off-line, with the use of historical data, and on-line, updating the state transition probability matrix with current usage from the user.

The movement prediction module is then requested and instantiated (step 3, 4 and 5). Each time the user sends commands to the real object (step 6, 7 and 8), the movement prediction module sends predicted future commands to the Digital Twin App (step 9), which sends them piggybacked in the

message with the current command issued by the user (step 10). If bad network conditions are detected, the predicted commands are executed. Otherwise the predicted commands are discarded (step 11).

Task learning

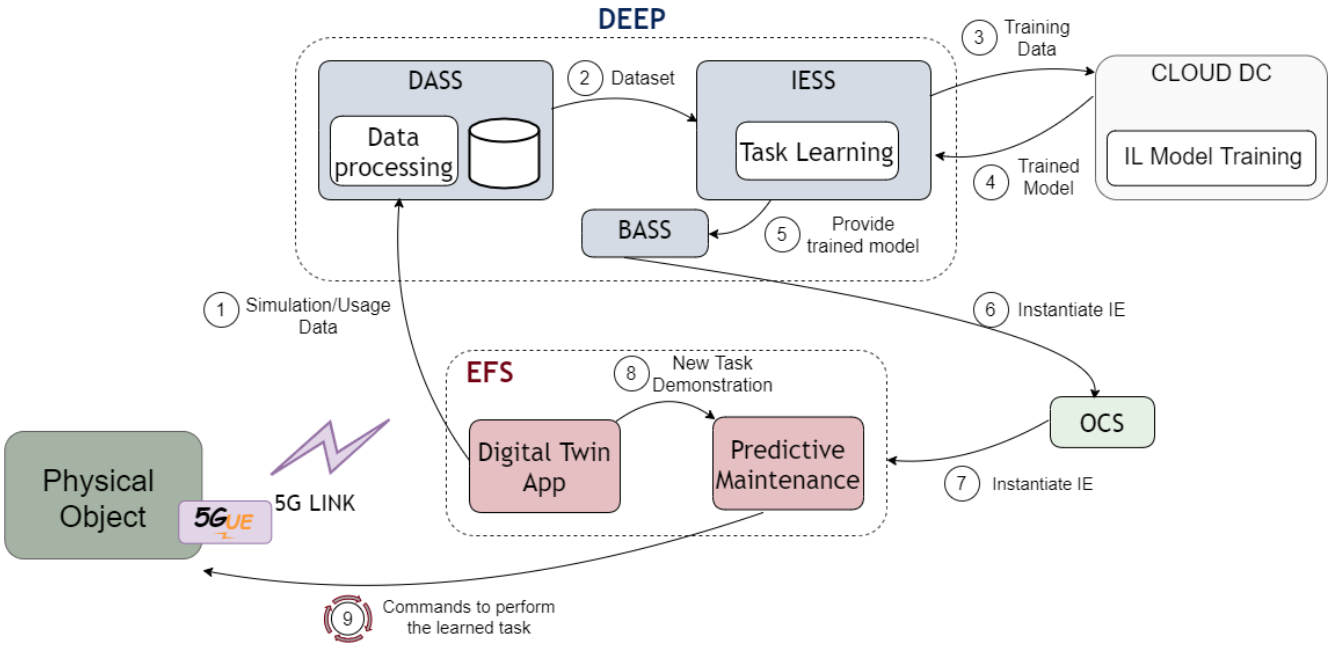


FIGURE 3-3 TASK LEARNING MODULE WORKFLOW.

The Task learning module aims at inferring to the Digital Twin the ability to perform tasks as a human will do without the human-in-the-loop. This module must perform correctly under different environmental conditions, meaning that providing the real object with a set of pre-registered navigation commands is not enough.

Mainly, two ML techniques are used to infer the ability to perform tasks to an agent, namely Imitation Learning (IL) and Reinforcement Learning (RL) as explained in Appendix 9. On one hand, IL is used to "teach" an agent to perform tasks by providing him demonstration (from an expert demonstrator). On the other hand, RL starts from a naïve agent that learns to perform the task using a trial and error approach. The agent in this case will learn how to achieve the goal only after thousands of failures. For this reason, among the two, IL is the safest and most efficient approach to infer human-like task-performing knowledge. Also, IL can be enhanced with domain randomization and meta learning techniques which will give the agent the ability to learn new tasks (from the same family of tasks) by only using one or few demonstrations.

In the case of the robotic manipulator, the demonstrations are composed of a set of images coupled with the joint state vectors of the robot. The required data can be gathered by the real usage of the Digital Twin App or generated through Digital Twin App simulations (step 1). In the first case, a camera is used to record the movement of the physical object and a mapping with the set of commands sent by the user to perform the task is needed. In the second case, not only the replica of the robot must be in

the simulations, but also a replica of the environment (objects, tables and lights). Also, to apply the domain randomization, the environment and the objects in it must provide domain-shift at each demonstration (changing texture, shape and position). The data can be sent as raw data to the DASS and then being processed after to produce datasets (higher granularity, need to pre-process the data before training), or directly sent by the Digital Twin App in bulk, forming a dataset (ready-to-use demonstrations stored in the DASS) (step 2).

Due to the high computation require, the IL agent training must be off-loaded to the cloud (step 3). The dataset will be used to find a generalized policy. A generalized policy is a meta-learning policy, obtained by using different tasks of the same family (e.g. pick-and-place task is a family of tasks, each different configuration of the environment is a task). The generalized policy found is then stored in the IESS (step 4).

Finally, when requested, the Task Learning module is instantiated in the EFS (step 5, 6 and 7). In order teach the real object to perform a new task, a new demonstration is needed (video of a human performing the action, 3D animated image from the twin) (step 8). The demonstration will be adapted to the generalized policy stored into the IESS and it will provide commands to send to the real entity to perform the task (step 9).

Risk reduction

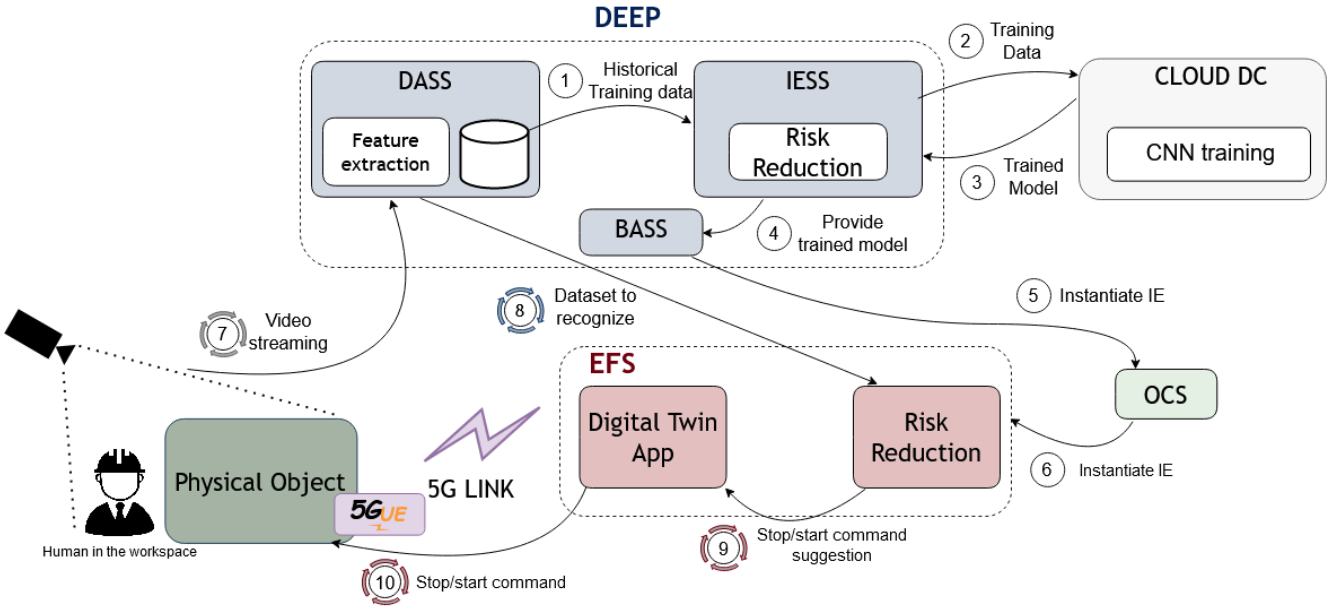


FIGURE 3-4 RISK REDUCTION MODULE WORKFLOW.

The risk reduction module is used for the adaptation of the functionality of the real object under various conditions. In the case of a robotic manipulator, one key example is the safety in human-robot collaboration. It is expected that in I4.0, human and robots will work together on different tasks. This

co-operation can lead to risks for human. The risk reduction can identify dangerous situations, via object recognition, and stop the robot pre-emptively.

In order to achieve this, Convolutional Neural Networks can be used to recognize objects. The CNN should be configured to achieve the highest accuracy possible, to avoid stopping the robot when no human is in the workspace (false positive). To do so, the two-stage approach should be used as presented in Appendix 9.

To achieve the desired precision, the CNN must be trained with an elevated number of training data. In this case, the training data is provided by a camera. The camera can be both RGB, to appreciate all the possible features of the environment (heavy dataset, high accuracy), and Depth, providing only information on the shapes (light datasets, less accuracy). The images and video are then stored to the DASS to be used for training the CNN.

The training of a CNN can be computationally intensive, showing the need for using a powerful cloud datacentre. The dataset is pre-processed by the DASS to produce training Batches (step 1) and then sent to the cloud for training (step 2). At each iteration, the accuracy of the training is evaluated, and the training stops when the desired accuracy is achieved. Finally, the trained model is stored in the IESS for future usage (step 3).

The trained model is requested to the IESS (step 4) and instantiated into an EFS application (step 5 and 6). New videos and images are then sent to the instantiated module to recognize if a human is present in the workspace (step 7 and 8). If it is the case, the risk reduction module sends a stop command to the robot, which lasts until the human leaves the workspace (step 9 and 10).

Predictive maintenance

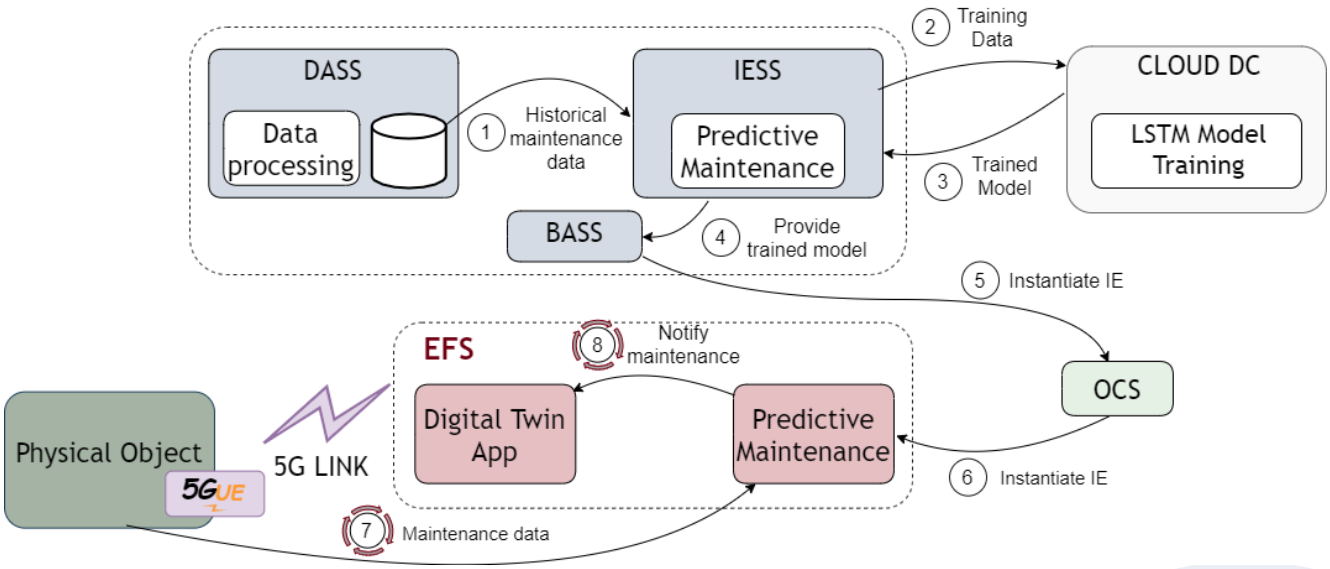


FIGURE 3-5: PREDICTIVE MAINTENANCE MODULE WORKFLOW.

The predictive maintenance module oversees the monitoring of the real object health status and notifications when critical conditions are predicted. Also, it can suggest (when possible) the maintenance actions needed to avoid disruption. In this way, it is possible to schedule maintenance operations pre-emptively.

For this purpose, Long-short term memory (LSTM) networks can be used. LSTM as explained in Appendix 9 are widely used for prediction of time series data based on a huge amount of historical data. Compared to a generic Recurrent Neural Network (RNN), the LSTM can keep memory of the dependencies among the input data for a longer time. In this way, it is possible to predict when the new input data is deviating from the normal behaviour.

Huge historical data is needed to train an LSTM network. Heterogeneous data sent from the real object (status of its mechanical components, heat, velocity), data from external IoT sensors and data from Digital Twin App simulations can be stored into the DASS. The data is then processed by the DASS to provide time-series datasets to the LSTM network.

The LSTM network is then trained off-line in the cloud, requesting all the historical data available from the DASS (step 2). Once the level of accuracy is achieved, the model is transferred to the IESS catalogue ready to be used (step 3).

Once the model is requested (step 4) and instantiated (step 5 and 6), it will continuously receive new data from the sources (step 7). The output of the prediction is then prompted as feedback to the Digital Twin App indicating the part of the hardware that needs maintenance (step 8).

3.2. I4.0UC2: Zero Defect Manufacturing Decision Support System

Increasing the throughput of production systems and reaching zero defect manufacturing processes is a key goal for all manufacturing companies. Through the solutions developed in 5G-DIVE project, a ZDM system can be facilitated and enhanced by the DEEP platform, helping the manufacturing companies achieve a greater production throughput with less wasted material.

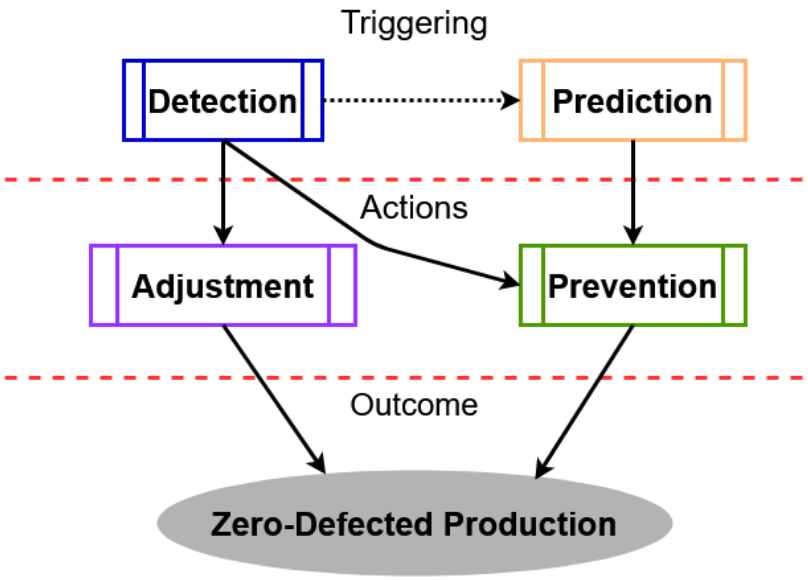


FIGURE 3-6: ZDM STRATEGIES.

One of the most promising strategies today is called Zero Defect Manufacturing (ZDM) that aims to decrease and mitigate failures within manufacturing processes to eliminate defected parts during production. Figure 3-6 illustrates four strategies of how ZDM can be implemented, namely: **detection, adjustment**, **prediction**, and **prevention.** Those strategies are interconnected together as follows. If a defect is **detected** then an **adjustment** can be performed (e.g., repair**,** remove**,** etc.**)**, also the data gathered by the **defect detection module** is populated to specifically designed models for **predicting** when a defect may occur and therefore be **prevented**. Defects are more likely to occur at several stages of the product manufacturing lifecycle, such as raw material transformation, manufacturing and assembly. Defects that are coming from manufacturing operations are not always avoidable. One of the reasons is that they are intrinsic to the operation itself. A single defect in the manufacturing life might also be the result of the accumulation of small defects that appeared in previous steps. If not treated or detected at an early stage of the manufacturing process life, defects continue to spread in the following life stages. The ZDM can be implemented in two different approaches. The product oriented ZDM and the process oriented ZDM. The difference is that a product-oriented ZDM studies the defects on the actual parts and tries to find a solution while on the other hand the process-oriented ZDM studies the defects of the manufacturing equipment and based on that can evaluate whether the manufactured products are good or not. In our ZDM use case, we will be focusing on the product oriented ZDM.

5G-DIVE ZDM solution includes three subsystems namely, EFS, OCS and DEEP, as follows:

- EFS: The adjustment and prevention models that execute the outcome of the ZDM system are represented as EFS entities and they integrate the main functionalities that interact directly with the machinery that is part of the manufacturing process (e.g., robots, production lines).
- OCS: As defined in [47], the OCS will compose, control, manage, and orchestrate the ZDM models. It will be responsible for fully automated management and operation of the distinct modules.
- DEEP platform:

- IESS: The detection and prediction models are represented as intelligent engines. While the detection modules focus on techniques that performed near-real time identification of defects, the prediction models power by historical data can prevent defect before they occur.
- DASS: the required data from the manufacturing companies will be collected and stored with the help of the DASS. This data will ensure proper work of the detection and prediction models.
- BASS: the auto-scaling feature will utilize the monitoring data from the DASS to perform horizontal scaling of the detection models. This part of the work will be developed in the next step of this work.

3.2.1. Reference Design

Figure 3-7 depicts an end-to-end block diagram of the 5G-DIVE ZDM solution. As shown in Figure 3-7, the manufacturing process on the most left-hand side includes a video camera system, a production line, and a robot arm manipulator.

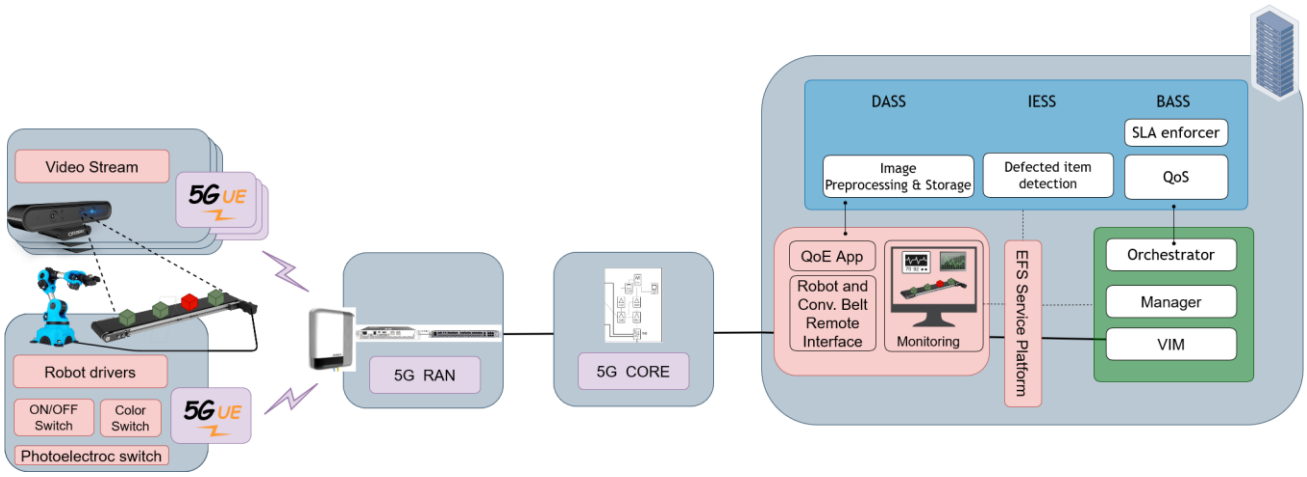


FIGURE 3-7: SYSTEM BLOCK DIAGRAM FOR ZDM.

The video stream latency (defined as the delay between the camera capturing an event and the event being processed by the ZDM system) sets the lower limit for **video-based defect detection**. Video stream encoding is performed close to the video source. There are two options that can be considered for defect detection. First option is to accelerate the defect detection close to where the video is produced. This helps reduce the network load as there is no need to transmit the video to the edge. The downside however is the need to have hardware accelerator close to the camera, which may increase the overall complexity and deployment cost. Second option is to transmit the video to the edge server which uses edge acceleration to perform video-based defect detection. This allows utilization of edge capabilities for the object detection. The selection of which option to use depends on many variable components such as available compute resources, defect detection latency requirements, power budget and overall system design.

The production line is composed of a set of sequential operations where components are assembled to make a finished product. The production line contains components such as, **photoelectric switch** to detect whether there are objects, **colour sensor** to categorize RGB objects that are conveyed or **on/off switch** to remotely control the production line. Additionally, the robot drivers directly interact with the robot hardware and together with the production line components they are responsible for: (i) making available sensor data and operational states; and (ii) executing instructions or navigation commands received from the **Robot and conveyor belt remote interface.**

To execute the outcome of the ZDM system, the detection and prediction models are realized as a **Robot and conveyor belt remote interface**. This interface interacts directly with the machinery that is part of the manufacturing process and performs basic conveyor belt control functionalities (e.g., starting and blocking the conveyor belt). Moreover, this interface is also in charge of robot manipulation to perform operations on the production line. On the other hand, the remote interface receives suggestions from the **QoE application** and the **defect detection**. The QoE application resides in the edge of the network and estimates if the quality of the video stream received at the edge is sufficient for performing **video-based defect detection.** If there is any problem with the video source, the QoE application instructs the remote interface to block the production line. Finally, the edge **monitoring** component is used by the remote worker and contains video presentation of the production process together with graphical representation of production performance metrics.

The ZDM system will exploit the **data lake pattern offered by the DASS**. A data lake is an architectural pattern of large data repository that allows storing of structured and unstructured data and is scalable based on user needs. Data Lake combines a large-scale storage repository with variety of high-performance processing engines that are usually virtualized. Large scale and independent scalability of processing and storage allows data lakes to store data as it is produced without pre-processing and processed it on-access. This approach allows flexible usage of produced data and it enables organizations run different types of analytics from dashboards and visualizations to big data processing, real-time analytics, and machine learning.

Data Lakes allow data-based access controls which enables multiple roles within the same organization or external organizations to gain access to a specific data for specific time. For instance, in one organization data scientists, data developers, and business analysts can access the same underlying data with their preferred tools and frameworks. This includes open source frameworks such as Apache Hadoop, Presto, and Apache Spark, and commercial products from data warehouse and business intelligence vendors. Data Lakes allows to run analytics without the need to move your data to a separate analytics system. The deployment of the data lake may be based on private or public cloud resulting that the data lake may be deployed either within operator network or external to operator network.

3.2.2. Analytics and Intelligence

Analytics and intelligence are using telemetry and video-based inference to provide insights from operations that are occurring in the smart factory network. The telemetry may include system state,

logs, configurations or other text / binary data. Some of the telemetry may be unstructured and stored as it has been received and other telemetry may be strictly based on standard models. Real-time telemetry can be a collection of measurements or other data in real-time at remote nodes and automatic transmission of the data. The nodes that are transmitting real-time telemetry are counterpart of telemetry deployed either at the edge or at cloud. Real-time telemetry may be consumed also non-real-time functions, for instance in data lake scenario.

Video analytics consist of inference performed on the video stream and analysis of those inference results. The video analytics are combined with the telemetry models in order to provide a specific, or overall view of the system allowing defect detection to perform actions that are intelligent in the case when faults occur or when collecting insights from the collected data.

Figure 3-8 depicts the workflow for the ZDM initial design for the intelligence-based defect detection module showing the key interactions amongst the different entities:

- Step 1: In a first step, the sensory (including camera) data together with measurements data are sent from various devices to the network via for example one or multiple 5G NR modems.
- Step 2: Some user data and network (including UE modem) telemetry data is passed to the EFS platform. The EFS includes the defect detection application responsible of the inference. It also includes a telemetry agent from the edge to collect the various telemetry data and aggregate it and pass it further up to the DASS for processing.
- Step 3: Various telemetry data from the applications, network functions and entities, and edge computing hosts, are passed on to the DASS for storage and pre-processing.
- Step 4: The telemetry data, after pre-processing, is presented to the IESS in the form of datasets for training or updating the training of different AI models.
- Step 5: At the IESS, the AI model is trained or retrained.
- Step 6: The new AI model, adapted to the actual telemetry data is then obtained.
- Step 7: The output AI model is passed on to the BASS for validation against QoS rules and adapted where necessary through interaction with the IESS.
- Step 8: The output AI model is passed on to the infrastructure orchestration for deployment.
- Step 9: The orchestration and control instantiate the inference algorithm in the edge platform and instructs for any update to the telemetry collection.
- Step 10: The inference is carried out by the defect detection application in the edge, and the result is passed to the control applications of the end devices including sensors, cameras and processors.
- Step 11: The various edge controller applications finally instruct the various devices to execute certain actions including change of configurations.

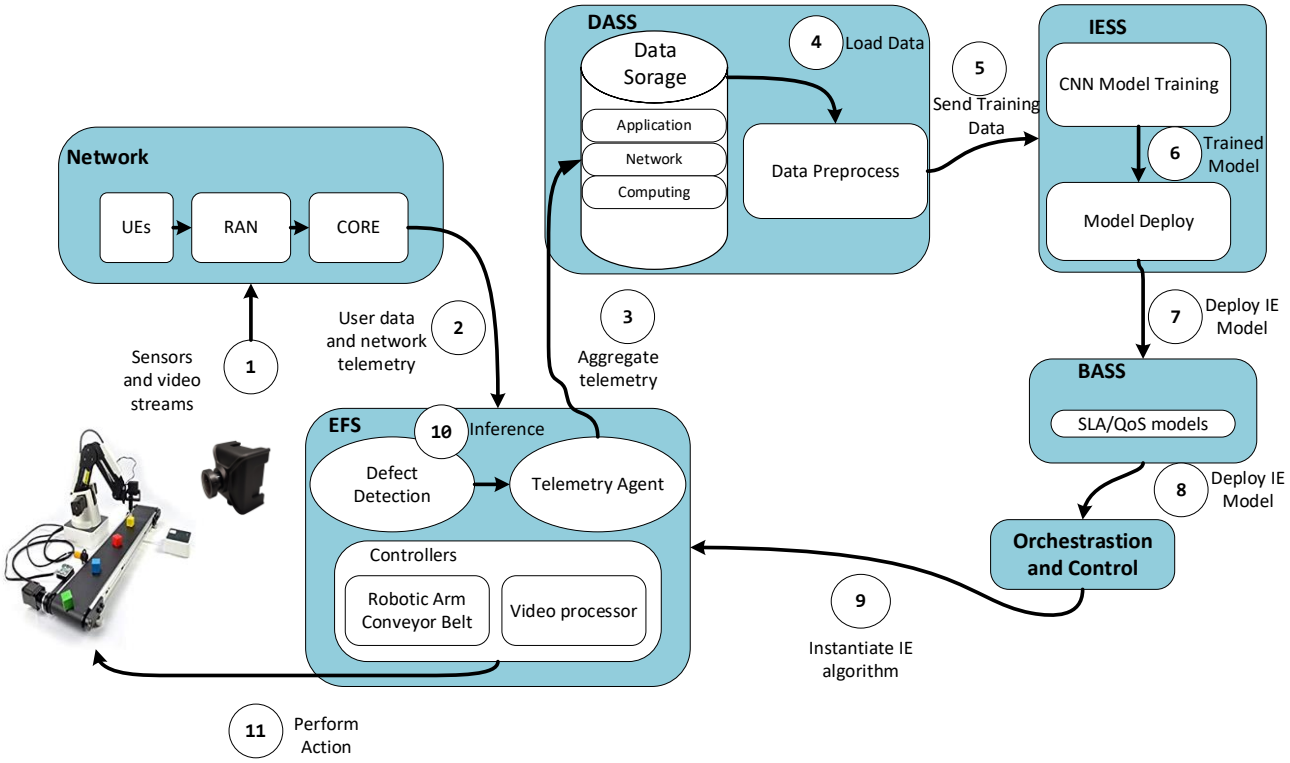


FIGURE 3-8: WORKFLOW FOR ZDM OBJECT DETECTION AND DEFECT RECOGNITION MODULE.

Object detection algorithm

The model used for Object Detection and Defect Detection in the EFS is based on Yolov3 algorithm originally trained with the COCO dataset with 80 classes of objects recognized. The algorithm "you only look once" (YOLO) does only one forward propagation of the image through the network, so that it can make predictions. After non-max suppression, it gives the name of the recognized object along with the bounding boxes around them. As a feature extractor, YOLO uses Darknet-53 which is trained with Imagenet classification dataset and is provided as an improvement to the previous versions of the algorithm, increasing this way the accuracy while keeping the real-time performance. YOLO design is built on Darknet, which is a GPU accelerated framework for designing and training deep neural networks. Darknet framework is implemented in C and CUDA. The model architecture is depicted in Figure 3-9 below.

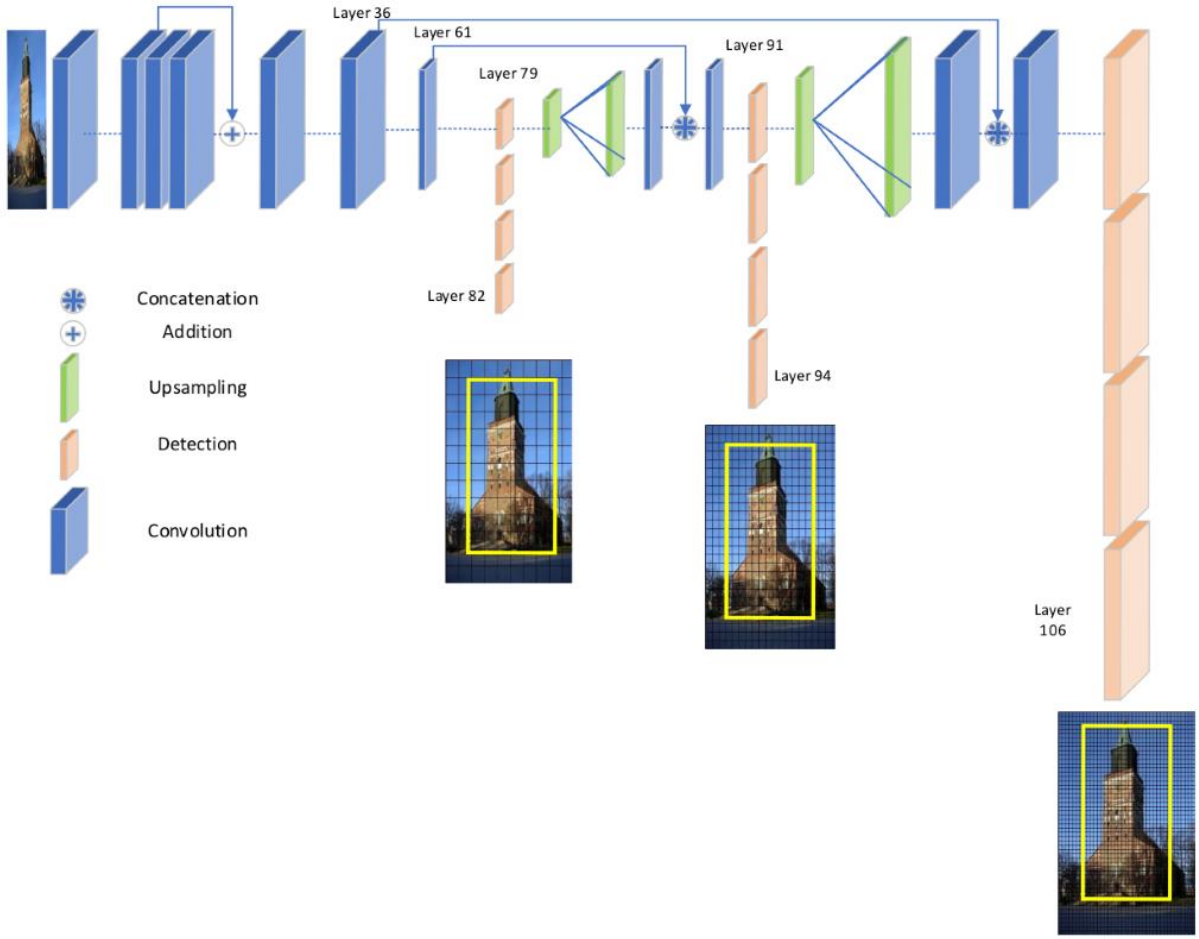


FIGURE 3-9: ZDM OBJECT-DETECTION MODULE.

The neural network uses multi-scale predictions, which means that detection phase is applied to different scales of the feature map. The input image is down-sampled three times by factors of 32,18, and 8. The first detection is made by layer 82, where up to that layer the image is down-sampled by the convolutional layers such that 81st layer has a stride of 32. If the input image is 416x416 then the resultant feature map is 13x13. One detection is made using 1x1 detection kernel, giving a detection feature map of 13x13x255. Then, from layer 79 the feature map in sent through convolutional layers before being up-sampled by a factor of 2. Then the feature map is concatenated with previous features from layer 61. The second detection is made at layer 94 yielding a feature map of 26x26x255. The final detection is done at layer 106, which produces a feature map of size 52x52x255. Detection at different layers helps in increasing the accuracy in detecting small objects. The up-samples concatenated with previous layers help in preserving the fine grain detection done before.

FPGA-based object detection

The YOLO design is built on Darknet, which is a GPU accelerated framework for designing and training deep neural networks. Where stringent energy consumption and processing latency are required, an alternative FPGA-based solution is considered. The key advantage of FPGA-based solution is the adaptable, fine-grained, and massively parallel nature of the compute and memory resources offered. This helps in enabling a wide range of Deep Neural Network Processing Unit (DPU) architectures that can take advantage of the multiple levels of parallelism mentioned and tailor to the specific requirements of a given DNN topology and the application-specific design constraints.

The diversity of DNNs is reflected in how much parallelism is available at each level mentioned earlier. Thus, any fixed hardware architecture that instantiates a fixed number of parallel compute elements communicating in a fixed fashion has limitations as to how efficiently it can execute a DNN. Especially considering the rapidly advancing techniques for creating efficient DNNs, adaptability is key to staying efficient in the changing DNN inference landscape.

In this context, the key advantage of FPGAs is the adaptable, fine-grained, massively parallel nature of the compute and memory resources offered. The heterogeneous devices facilitate a wide range of Deep Neural Network Processing Unit (DPU) architectures that can take advantage of the multiple levels of parallelism mentioned and tailor to the specific requirements of a given DNN topology and the application-specific design constraints. We consider here that:

- The FPGA can provide lower latency than the more traditionally used GPU platform for this type of application. This is due to the data path architecture of the FPGA and DPU, which does not require to first "flood" a large number of Streaming Multiprocessors (SM) as in a GPU.
- Taking advantage of the DPUs, we can reach higher throughput in terms of number of processed frames per second.
- The FPGA platform will provide a better energy efficiency solution compared to CPU and GPU based solutions.

Table 3-1 presents a preliminary comparison for three different KPIs namely E2E latency, power, and frame per second per Watt, for the object detection algorithm when run on GPU (GeForce RTX 2080 Ti) compared to when it is run on an FPGA (Xilinx ZCU102). These preliminary results show superior performance for the FPGA-based solution in all metrics most noticeably in terms of power consumption with a reduction factor of approximately 10 times less than the GPU-based solution. Further evaluation and recommendations for deployment will be reported in the next deliverable.

TABLE 3-1: PRELIMINARY MEASUREMENTS OF KPIS BETWEEN GPU AND FPGA SOLUTIONS.

		E2E_latency (FPS multithreaded)	Power(W)	FPS/W
	Yolov3_adas	5.72ms (175)	120	1.46
	Mobilenetv2_SSD	49.75ms (20)	120	0.167

GPU: GeForce RTX 2080 Ti	VGG16_SSD	22.1ms (45)	240	0.1875
FPGA: Xilinx ZCU102	Yolov3_adas	4.52ms (220)	17	12.94
	Mobilenetv2_SSD	16.372ms (61)	13	4.69
	VGG16_SSD	22ms (45)	22.5	2

3.3. I4.0UC3: Massive MTC

In this section, we first present the current system design of Massive MTC use case and how it fits to the 5G-DIVE architecture. Up to the time of writing this deliverable, some key components of the system design have been developed. In the following, we also provide more implementation details and evaluation results of different components.

3.3.1. Reference Design

Figure 3-10 shows the block-diagram of Massive MTC system design and its mapping to 5G-DIVE architecture which comprises DEEP platform, EFS functions/applications, EFS service platform and OCS.

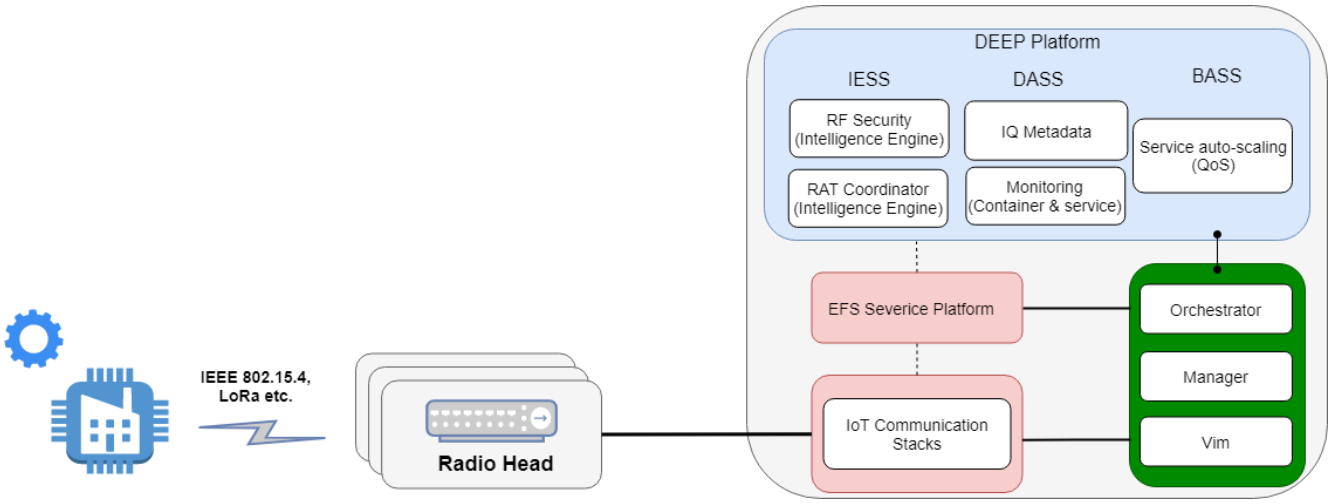


FIGURE 3-10: SYSTEM BLOCK DIAGRAM OF MASSIVE IOT USE CASE.

The following lists the components in the 5G-DIVE Massive MTC solution:

- EFS functions: IoT communication stack functions of LoRa and IEEE 802.15.4 have been developed. In 5G-DIVE, we will further investigate the Cloud native design possibilities of the IoT communication stack, breaking the communication stacks into smaller functions which can

be scaled and managed independently. This perspective together with resource pooling of virtualized RAN (vRAN) for IoT will be addressed later in this deliverable.

- EFS service platform: MQTT is currently used in the EFS service platform to provide pub/sub mechanism for data transport and sharing between different components. It is under discussion to later adopt Zenoh developed in 5G-DIVE.
- OCS: the plan is to first use Kubernetes as the framework for OCS before the required orchestration features are provided by Fog05.
- DEEP platform
    - IESS: two intelligence engines for RF security and RAT coordination will be developed in 5G-DIVE. More details about RF security will be provided in the next section.
    - DASS: IQ data will be collected from IoT communications stack functions and stored in DASS. The stored IQ data are used by the intelligence engines to improve security and mitigate interferences between RATs on the same band. Furthermore, DASS will also handle the container and IoT service level monitoring which will be used for the service auto-scaling in BASS increase the system scalability and achieve vRAN resource pooling.
    - BASS: the service auto-scaling will utilize the monitored data from DASS to manage the horizontal scaling of communication stack functions through the interface towards the orchestrator. This part of the work will be developed in next step after this deliverable.

3.3.2. Key Design Components

This section presents the progress of the solution design on two tracks: i) Cloud native design; and ii) the intelligence engine for RF security. Section 3.3.2.1 starts by presenting the concept of vRAN for Massive MTC networks and analysing the resource pooling behaviour to provide some insights on the impact to resource auto-scaling. To study further the resource pooling, we have been developing an emulation testbed which can be used to evaluate a large-scale vRAN IoT/MTC networks and help the design for service auto-scaling algorithms which will be addressed in next step of this project. In section 3.3.2.2, we present our first design of the emulation testbed and the emulation results. It shows that the testbed is able to support the next-step study for auto-scaling. Regarding the Cloud native design for IoT communication stack function, section 3.3.2.3 provides an in-depth analysis for IEEE 802.15.4 as an example. In section 3.3.2.4, the intelligence engine for RF security is presented and the evaluation results are provided, showing the possibility to improve security with AI techniques utilizing the IQ data.

3.3.2.1. Virtualized RAN and resource pooling for massive IoT networks

Cloud native is an approach for building applications as micro-services and running them on a containerised and dynamically orchestrated platforms that fully exploits the advantages of the cloud computing model [51][52]. In vRAN system, communication stacks are implemented in software (instead of in HW), which are virtualized and orchestrated in a Cloud environment, e.g. at the Edge [53]. Such a design fits well to future IoT networks to cope with the required flexibility and scalability.

The Cloud (or Edge Cloud) resources are pooled among many cells. With an auto-scaling feature, the utilized resources would scale dynamically with the actual traffic load.

Figure 3-11 shows an example of a vRAN setup for IoT networks. The system comprises of four parts: remote radio heads (RRHs), fronthaul network, Edge Server/Cloud and Cloud. The RRHs are in charge of transmitting and receiving radio signals. It is connected via a fronthaul network (e.g. Ethernet or IP networks) to the communication stack functions, running in an Edge Cloud. Each communication stack is in charge of lower layer (e.g. PHY layer) processing like demodulating/demodulating radio signals, as well as higher layer processing like MAC and transport layers. In this work, we focus PHY layer processing. Then the IoT data will be processed in the IoT applications located in Cloud.

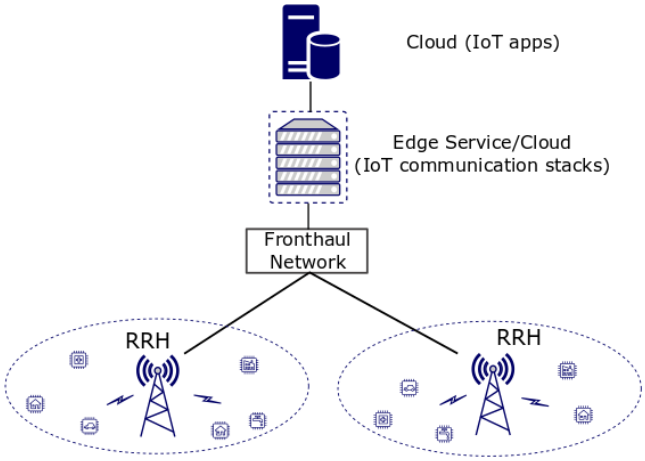


FIGURE 3-11: SCHEMATIC DIAGRAM OF A TYPICAL VRAN NETWORK.

In this way, the radio processing is centralized and cloudified in an Edge Cloud infrastructure. As described previously, the processing resource in Edge Cloud can be pooled to process the IoT traffic from all RRHs (cells) connected. This would make resource utilization much more efficient, which can dynamically scale with the actual traffic load. Especially when traffic load is unevenly distributed among cells, the resource saving would be significant. The saved resources in the Edge Cloud can be released to other Edge clouds. Such approach is referred to as resource pooling, which is considered as one of the key advantages of vRAN systems to enable significant savings on the costs of HW resources.

To elaborate the resource pooling effects, we illustrate how multiple cells can be processed in common computing resources in a time-multiplexing manner in Figure 3-12.

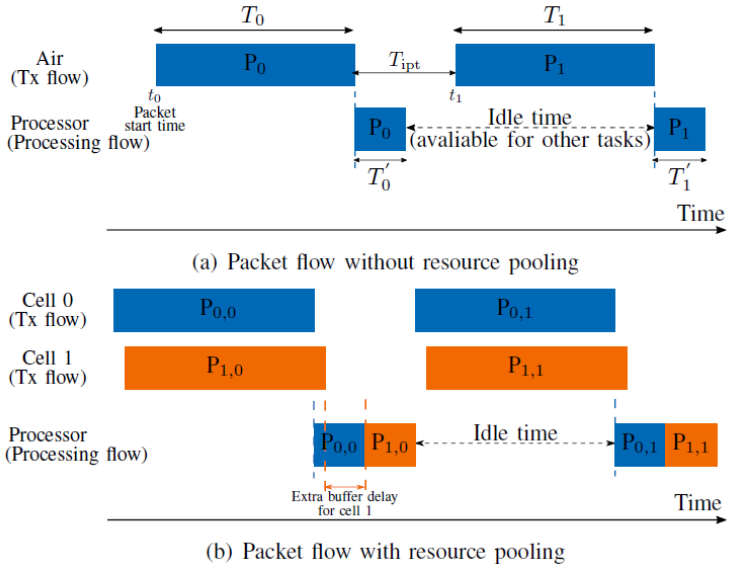


FIGURE 3-12: ILLUSTRATION OF RESOURCE POOLING.

In Figure 3-12(a), we first consider the situation when there is no resource pooling on the processor, i.e. one cell processed by one communication stack function. We assume the air transmission time of the i-th packet Pi is T_i. T_{ipt} is the packet interval time between packets P_0 and P_1 in this example. On processor side, we assume the processing starts as the whole packet data arrives, with processing time T_i' for the ith packet. For IoT stack processing, the powerful CPUs used in Edge Cloud servers can process very fast. Therefore, the transmission time T_i is much longer than the processing time T_i', and the idle time between two packets processing can be used to process the packets from other cells. Otherwise, the CPU time would be wasted. Therefore, it shows the possibility that multiple cells can be processed in common computing resources in a time-multiplexing manner, which is referred to as resource pooling as described previously.

Figure 3-12(b) shows the resource pooling functionality with two cells processed by one instance of the communication stack function. In this example, we illustrate the scenario when the packets from two cells overlap in time, i.e. packets collide over time, which leads to the demand for buffering the latter packet. It is shown that the resource utilization on the processor side increases since we have less idle time. Also, extra buffer delay is introduced due to the packet collision.

The simple example above shows that resource pooling can increase the resource utilization but can also cause extra packet delay. Thus, for more complicated and large-scale IoT networks, it is important to understand the resource pooling behavior, which would help design auto-scaling algorithms to optimize resource pooling, e.g. maximizing resource utilization while keeping acceptable latency.

3.3.2.2. Emulation Testbed for Large-scale vRAN IoT networks

To be able to study the resource pooling behaviour and develop auto-scaling algorithms in 5G-DIVE, we focus on developing a large-scale emulation testbed, using LoRa stack [54] as an example, which enables to emulate resource pooling effects.

Emulation testbed design

In this work, we focus on investigating the resource pooling effects with one instance of IoT communication stack function to process multiple cells. Figure 3-13 illustrates an emulation testbed design where the uplink signals from multiple cells are aggregated to be processed by one instance of the communication stack. The packet generators (implemented in software) creates the IQ sample streams of multiple cells and send them to the Edge. Then, a packet aggregator is developed to aggregate the packets to a single IQ sample flow, which would be processed by one instance of the IoT communication stack function. In the following, we describe more details regarding the implementation of different components.

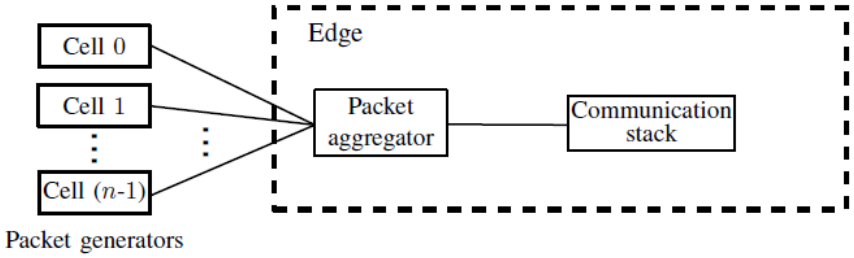


FIGURE 3-13: ILLUSTRATION OF THE EMULATION TESTBED DESIGN.

Packet generator

A packet generator creates IQ samples corresponding to the packets following a certain traffic pattern of an individual cell. Multiple packet generators are used to generate multiple cells. This makes it an effective HW-free tool for enabling large-scale network testing to study resource pooling and the performance. Using packet generators, each stream can be easily configured with various attributes e.g. traffic pattern (e.g. periodic, Poisson distribution, Gaussian distribution), packet length (payload length), etc. It dramatically improves the flexibility and scalability of the packet generation part of the system compared with conventional HW-based system, which would require real network nodes or devices.

Packet aggregation

Packet aggregation is a software function of aggregating multiple IQ sample packet streams together into a single IQ stream to be processed in one instance of communication stack function. Figure 3-14 schematically depicts the packet aggregation function with a two-cell flow. To enable the functionality, we need to design a scheduling function to allow us to dynamically select input from different cells and aggregate the packets with a first-in, first-out (FIFO) buffer. We emphasize below two techniques that are used to manage these features:

- *Round-Robin scheduling*: Round-Robin scheduling is popular for its simplicity to implement and fast switching characteristics. A round-robin scheduler generally employs time-sharing to schedule processes fairly i.e., handling all processes without priority. For packet aggregator, scheduling decisions are made according to the Round-Robin scheduling policy, then a packet would be appended into the buffer queue when the whole packet is received.

- *Circular buffer*: circular buffer (or ring buffer) is a certain type of queue with a FIFO structure. The main advantage of a circular buffer is that each time a new element is added, we just need to store two pointers to the front and back of the structure instead of to have its whole packet elements shuffled around. With a fixed circular buffer size, a packet would drop if not been processed when next packet arrives if the buffer is full.

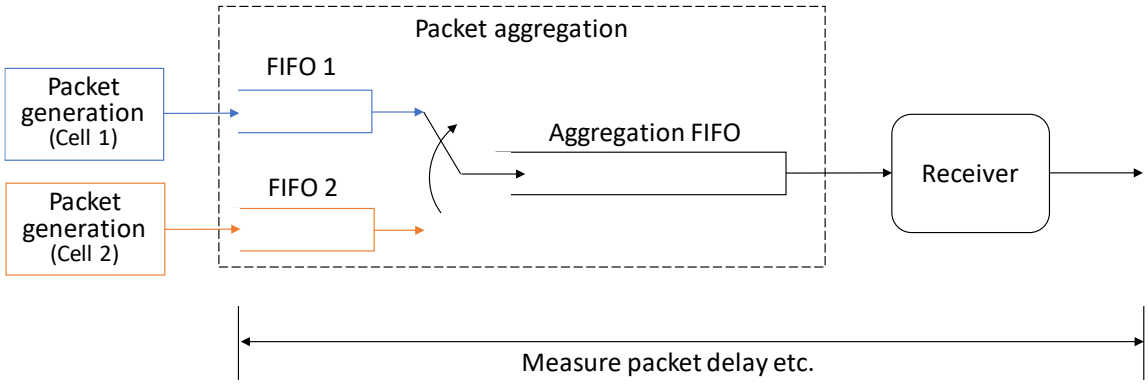


FIGURE 3-14: ILLUSTRATION OF PACKET AGGREGATION FOR TWO CELLS.

Communication stack function

In this work, we use LoRa as one IoT example for communication stack implementation. LoRa is a low-power wide-area network (LPWAN) technology using a unique chirp spread spectrum (CSS) modulation that modulates data onto chirps. It uses a multi-stage encoding pipeline that includes Hamming coding, interleaving, data whitening, and gray encoding of symbols. For implementation, we use an open source implementation of the LoRa PHY[55], to decode the LoRa packets. Performance of the decoder has been evaluated in our previous work[51].

Testbed platform

As the first step for validating the basic design, we currently run the whole testbed in one process, which would affect the resource pooling gain, as all functions share the same CPU resources. In later steps, we will split them in different processes or containers if virtualized. An example of two-cell implementation is illustrated in Figure 3-15.

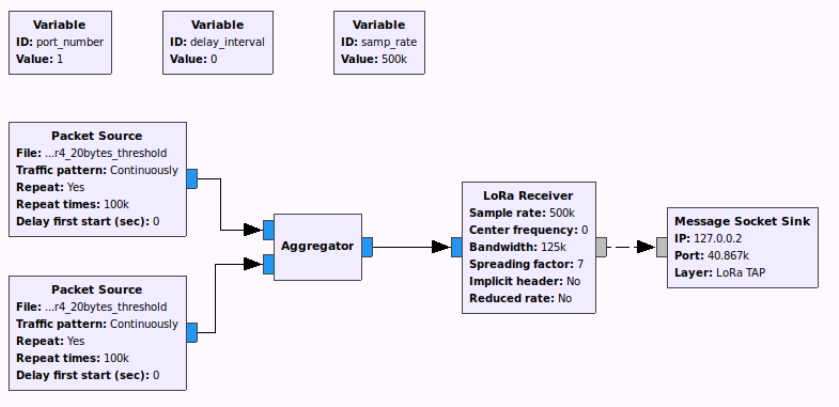


FIGURE 3-15: EXAMPLE OF GNURADIO IMPLEMENTATION FOR TWO CELLS.

Our emulation testbed comprises one Intel NUC mini-PC with a 4-core i7-6770HQ CPU@2.6GHz (Turbo to 3.5 GHz), 16GB RAM and 500GB SSD. The mini-PC runs GNURadio 3.7.9.3 environment[58], which contains software blocks with aforementioned functions: packet generator blocks used to send pre-saved LoRa IQ samples; an aggregator block which aggregates packets received from multiple packet generator blocks representing different cells; a LoRa decoder to decoder the IQ samples of LoRa packets and a socket sink to save the data. Besides, for comparison purpose, we deploy a HW-based end-to-end testbed as introduced in our previous work[57], HW components are depicted in Figure 3-16. PC runs a Python program to control IoT device (Fipy node [60] used in this work) to send LoRa packets. NUC mini-PC is connected with a USRP SDR kit [59] via USB3.0 and runs the GNURadio implementation of the LoRa decoder. To avoid air interference, IoT device and USRP are connected with a coaxial cable.

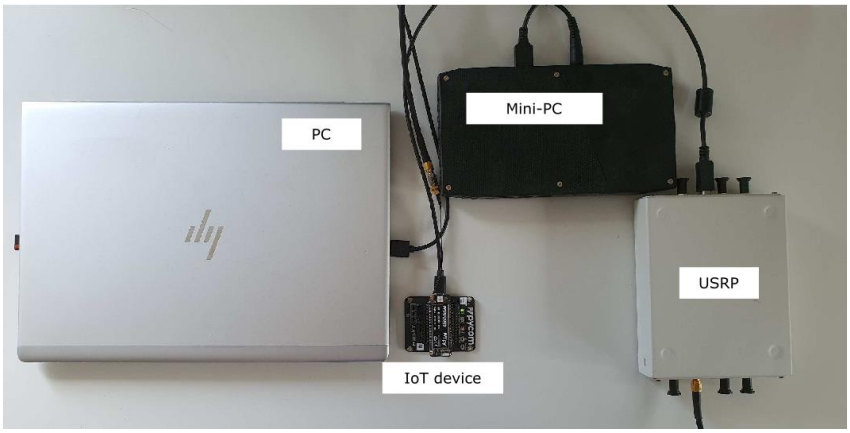


FIGURE 3-16: PHOTOGRAPH OF TESTBED SETUP WITH ONE IOT DEVICE.

In the measurements, we sent LoRa packets with spreading factor = 7, coding rate = 4/8, bandwidth = 125KHz, payload length = 20 bytes, and sampling rate = 500KHz. The number of IQ samples captured with a threshold of -60 dB is 40000, resulting a 320KB binary file, which is used for the packet generator

to generate the IQ sample stream according a certain traffic pattern. With these configurations, the transmitted LoRa packet on air time is 73.98ms, which means the maximal achievable packet rate is 127 packets per 10 seconds i.e. the traffic of a full-load cell. It is worth mentioning that in all measurements, we lock the CPU frequency to avoid the frequency scaling in Ubuntu system. Specially, for validation tests, we lock the CPU frequency to 1.2GHz for a better comparison between different traffic loads. In multi-cell measurements, we lock the CPU frequency to 3.2GHz to achieve the best performance of the system. Regarding the resource utilization for each measurement, resource usage is measured 50 times with 1s interval between each measurement and average values are presented.

Measurement results

To evaluate our testbed performance, we conduct three different measurements:

1. Verification of packet generator.
2. System resource usage and packet delay with multiple full-load-traffic cells.
3. Maximum number of cells supported with partial-traffic cells.

Verification of packet generator

To verify the developed packet generator can achieve the similar resource pooling effects as real IoT devices, we first perform the tests of the one-cell scenario to compare with the results obtained from the real test setup with a real IoT device, as shown in Figure 3-16.

Figure 3-17 compares the packet traffic pattern generated by IoT device and packet generator with LoRa payload length equals to 13 bytes in a time span of 0.4 s. Since no strict timing to control the sample rates in a PC environment, the data transmission speed depends on the scheduling of Linux, thus leading the much shorter transmission time compared with IoT device. We also assume that a packet starts to be processed when the whole packet arrives. Then, we try to align the end point of the packets to achieve comparable processing time. It reveals that by implementing packet generators, we are able to produce the same packet traffic pattern as real IoT devices, from the processor perspective.

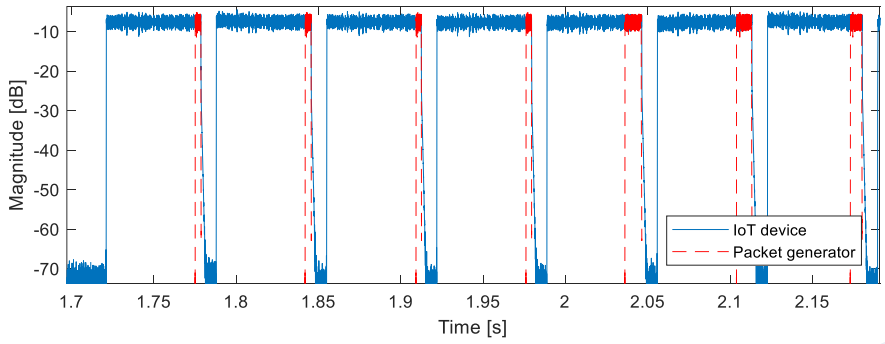


FIGURE 3-17: COMPARISON OF PACKET TRAFFIC PATTERN GENERATED BY IOT.

To further prove that similar processing flow is obtained with packet generator. Figure 3-18 illustrates the average CPU utilization of LoRa decoder function for two set of measurements. We conduct measurements by sending periodic packets with various traffic loads. We can observe that for both measurements, CPU utilization increases as the traffic load increases. Moreover, for all traffic loads, the CPU difference between the two methods is less than 0.1%. Thus, we claim that we can successfully emulate IoT devices with implementing packet generators.

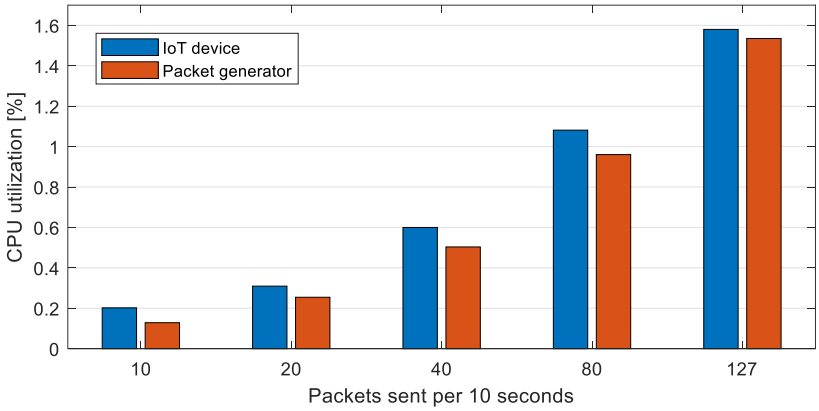
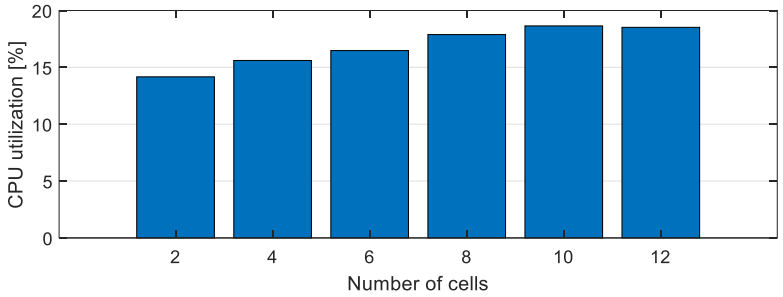


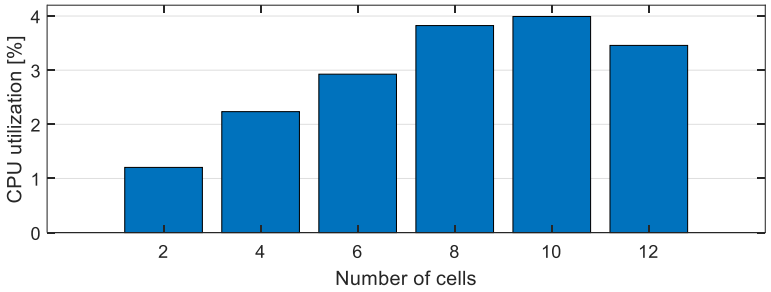
FIGURE 3-18: COMPARISON OF CPU UTILIZATION FOR DECODING.

System resource usage and packet delay with multiple full-load traffic cells

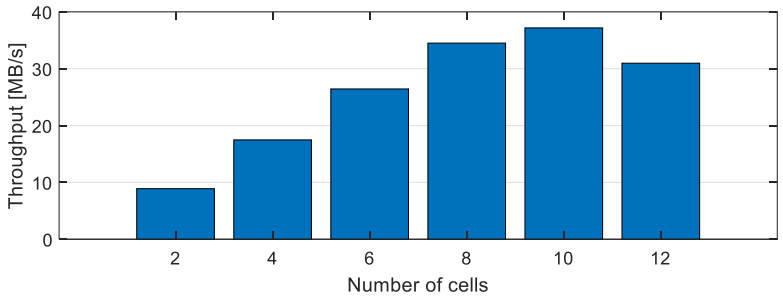
We then extend our emulation testbed with multiple cells represented by more packet generators connected to the aggregator, to study the resource pooling behaviour. In this measurement, each packet generator sends full-load cell traffic, i.e. 127 packets per 10 seconds. Figure 3-19(a) and Figure 3-19(b) shows the CPU utilization of whole emulation testbed process and of LoRa decoder only, respectively. For the whole emulation testbed process, the CPU utilization first grows as we add more cells to the system. A bottleneck around 19% CPU utilization is observed with cell number equals to 10. The restriction is due to the system CPU policy, in other words, we have a four-core CPU and with one program running on it, system would assign the thread to an arbitrary core, leading the CPU consumption no larger than 1/4 = 25%. Similarly, for LoRa decoding function, CPU utilization also increases at first and reaches the maximal value with 10 cells. However, a drop of CPU utilization is found when we further add cells to the system. The reason for this is that packet generation is implemented in the same process as the decoder. Thus, adding more cells will take more CPU resources, while the whole CPU utilization reaches the bottleneck and will not increase, which results the decrease of CPU utilization for LoRa decoding function. Regarding the throughput of the system as shown in Figure 3-19(c), we find exactly same trends compared with Figure 3-19(b). It proves that the throughput of the system depends on the CPU resources allocated to the LoRa decoder function, i.e., if more CPU resources are allocated to LoRa decoder, we would achieve a higher throughput.



(a) CPU utilization of whole emulation testbed process



(b) CPU usage of LoRa decoder only



(c) System throughput

FIGURE 3-19: SYSTEM RESOURCE UTILIZATION WITH DIFFERENT NUMBER OF CELLS.

Figure 3-20 shows the memory utilization for LoRa communication stack with data captured for 50 seconds. From the figure, it is obvious that when cell number equals to 10 and 12, the buffering size keeps increasing and the aggregator block fails to handle such traffic loads. Therefore, in the current implementation, 10 full-load cells can already cause overflow and thus the number of full-load cells is limited to 9. As mentioned before, in the future work, we will separate different functions to different processes or containers. More CPU resources will be allocated to LoRa decoder and more full-load cells can be supported.

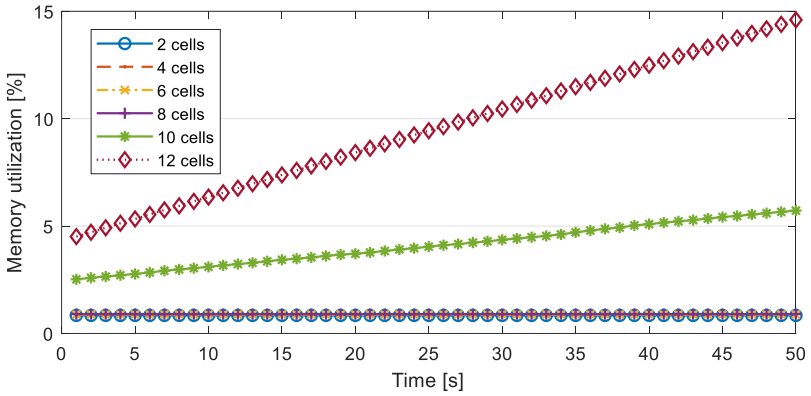


FIGURE 3-20: MEMORY UTILIZATION FOR WHOLE EMULATION TESTBED PROCESS.

Figure 3-21 compares the packet delay due to resource pooling for one instance of communication stack function to process multiple cells as described in Section II. We consider four cases with cell numbers equal to [2, 4, 6, 8]. The results confirm that pooling more cells increases the packet delay. It shows that the designed testbed is able to evaluate the packet latency of a vRAN system.

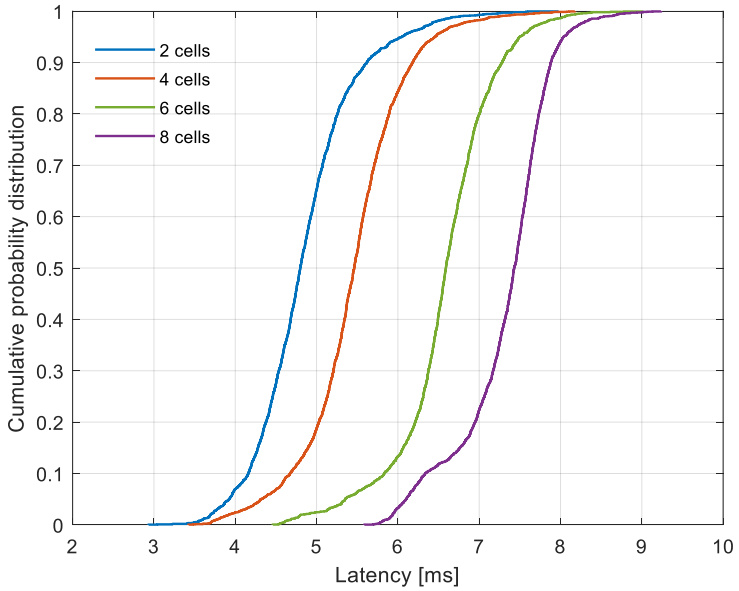


FIGURE 3-21: PACKET DELAY STATISTICS WITH DIFFERENT NUMBER OF CELLS.

Maximum number of cells supported with partial-traffic cells

To further showcase the resource pooling effects when cells are less loaded, we then reduce the cell traffic load and repeat the measurement steps. The maximum number of cells is measured until the system is overloaded as presented before. Table 3-2 shows that more cells can be pooled when traffic load is lower.

TABLE 3-2: MAXIMUM NUMBER OF CELLS SUPPORTED WITH 20-BYTE LORA.

Traffic pattern **Packets sent per 10 seconds**	10	20	40	80	127
Maximum cells supported	40	27	18	12	9

3.3.2.3. Cloud-native design considerations: example of IEEE 802.15.4 stack

Cloud-native design is effective in addressing these needs for cloud-based distributed services. In cloud-native design, services are designed as a combination of microservices that interact through well-defined interfaces. This enables the orchestrator to provide demand-based scaling of the microservices. Looking from a deployment point of view, the modularity of microservices helps in the agile development and stability of the system. These microservices are containerized enabling flexible deployment independent of the underlying infrastructure.

We incorporate cloud-native design methodologies to the development of our virtualized IoT network stack for IEEE 802.15.4 following these principles:

- We architect the IoT communication stack as loosely coupled microservices, where the functionality of each microservices is limited to a single well-defined network layer.
- These microservices expose these functionalities over well -defined Application Programming Interfaces (APIs).
- Each microservice is inherently stateless in the deployment environment. These ensure demand-based horizontal and vertical scaling.
- The microservices can be configured using auto-configuration enabling zero-touch deployment. This can be implemented using environmental variables or using an external configuration utility.

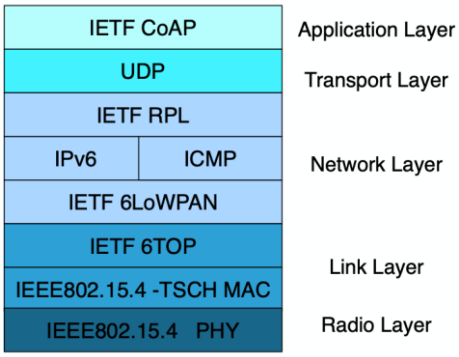


FIGURE 3-22: IEEE 802.15.4 NETWORK STACK.

Figure 3-22 shows our current IEEE 802.15.4 network stack, we investigate the applicability of cloud-native methodologies to each network layer.

Physical Layer: The physical layer does the modulation and demodulation of packets to baseband samples and vice-versa. The physical layer functions are independent of the previous service calls

hence, it is stateless both in the transmit chain and receive chains. We can adapt cloud-native methodologies in our design of physical layer functions by adding external configuration specifying the endpoint it is communicating with and the RF hardware features needed by the function.

Medium Access Control (MAC) Layer: The MAC layer manages access to the medium (air) for incoming and outgoing packets. The MAC layer has internal states of Acknowledgement, retransmission, and sequence number. When sharing a single wireless channel among multiple MAC layers, these internal states make the MAC layer stateful. Since the computation load of a single MAC layer is much lower compared to a PHY layer, in our use case, we envision a single MAC layer coordinating multiple PHY layers. In this case, the MAC layers become stateless. In the case of horizontal scaling with shared channel control, the internal state (sequence number) needs to be shared between multiple instances.

Network Layer: The network layer provides services to enable the exchange of data across the network. It provides addressing for identifying each node in the network with an IP address. Routing functionality directs the packets to the destination node on the same network or different networks. The routing and addressing functionality for the same network needs to be consistent across multiple instances of the network service, hence it is not stateless. To apply cloud-native methodologies in the design of the network layer, a bootstrapping service needs to be created. The bootstrapping service would perform the migration of node context from the current network service instance. The addressing scheme and routing information from the routing table would need to be shared between multiple instances. The bootstrapping service would also include the initial configuration of the service which binds the network function to one or more MAC services.

Transport Layer: The transport layer, manages connections for different applications and services. TCP and UDP are the most commonly used transport layer functions. TCP manages state for every connection, hence not stateless. To provide horizontal scaling functionality, every new transport layer instance we need to transfer connection context for the migrating connections.

In the previous 5G-Coral project[50], we developed our multi-channel and multi-RAT IoT gateway. Our implementation was limited by having several separate network stacks for each radio channel, leading to higher resource usage. Since all the devices on different physical radio channels were not part of the same network, hence we could not support multi-channel networking functions. Radio channel capability limits the number of IoT nodes that can be supported by a single gateway. The ability to allow multiple devices to be part of the same network allows us to scale the number of devices supported by our IoT gateway.

With the incorporation of cloud-native methodologies in our IEEE 802.15.4 Network Stack, we split up our earlier monolithic container two layers: one for the physical layer and the other for the complete network stack as shown in Figure 3-23.

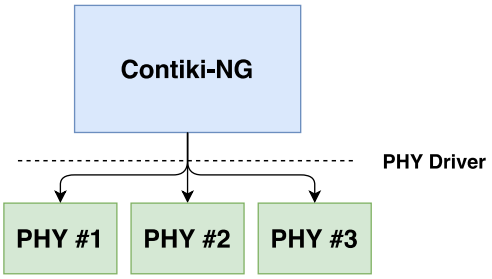


FIGURE 3-23: IEEE 802.15.4 CLOUD NATIVE IMPLEMENTATION.

PHY Layer: The PHY layer, provides the radio technology services and connects to the PHY driver over UDP links. The PHY layer on initialization provides a configuration and bootstrapping function, which registers the PHY layer services to the PHY driver. The initial configuration messages are used by the PHY driver to identify each instance of the PHY layer.

PHY Driver: The PHY driver, manages the PHY instances and is responsible for tagging uplink and uplink packets approximately. It translates the identity of the PHY instance into attributes that are then used by the Contiki-NG network stack

Contiki-NG: It provides the complete network stack for the IEEE 802.15.4 with an LWM2M application layer. The attributes of the PHY layer messages are propagated to the network layer. The routing service manages a lookup table to find the physical layer for each incoming message from the application. This enables the applications and transport layer to function without knowledge of the underlying physical layer instance.

3.3.2.4. Intelligent Engine for Radio Security

In a dense deployment of low-power sensor nodes, we need to ensure secure connectivity to the devices to preserve data integrity and the correct operation of the factory operations. The traditional way of ensuring secure connectivity on IoT nodes is to use digital certificates-based authentication. But the constrained nature of the IoT nodes limits the use of complex cryptographic solutions. This limitation makes the nodes vulnerable to forging and replay attacks.

The main problem we need to address is the identity problem. How can we find the identity of the node sending the message? As shown in Figure 3-24, two nodes are communicating over the wireless link to the IoT Hub/Gateway. The malicious node has forged the certificate of Node 1. In this case, how do we verify it is Node 1 which is transferring data to the IoT Hub. We need to design an identity, which is unique to every node and it should not be duplicatable.

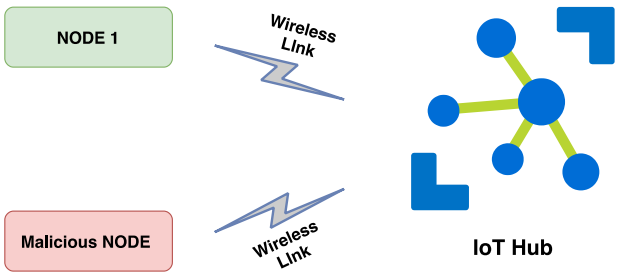


FIGURE 3-24: RF SECURITY PROBLEM.

One of the most unique identities already present in every device is the hardware impairments in the RF transmitter chain. This ensures, that the transmitted signal has unique characteristics for every node. It is very hard to duplicate since it arises from hardware impairments. In a standard RF transmitter, the local oscillator and mixers impart phase noise and slight deviations in the transmitted signal. This creates a unique signature which we refer to as RF fingerprints.

In our virtualized IoT gateways, we provide the wireless IQ signals through the IQ Metadata service. Using the IESS, we can train a deep learning model to identify the RF fingerprint present in the radio signals from different IoT nodes. This model provides an added layer of security on top of digital certificates. With this, we can enable a two-factor authentication system. Our proposed system is shown in Figure 3-25.

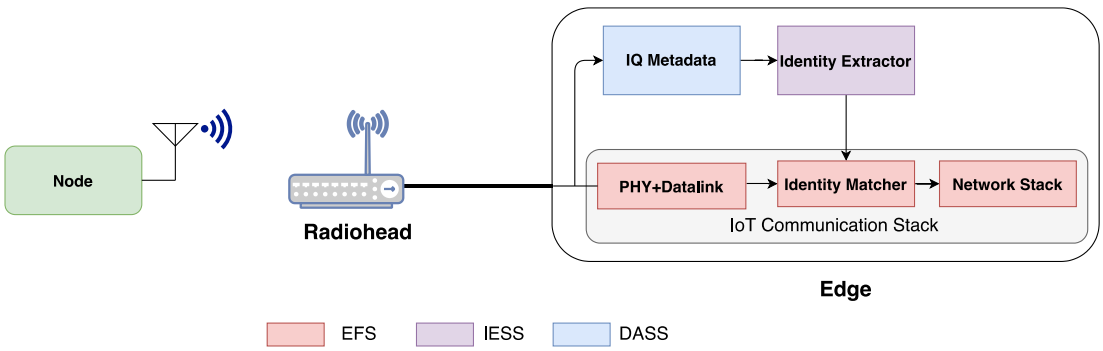


FIGURE 3-25: RF SECURITY SYSTEM.

The deep learning model in the IESS is used to extract the RF fingerprints which are then used in the IoT Communication Stack to ensure the authenticity of the signal. If the authenticity is verified, we pass the packets to the Network Stack. The network stack provides digital certificate-based security.

Data Collection

We collect data for training our IESS deep learning model using the setup shown in Figure 3-26. We program the Zolertia firefly nodes to transmit periodic network association packets, which the gateways would use to fingerprint the devices. The USRP captures the radio signals at the gateway with a sampling frequency of 4 Mega Symbols per Second (Msps). Since we are using a low sampling rate, we learn the steady-state characteristics of the RF signals. To enable the deep learning model to learn only the inherent characteristics of the signals and not any embedded information in the packets, we set the same MAC ID across all nodes. Furthermore, to remove the receiver bias, we use DC offset correction and IQ imbalance correction on the USRP.

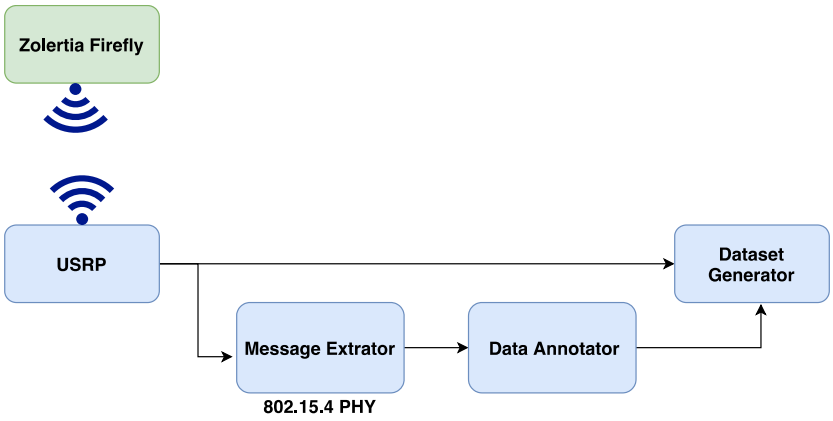


FIGURE 3-26: DATA COLLECTION SETUP.

The USRP streams the radio samples to the dataset generator which creates a SigMF data file. To extract the messages, we also create a metadata file that states the starting and ending sample index for a message. We use our PHY layers to extract the starting and ending index, which the data annotator converts to metadata annotations following the SigMF standard. For each node, we collect 100 messages.

IESS Deep Learning Model Training

For robust learning, we do data preprocessing on the dataset. Our deep learning pipeline does data normalization and sliding window operations on the dataset. The normalization helps the learning to be robust against amplitude variations. The sliding window operations enhance shift-invariance of features and also helps in data augmentation, creating multiple examples from a single message.

We use time-series classification methods for learning the device signature. The time series is segmented into 128 samples chunk. This allows our deep learning model to learn features over multiple IEEE 802.15.4 and transitions between multiple symbols. We adopt an experimental approach to selecting deep learning architecture. Convolutional Neural Network (CNN) is good at extracting features but lacks learning sequential features. Whereas Long Short Term Memory (LSTM) helps in extracting sequential features across RF samples. But a single RF sample does not hold much

information, so we need to learn features of adjoining samples. ConvLSTM architectures allow us to extract features over adjacent samples within a symbol, and the LSTM layer learns the sequential nature of this symbol representation as shown in Figure 3-27.

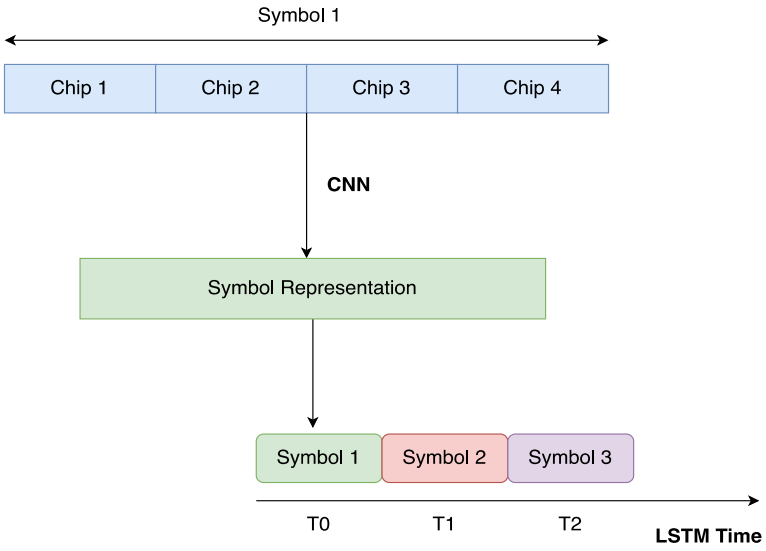


FIGURE 3-27: CONVLSTM OPERATION.

With the ConvLSTM layer as the main learning block, we design a 10-layer deep neural network. The model depth was chosen using trial and error method. We trained the model using our dataset of 14 nodes. The hyperparameters for training the model are shown in Table 3-3.

TABLE 3-3: LIST OF HYPERPARAMETERS.

Hyperparameter	Value
Batch Size	64, 256
Epochs	50
Learning Rate	0.001
LR Decay	0.00001
Optimizer	Adam

Evaluation and results

To evaluate our model, we run three different experiments:

1. Overall feasibility of our model.
2. Accuracy of the model variations with the amount of training data.
3. Impact of preprocessing stages

Feasibility

To ascertain our model would work, we evaluate the accuracy of the model. For the best-measured configuration, we achieve an accuracy of 95.43%. The confusion matrix is shown in Figure 3-28.

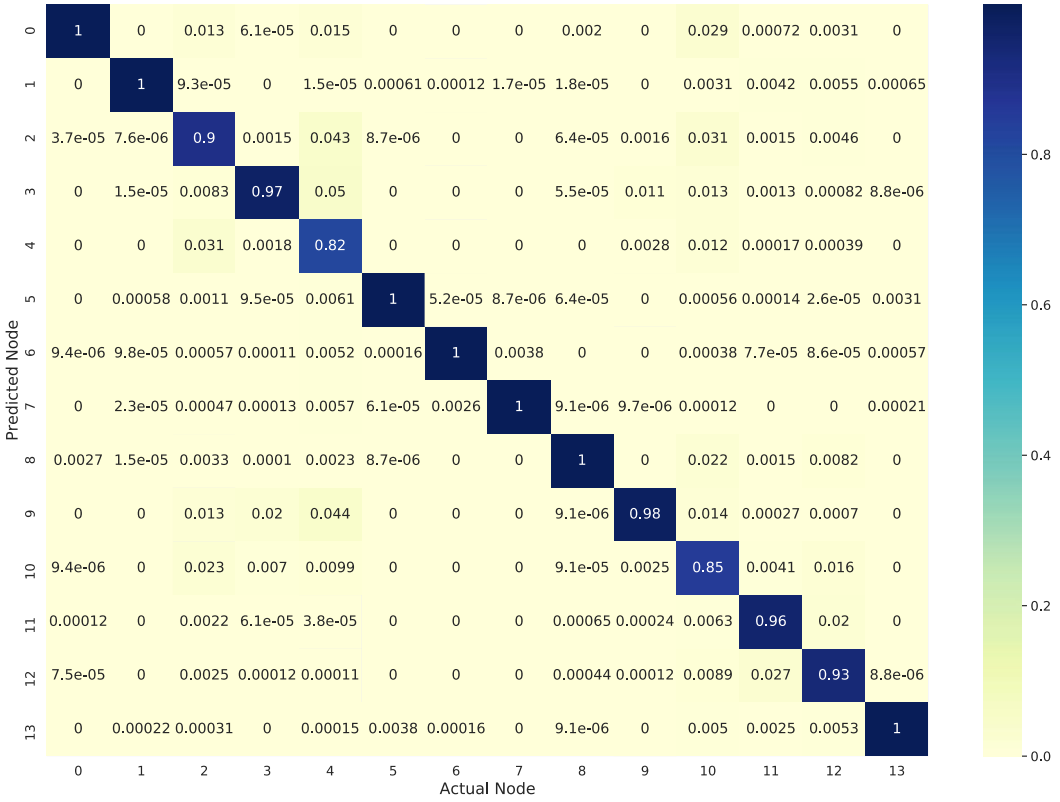


FIGURE 3-28: CONFUSION MATRIX FOR BEST MEASURED CASE.

We see we achieve a near-perfect accuracy on most nodes, with nodes 10 and 4 having lower classification accuracy of over 80%. Our two-factor authentication system would help mitigate these lower accuracy nodes, thus ensuring secure connectivity for low power IoT sensor nodes. Hence, we can conclude that our system is feasible.

Accuracy vs Number of Samples Needed

Figure 3-29 shows the accuracy of our model for different numbers of training examples. It is important to know how many messages we need to create a signature for the device. Our analysis shows that the accuracy of the model decreases with a lower number of training messages. Also, the accuracy varies more when the model is trained with a lower number of training messages. But overall, training with just 10 messages, gives us an accuracy of 70%, which we can improve with subsequent messages through online training.

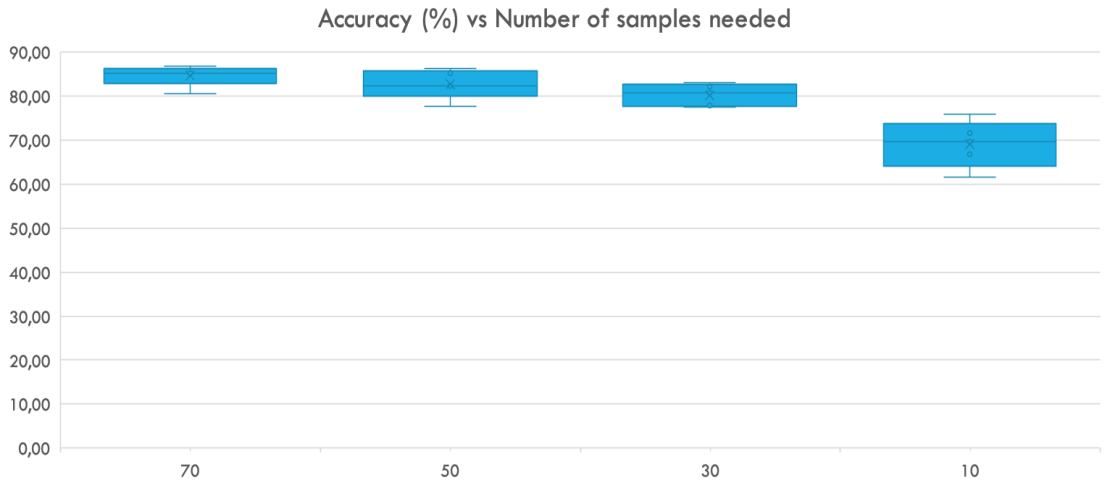


FIGURE 3-29: ACCURACY VS NUMBER OF TRAINING MESSAGES.

Impact of pre-processing stages

We also access which pre-processing techniques work best for our model. We do a comparative evaluation of different pre-processing techniques for training the model with 50 training messages. We use that normalization leads to a loss of accuracy, whereas scaling and no preprocessing have similar performance. With normalization, the amplitude of the signals is reduced from [-1,1] to [0,1], hence limiting the range of variations, making it harder for the model to learn. Also, the phase shifts at the zero crossing is an important feature for the model. Scaling (from [-1,1] to [-10,10]) does not contribute to increased accuracy which highlights that the model is learning inherent phase differences and the sequential nature of the signals rather than the amplitude differences. We get the best performance when applying the sliding window approach, which produces data augmentation and also helps in learning shift-invariant features, as shown in Figure 3-30.

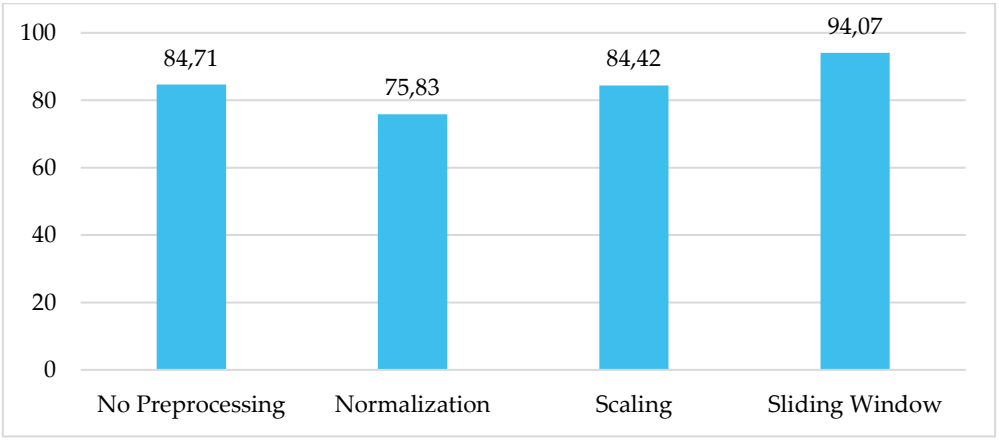


FIGURE 3-30: IMPACT OF PREPROCESSING STAGES.

4. Disaster Relief Using Autonomous Drone Scout

Recently, there has been a growing interest in drones and their applications to various sectors in the economy and in particular for applications pertaining to public safety. In particular, the Autonomous Drone Scout (ADS) scenario focus is on the need of a Public Safety agency. Drones are commissioned to scout a disaster area while utilizing efficient data collection and efficient transmission over 5G communication networks. The collected data are then transmitted to the edge of the network for cognitive processing and decision-making. The fleet control will be managed via drone fleet navigation software on the edge and drone collision avoidance system adopted on the drone itself. Together, the communication and the computing capabilities over different tiers of the platform facilitate the necessary means for low-cost and efficient rescuemissions. The ADS scenario will be used to evaluate two use cases, namely i) Drones Fleet Navigation and ii) Intelligent Image Processing for Drones. Both use cases are draw a complete ADS scenario, where in Drone fleet navigation the elementary needs of collaboration among drones are established. On top of that, intelligent image processing for drones can detect risks in disaster areas efficiently.

4.1. Use Case #1: Drone Fleet Navigation

A fleet management model is key to the success of drone missions where drone fleet navigation allows performing horizontal flight avoiding obstacles, with characteristics of low speed and high precision in flight. Several models have been developed to meet the requirements for drone fleet navigation. In the following, we present the 5G-DIVE solution for the drone fleet navigation use case including 5G-NSA connectivity, computing, virtualization (a.k. drone collision avoidance system), geolocation and image analytics for drones.

4.1.1. 5G-NSA Connectivity for Drone Navigation

For non-standalone (NSA) core network architecture used in ADS, the LTE-NR dual connectivity (EN-DC) feature is implemented within 5G architecture option 3, where the 5G-NSA core supports eNB connected via S1 interface and eNB/gNB connected via the X2 interface. In this architecture, the network owners have the option of deploying dual connectivity for data such as higher throughput with NR and larger coverage with eNB data traffic (i.e. it will be fully based on eNB). By using eNB as the anchor point of the control plane, it can guarantee service continuity and support rapid network construction in the initial stage of gNB deployment.

As shown in Figure 4-1, 5G components support drone-to-infrastructure (including camera hosted on drone) communication. In particular, the drone data and video stream from the camera are transmitted over the 5G NR in this use case.

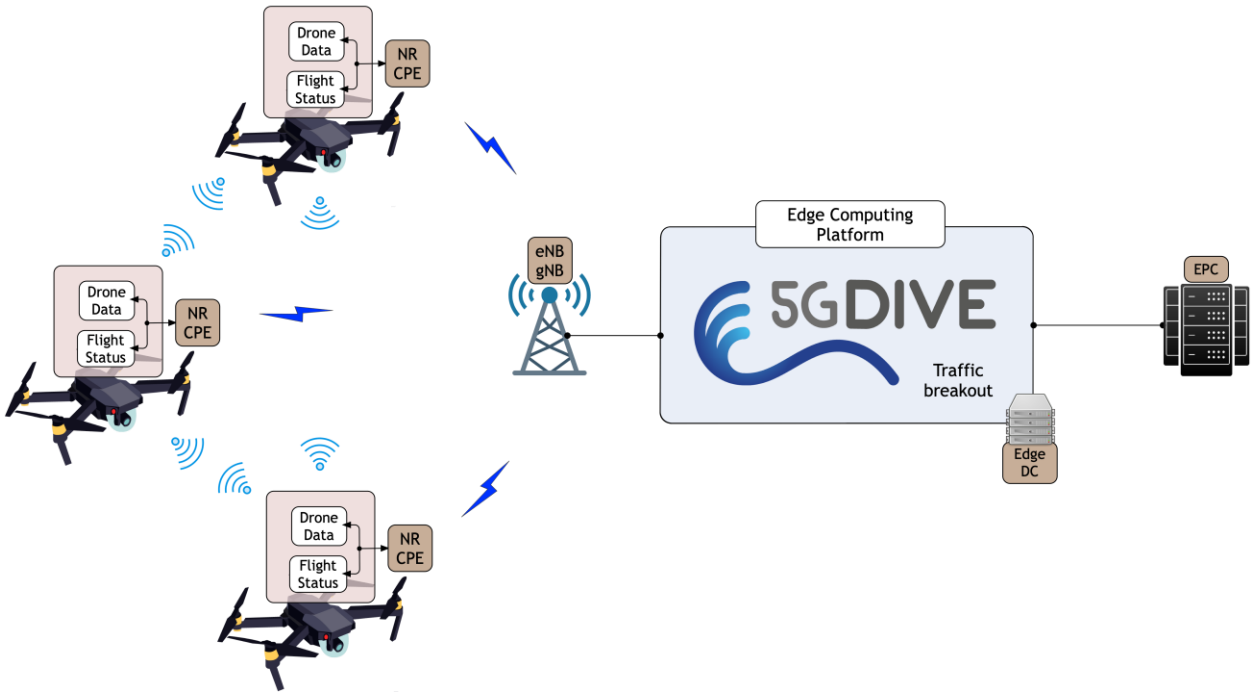


FIGURE 4-1: DRONE FLEET NAVIGATION CONNECTIVITY SUBSTRATS.

4.1.2. Computing and Virtualization for Drone Navigation

During the disaster relief mission of the drone fleet, the drone navigation server and drone collision avoidance system (DCAS) function are applied to support drones for executing missions. As depicted in Figure 4-2, , the drone navigation server, which is at the edge, sends waypoints and actions to drones, and the DCAS on each drone exchanges the drone' flight status in order to prevents colliding with each other. Furthermore, the collision avoidance function has decentralized characteristics while each drone automatically changes the drone's waypoint if potential collisions is detected based on the information collected by each drone. The flight status service passes the required information by DCAS application such as GPS coordinates, speed, waypoints, height… etc. Besides, Drone priority ID is an ID that indicates the flight priority of Drones. The Drone with higher priority will fly first rather than the drone with lower priority. While executing disaster relief missions, each drone continuously exchanges GPS location and drone ID with drone-to-drone communication (sidelink communication), which is used for DCAS.

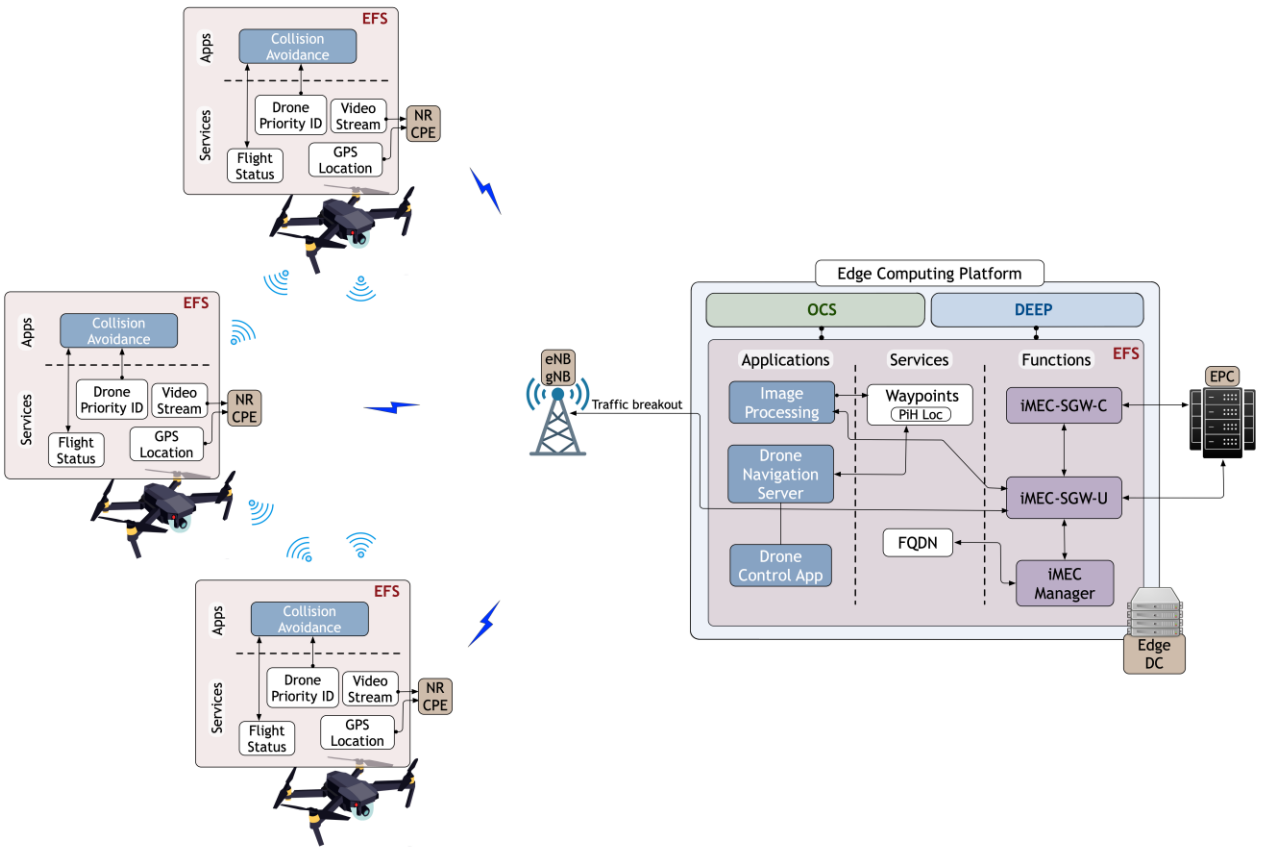


FIGURE 4-2: COMPUTING AND VIRTUALIZATION SUBSTRATS.

The operation of DCAS is depicted in Figure 4-3. The operation of DCAS is based on two cylinders defined volumes. Each drone has two concentric cylinders. These two cylinders are centred on the position of the drone itself as follows:

1. A collision Cylinder is used to detect potential collisions. If true, the drone will enter AVOID state.
2. SlowDown Cylinder used to slow down the drone's speed while there are drones nearby; Also used to detect the end of the AVOID state.

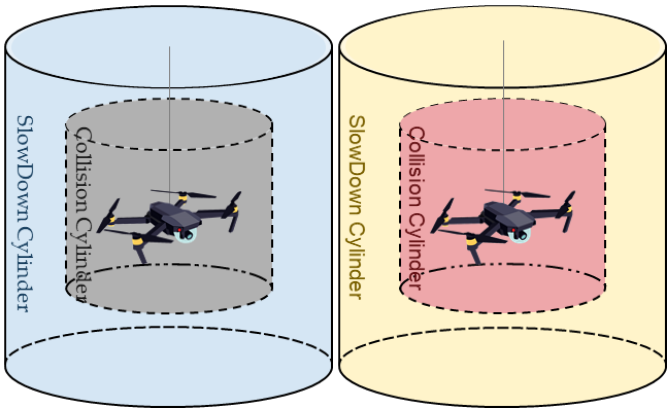


FIGURE 4-3: CONCENTRIC CYLINDERS OF DRONE COLLISION AVOIDANCE SYSTEM

The DCAS function keeps detecting the status of the SlowDown Cylinder and Collision Cylinder. The speed of a drone is slowed down if the drone's SlowDown Cylinder is overlapping with another drone's SlowDown Cylinder. A drone flies toward its destination if the drone's Collision Cylinder is not overlapping with any other drone's Collision Cylinder. Otherwise, the drone will enter the AVOID state. In the AVOID state, the collision avoidance function will dynamically adjust the drone's waypoint, as shown in Figure 4-4. . In case 1, the drones are too close to each other that their Collision Cylinders are overlapped. To make the Collision Cylinders not overlap, the only choice is to fly with right angle. In case 2, there is a short distance between drones. In order to keep the drones' Collision Cylinders not overlapped and make the collision avoidance process smooth, the DCAS decided waypoint is set besides to the other drone. A drone leaves AVOID state when it's SlowDown Cylinder is not overlapping with other drone's SlowDown Cylinder anymore. If a drone leaves AVOID state, it continues to fly toward its planned destination.

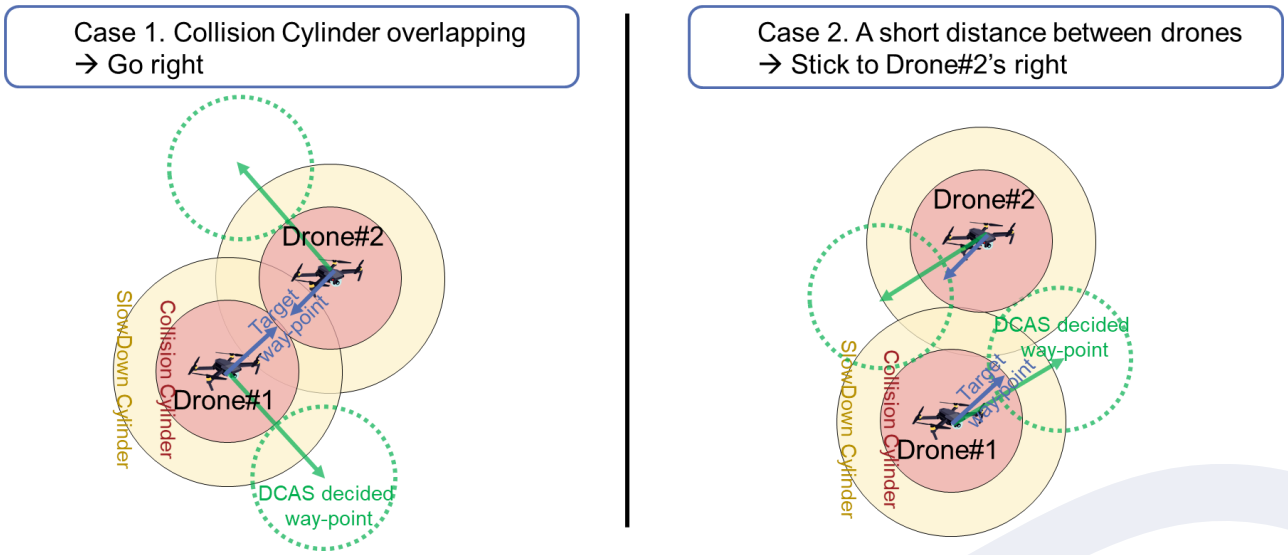


FIGURE 4-4: EXAMPLE OF DRONE COLLISION AVOIDANCE.

The drone navigation system will manage the flight control as follows:

1. Connect with the drone;
2. Operate in the route control interface map;
3. Click each flight point in the sequence;
4. Enter the altitude of each flight point;
5. Start flight plan automatically.

In Drone Fleet Navigation use case, the DCAS will be integrated with the drone navigation system as depicted in Figure 4-5. The DCAS will interact with drone navigation software at the edge through 5G network. The control of drone will be managed by DCAS and drone navigation software as follows:

- If it's blocked by any other drone, DCAS generates a way-point by itself.
- Else, DCAS flies to the waypoint retrieved from drone control software using API, e.g., RESTful API.

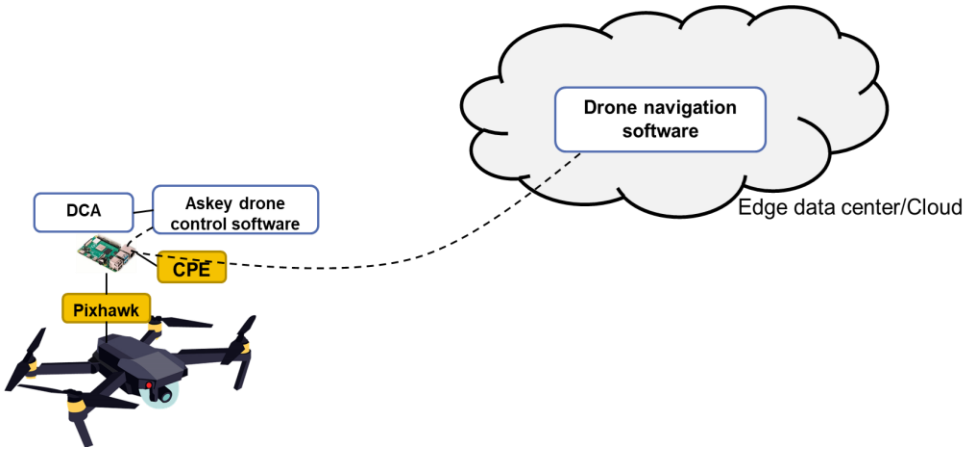


FIGURE 4-5: API INTERFACE BETWEEN DRONE NAVIGATION AND DRONE COLLISION AVOIDANCE SYSTEMS.

4.1.3. Geolocation and Image Analytics

During a disaster relief response scenario, drone video data and geolocation data are transmitted to the edge. On the edge, DASS will perform necessary operations like pre-processing and storage (Figure 4-6). For the drone video, the DASS is utilized in two ways. First, the drone video data is extracted into image frames. In this use case, the drone video frame rate is set to 30 frames per second. Second, those extracted image frames will be down-sampled into a specific target resolution that the Object Detection application (described in Figure 4-6) uses. This target resolution has a square shape (e.g.: 832 x 832). For an input image that is not square, padding (E.g.: black bar) will be added to the top and bottom of the image to make it square.

As for the geolocation data, the DASS will store it in its database directly without pre-processing. The geolocation data is timestamped and will be referenced in the future when the Object Detection application detects a Person in need for Help (PiH). When a PiH is found, GPS data will be retrieved and sent to the Drone Navigation Server. The Object Detection application will alert the Mission Dashboard for PiH geolocation data, while the Drone Navigation Server will calculate new waypoints to navigate the drone to PiH location.

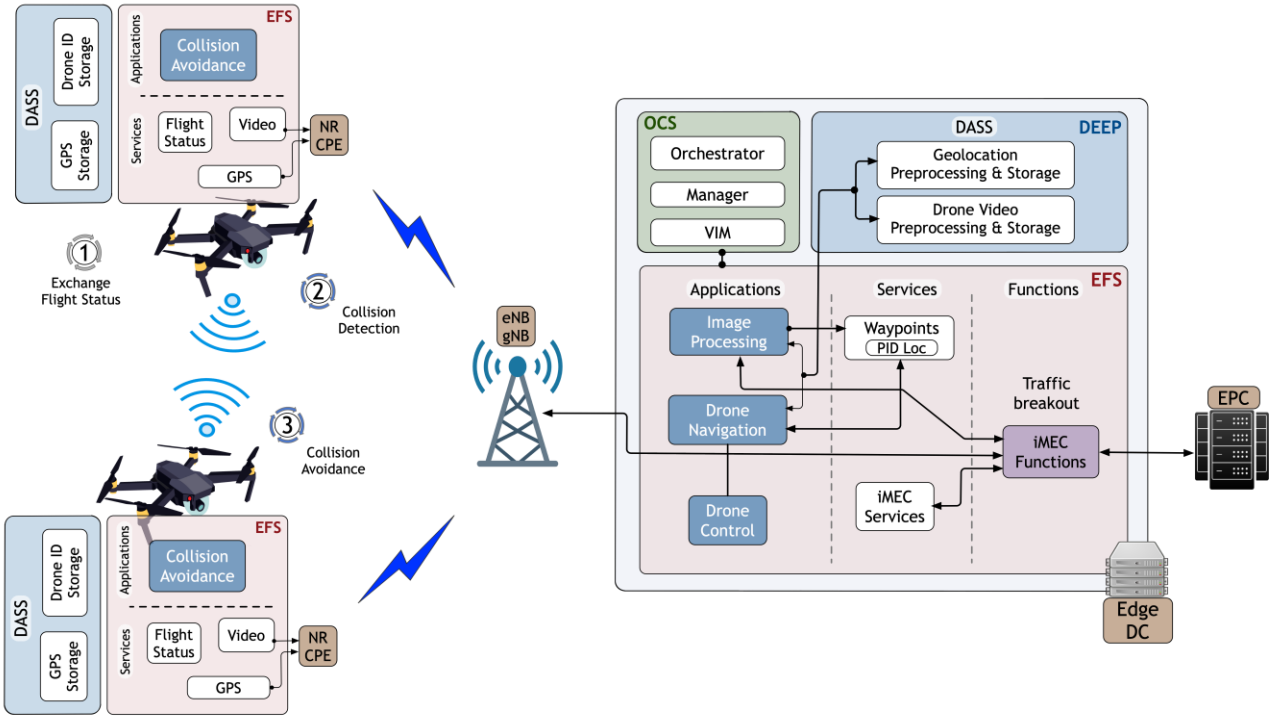


FIGURE 4-6: DASS FOR DRONE FLEET NAVIGATION.

Moreover, as shown in Figure 4-6, we detailed the DCAS workflow into several logical steps. First, in the EFS, DASS stores the drone data such as Drone ID and GPS. Then, each drone broadcasts the trajectory repetitively. Next, the drone location is used by drone fleet navigation software to decide the drone mission at the edge. Where the done automation path is sent to the drone (EFS), DCAS will use the automation path and the stored data to trace any risk and then change the drone path if needed. In this use case, DASS captures and extracts the GPS information and feeds them into DCAS to detect the possible collision and provide a collision avoidance mechanism.

4.1.4. Deployment options

Recent developments in the drone industry are adopting a new way of communication such as implementing mobile edge computing, drone-to-drone communication, and 5G connectivity. In drone fleet navigation, it is important to maintain the drone flight lifetime, quick application reaction and preserve the public safety. Besides, the effectiveness of fleet navigation to execute the required mission can be jeopardized if the battery is not sufficient, the communication interference is high and GPS is

not accurate during the flight. On the other hand, drone sensors produce raw data that requires computing analysis to make that data useful and actionable without human intervention. The deployment should deliver essential data fast and reliably enough for autonomous fleet navigation.

In this use case, the initial deployment option is to manage the drone fleet navigation from the edge computing platform. However, in certain missions, the drones could become close to each other and raise the risk of collision. Hence, we design the DCAS where it can be part of the drone EFS design. The DCAS is deployed on top of Raspberry pi with minimal power consumption. This deployment allows adopting more light-weight computing and a fast-distributed reaction for the drone environment. As for now, we use the GPS broadcast among drones to identify the safe area to keep executing the fleet navigation mission. DCAS will take control of the drone if any risk is detected. Also, the DCAS will release the drone control to the edge platform if the risk disappears. The current deployment option opens the door to adopt an intelligent algorithm that can adapt in runtime during drone flight time. Yet again, the rigorous design is a must when allowing any new smart application to be operated in the drone itself. Finally, yet importantly, we only considered the deployment at a drone or edge platform and excluded the cloud deployment option since it may not be able to serve low latency services targeted in ADS use cases.

4.2. Use Case #2: Intelligent Image Processing for Drones

Recently, drones have been extensively utilized because of their size and flexibility in the operating environment, for example, in[62], an aerial disaster relief response architecture is proposed for victim detection. Using sensors such as laser scanner and infrared depth camera, a 3D reconstruction of the area is created to aid in the detection. The similar architecture was developed in [63] where a drone with an electromagnetic sensor is utilized to perform victim detection. Another cloud-based multi-drone approach is proposed in [64] for disaster relief response. Although these efforts can detect disaster victims, they lack the real-time capability, and several do not provide latency measurements.

To add an intelligence feature on the drones, machine learning algorithms are commonly used. In[65], Haar cascade classifier was utilized for real-time detection of people and vehicles. The authors reported a detection rate of approximately 70% for people in each image frame. However, the use of Haar cascade classifier for detection and classification has a drawback of being application-specific. On the other hand, Convolutional Neural Network (CNN) has been widely used for object detection and classification. Unlike Haar cascade classifier, CNN feature set is learned automatically during training. In[65], the use of CNN for aiding a SAR team in a snow avalanche victim-detection scenario was proposed. The authors reported an accuracy of up to 97.59% and processing latency of 2.8s per image frame. In [66] and[67], two-stage object detection is proposed. Good performance accuracy for object detection and classification is obtained but with high inference time latency. Another method for object detection called single-shot detection is developed in [68][69]. Unlike the two-stage detection, this method skips one of the stages in its two-stage counterpart and hence has a lower inference time latency. Nevertheless, the realization of these disaster relief response systems gives rise to new challenges of meeting latency requirements in addition to efficient deployment, reliability, and scalability. To rise to these challenges, two key technologies, mobile edge computing (MEC) and

network function virtualization (NFV), have emerged as solutions where applications and services are brought closer to the edge of the network. While MEC eliminates traffic backhauling and thus end-to-end latency, NFV decouples network functions and applications from the underlying hardware and facilitates them in software over commodity hardware. Together, MEC and NFV could be complemented by machine learning and cognitive techniques to orchestrate and manage the virtualization environment.

In this part, we will introduce three pillars of intelligent image processing use case, in the end; we will present the use case flow including the interaction among these pillars. We consider the disaster relief area as for the topic. In the very first use case, we highlight the challenges lie between the communication of the user equipment (UE), i.e. drones, and the serving applications on the external network. We need to figure it out on dealing with any delay-sensitive applications. Furthermore, we also consider proposing a comprehensive design to encapsulate, manage and control the system by orchestrating the software and the hardware and have them virtualized. As for the applications, we propose two applications according to this topic. We detailed every part in the subsections below.

4.2.1. Drone Traffic Breakout for 5G-NSA

The main functionality of a regular cellular network is to move data packets back and forth between user equipment (UE) and the serving applications on the external network, i.e., the Internet or the Cloud. However, the distance of packet movement incurs long latency, and thus the delay-sensitive applications demand a remodelling of the network architecture.

To deal with this issue, iMEC, invented by ITRI, provides a powerful computing platform located on the border between random access network and core network. ITRI implements a specialized serving gateway (SGW) that not only interact with other 3GPP components by standard signaling, but also the routing module that intercepts and redirects packets to the localized applications. These applications are collocated with SGW on iMEC.

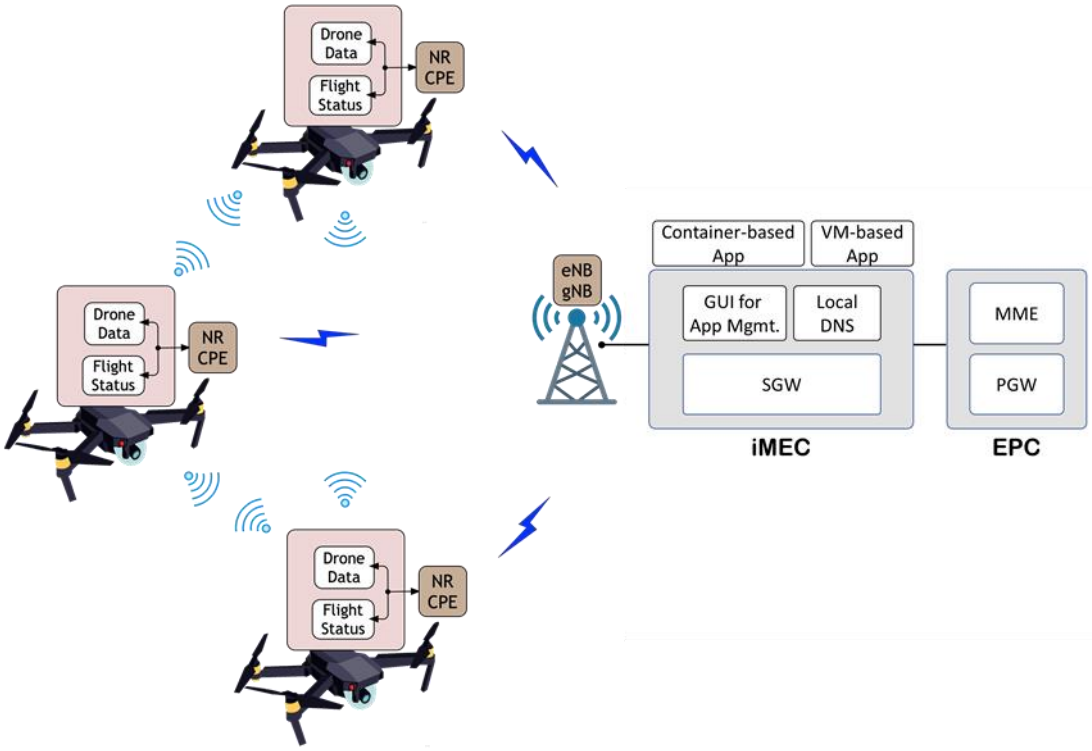


FIGURE 4-7: ADS TRAFFIC BREAKOUT.

Next, drones boot up, attaching to the cellular network. The drone-side applications ask the local DNS for the IPs of server-side Fleet Navigation and Intelligent Image Processing programs on iMEC, and then initiate connections. After the packets arrive at the eNB/gNB, the packets are encapsulated in a GTP-U tunnel and are transmitted to iMEC's vSGW. Eventually, vSGW decapsulates the packets and identifies that the destination IP address belongs to localized applications. Subsequently, vSGW directly outputs the packets to localized drone programs, rather than continuing to push the packets to another GTP-U tunnel to EPC. As a result, the traffic breakout at vSGW dramatically reduces the latency between a drone and its serving applications.

4.2.2. Computing and Virtualization for Drone Image Processing

As shown in Figure 4-8, Object Detection is an EFS application that performs pattern recognition in real-time. It consumes the data provided by the Drone (video stream and GPS data). It exposes the PiH Location to the EFS service platform. Each Object Detection application, denoted as a worker node, is a virtualized object detection algorithm with GPU-enabled computing capability. It executes two algorithms: (1) object detection algorithm to detect Person and/or Flag objects in form of bounding boxes, if any, and (2) candidate selection algorithm to searching any possible PiH objects from given bounding boxes information. Afterward, it forwards the information into the PiH Candidate Selection module, an EFS function, to validify the existence of the PiH objects by collecting consecutive processed images. Once verified, the validated images are matched with the current GPS information and exposed to the EFS service platform. Finally, end-users monitor the results with Mission Dashboard, an EFS application.

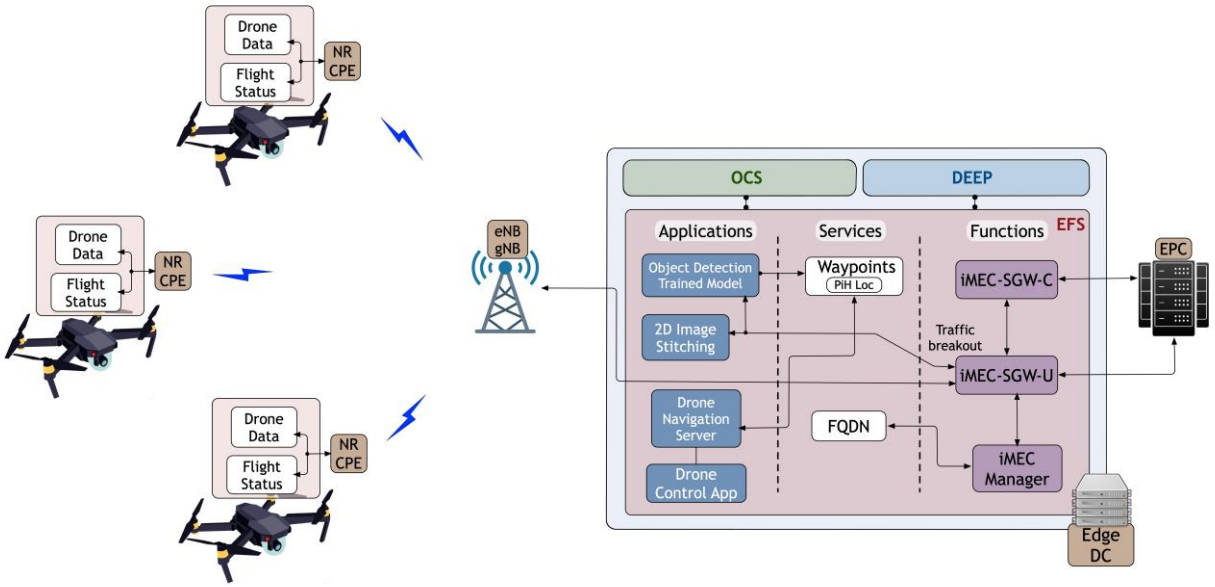


FIGURE 4-8: COMPUTING AND VIRTUALIZATION SUBSTRATS.

2D Image Stitching is another EFS application responsible for performing 2D aerial image stitching in real-time. It consumes data (images) extracted from the video stream provided by the Drone. It sends the output data to the Progressive Map service. Object Detection application triggers this 2D Image Stitching to initiate the stitching process once PiH objects are detected. It stitches images from a single and/or multiple drone(s).

4.2.3. Drone-based Intelligent Emergency Response

4.2.3.1. Victim Detection

The object detection module for the disaster relief response is based on the YOLOv3 algorithm[69]. We made some amendments and adopt YOLOv3 as the base of our object detection algorithm to be deployed on the DEEP platform. To fit our scenario, we train only Person and Flag objects sourced from COCO Dataset [70] and Google Open Image Dataset (OIDv4)[71]. We build an architecture model denoted as EagleEYE which stands for: Aerial Edge-enabled Disaster Relief Response System. Deployed at the edge with high computation capability, EagleEYE is able to process the aerial video data from drones and provide real-time PiH GPS location information.

The proposed EagleEYE System provides a scalable and latency-aware PiH detection with a guaranteed real-time response. It enables a high-resolution video stream to be extracted and processed in the GPU-Enabled edge servers in parallel by scaling up/down number of available worker applications. This intelligent adjustment guarantees the quality of service in maintaining the real-time responses. Here, we present a detailed design of EagleEYE architecture and map its design to the 5G-DIVE architecture. For DASS, EagleEYE employs two analytics components: (1) Data pre-processing, and (2) Data Storage.

When drone video data is received at the edge, the DASS extracts image frames and then to down sample it to a tar-get resolution. The number of the extracted frames is equal to the drone video source frame rate. The drone GPS data is timestamped and stored in the DASS for future cross-referencing with frames when a PiH object is found. The IESS of EagleEYE is utilized to train the dual object detection model. The IESS subsystem contains model catalogue which is a repository which stores algorithm implementations and trained models. After training, the runtime environment of the trained model is passed to the BASS subsystem. To deploy EagleEYE services, the BASS and OCS monitor the application-level latency and resource usage to man-age and scale the EFS components on the fly based on measurement reports.

Finally, EagleEYE employs five EFS applications: (1) Scheduler, (2) Worker application, (3) PiH candidate selection, (4) Drone Navigation, and (5) Mission Dashboard. The scheduler assigns image frames to the available worker applications. A worker application contains dual object detection module which is responsible to detect for person and flag objects and is the only scalable EFS component. PiH candidate selection is used to generate and locate PiH objects based on the results from the worker application. The drone navigation is used to update the drone's trajectory based on the located PiH objects. The mission dashboard is responsible to visualize and monitor PiH detection results. Figure 4-9 illustrates the EagleEYE architectural model and the workflow which is described as follows.

illustrates the EagleEYE architectural model and the workflow which is described as follows.

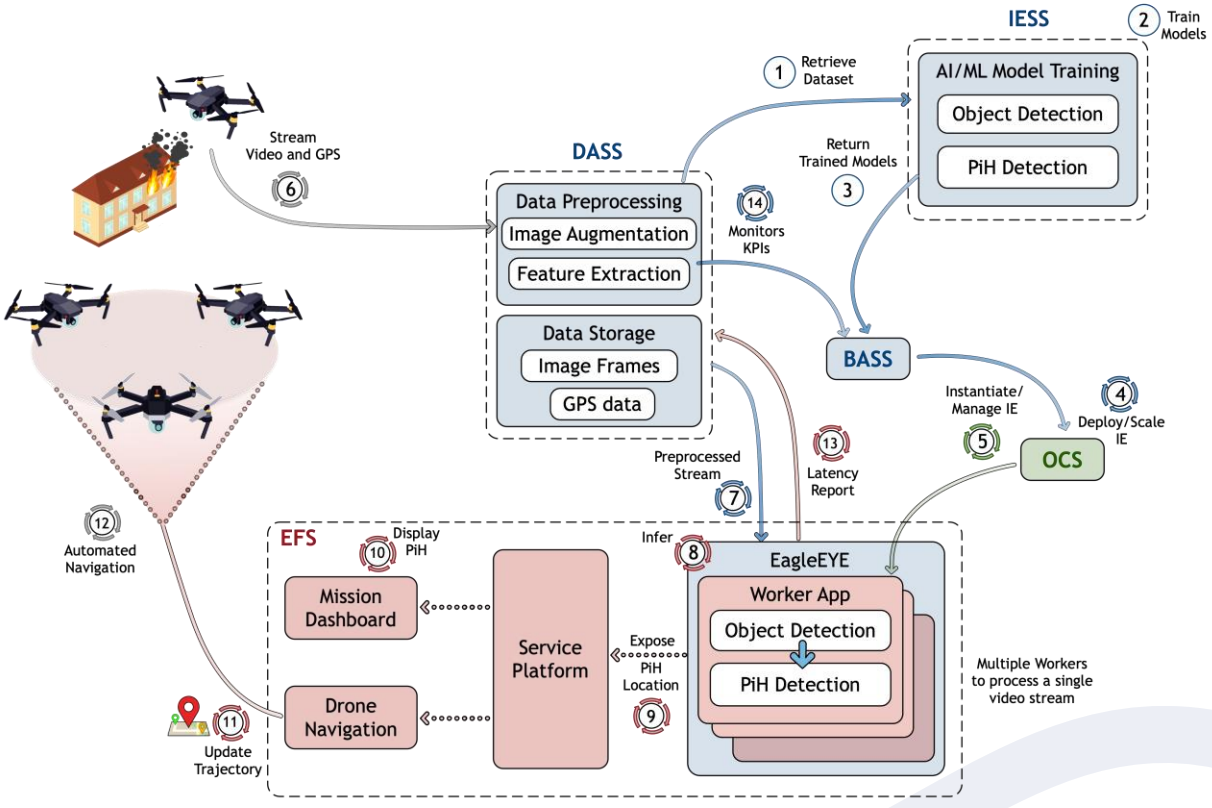


FIGURE 4-9: INTELLIGENCE ENGINE WORKFLOW FOR VICTIM DETECTION.

1. The datasets are retrieved from DASS which person and Flag objects.
2. IESS responsible to train the dataset in the Edge Server generating models, denoted as Trained Models (TMs).
3. The BASS selects the most suitable trained model to deploy worker applications in the EFS.
4. The BASS deploy worker application in the EFS via the OCS. Also, the BASS horizon-tally scales the deployed worker applications according to the system workload and measurement reports coming from the EFS.
5. The OCS allocates resources and instantiates the worker applications within the EFS in response to the BASS deployment specification.
6. After training and system configuration, the drone video stream and GPS data are transmitted to the DASS for augmentation and feature extraction.
7. The EagleEYE software components in EFS subscribe to the pre-processed drone data in DASS.
8. EagleEYE EFS components perform PiH detection. First, the scheduler extracts frames from the video stream and schedules tasks among worker applications in round-robin fashion. Each worker application produces bounding boxes of detected person and flag objects. The bounding boxes shared with the PiH candidate selection. Then, PiH candidate selection algorithm matches each generated PiH object, if any, with the collected GPS data.
9. The PiH information which includes the bounding boxes and GPS location is exposed to the service platform.
10. The drone navigation and the mission dashboard subscribe to the PiH information in the service platform.
11. The drone navigation server sends way-points to the drone for trajectory updates.
12. The drone automatically adjust its trajectory to provide a better view of the located PiH. In multi-drone deployment, different imagery angles can be obtained by autonomously navigate and coordinate drones at the disaster area.
13. The EagleEYE EFS components continuously send computing latency reports to DASS.
14. Finally, the BASS analyses the latency reports scale up/down the worker applications accordingly.

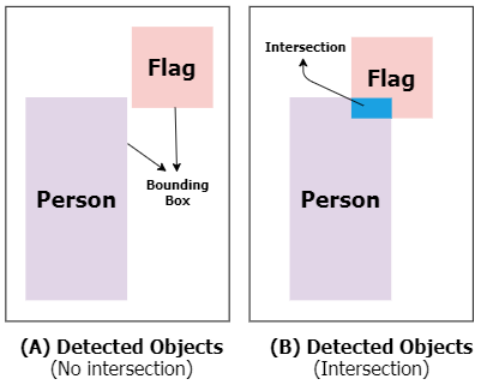


FIGURE 4-10: AN ILLUSTRATION OF A VALID AND INVALID INTERSECTION.

As for the PiH object modelling, we built an algorithm called PiH Candidate Selection (PCS). This algorithm uses the Person and Flag bounding box information from the object detection results as the

input. It brings the benefit of using a public dataset and avoids the extra efforts of creating a custom training dataset. To successfully pair the objects in real-time, PCS algorithm requires to collect at least one Person object and one Flag objects. If this condition is fulfilled, then the algorithm checks whether these objects exist in the output frame that has a valid intersection. An illustration of a valid intersection can be seen in Figure 4-10. In a successful merge, a bounding box will be drawn in the final output frame. Otherwise, no bounding box will be drawn.

4.2.3.2. Autoscaling towards Real-time Detection

During run-time, the EFS application will publish monitoring data to the DASS. The BASS will then subscribe to the DASS to retrieve the monitoring data and make necessary control and adjustment through the OCS to the EFS application deployment. This control and adjustment feature made by BASS is called the horizontal auto-scaling (Figure 4-11). With the horizontal auto-scaling, the BASS can automatically adjust the deployment of EFS application either by increasing or decreasing the number of deployed containers or VM or servers to handle for the change in workload.

In the case of ADS Object Detection, the BASS will adjust the EFS application deployment based on monitoring data such as object detection inference latency, CPU & GPU resource utilization, and the number of active EFS applications. These are some of the monitoring data that the BASS can utilize to make adjustment decisions. The adjustment is made by the BASS with periodic checking of monitoring data to see whether they are within the set KPI limit or not. For example, if the set KPI limit for object detection inference latency is reached, the BASS can increase the deployment of the object detection application to help cope with the load and to reduce the inference latency. The BASS can then also reduce the number of deployed EFS applications once the KPI requirement is relaxed.

The Horizontal auto-scaling of the BASS is a crucial feature to have. Instead of costly vertical scaling to add more computing hardware (e.g.: CPU, GPU, etc.) to cope with the increasing load, horizontal auto-scaling can provide a more efficient way to manage the resources in the edge. For the ADS use case, horizontal auto-scaling means that we can efficiently scale our EFS application deployment to serve for more drones without having to add more computing hardware to the edge.

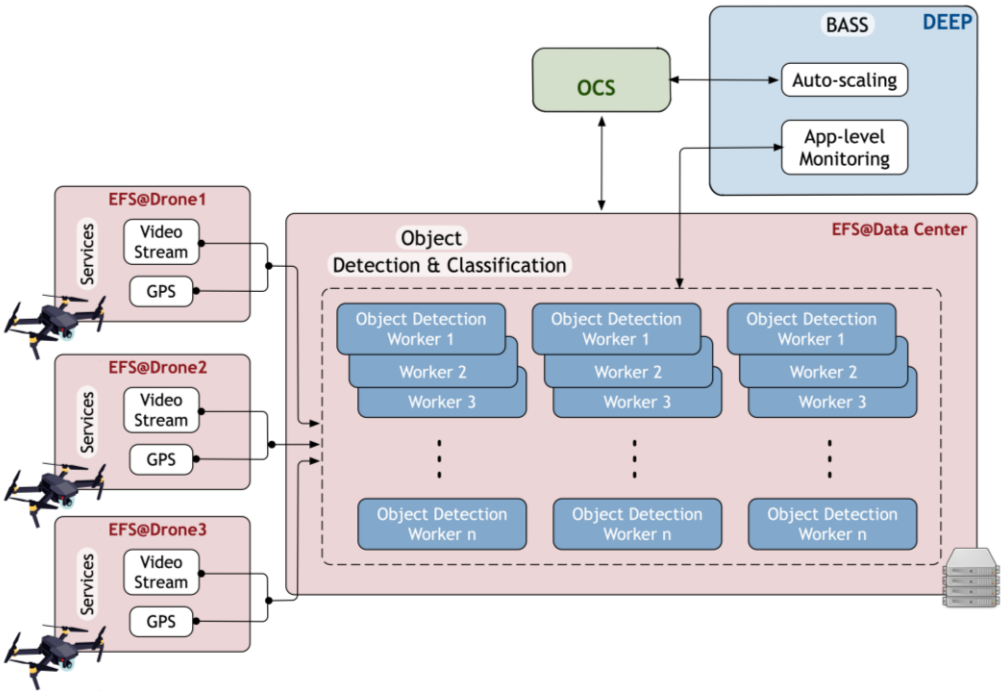


FIGURE 4-11: HORIZONTAL AUTOSCALING FOR VICTIM DETECTION.

5. Conclusions and Next Steps

This deliverable presented the baseline specification of the 5G-DIVE solution for the use cases targeted in the I4.0 and ADS vertical pilots.

Section 2 presented first the overall design framework for the solution targeted in 5G-DIVE. It introduced first the underlying 5G connectivity stratum with its Core, Transport, and RAN, putting emphasis on new features in 5G compared to 4G which are deemed beneficial to the use cases targeted in 5G-DIVE. Preliminary results of the RAN connectivity were also provided. Next, the underlying edge infrastructure stratum was presented taking reference in the 5G-CORAL edge and fog computing infrastructure and complementing this infrastructure with an enhanced edge data centre OPTUNS. The 5G-DIVE DEEP stratum was next presented introducing the three support systems targeted namely DASS, BASS, and IESS, and the framework of their use by the intelligence engines targeted. A preliminary study on the potential application of DLT technologies into DEEP was also outlined.

Following on the design framework from Section 2, Section 3 and Section 4 presented a deep-dive of the detailed specification in each of the use cases targeted, namely, i) digital twinning, ii) zero-defect manufacturing, and iii) massive MTC, for the I4.0 vertical pilot, and i) drones fleet navigation and ii) intelligent image processing for Drones, for the ADS vertical pilot, respectively. For each use case, we presented the reference design showing the end-to-end 5G-DIVE solution and elaborated on the key design components, their mapping to the 5G-DIVE architecture, and the workflow showing the different steps and interactions with DASS, BASS, IESS, EFS, and OCS, in deploying a given intelligent engine for a given use case. These engines included modules that involve applications and functions to execute intelligently a given task for a given use case, such as predictive maintenance in digital twinning, object defect detection in ZDM, radio security for massive MTC, geolocation and image analytics in ADS use cases.

The specification in this deliverable already served as a basis for first implementations and reported some preliminary results from these implementations. The next steps therefore lie in progressing first the ongoing implementations, identifying additional modules for implementation, and the evaluation of the end-to-end solutions in the targeted vertical pilots in conjunction with WP3. Following on the insights gained from the experimentations, the specification in this deliverable will then be refined and the final results reported in the future deliverable D2.3.

6. References

[1] ITU-R M.2083-0, "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond", September 2015. [Online]. Avalaible: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf [Accessed 24 September 2020]

[2] 3GPP, "System architecture for the 5G System (5GS)", 3GPP TS 23.501 V16.5.1 (2020-08).

[3] China Mobile Research Institute, "C-RAN. The Road Towards Green RAN", version 2.5, October 2011. [Online]. Avalaible: http://labs.chinamobile.com/cran/wp-content/uploads/CRAN_white_paper_v2_5_EN.pdf [Accessed 24 September 2020]

[4] ITU-T Technical Report, "Transport network support of IMT-2020/5G", October 2018.

[5] R. Rokui, S. Homma, K. Makhijani, L.M. Contreras, J. Tantsura, "IETF Definition of Transport Slice", draft-nsdt-teas-transport-slice-definition-03 (work in progress), July 2020.

[6] L.M. Contreras, S. Homma, J. Ordonez-Lucena, "Considerations for defining a Transport Slice NBI", draft-contreras-teas-slice-nbi-02 (work in progress), July 2020.

[7] L.M. Contreras, Ó. González, V. López, J.P. Fernández-Palacios, J. Folgueira, "iFUSION: Standards-based SDN Architecture for Carrier Transport Network", IEEE Conference on Standards for Communications and Networking (CSCN), Granada, Spain, October 2019

[8] E. Dahlman, S. Parkvall, and J. Sköld, "5G NR - The Next Generation Wireless Access Technology", Elsievier, August 2018.

[9] Ericsson White Paper, "5G wireless access: an overview", [Online]. Avalaible: https://www.ericsson.com/en/reports-and-papers/white-papers/5g-wireless-access-an-overview [Accessed 24 September 2020]

[10] Janne Peisa et al., "5G evolution: 3GPP releases 16 and 17", Ericsson Review, March 2020, [Online]. Avalaible: https://www.ericsson.com/493cdb/assets/local/reports-papers/ericsson-technology-review/docs/2020/5g-nr-evolution.pdf [Accessed 24 September 2020]

[11] Ericsson radio system, "Ericsson Radio Dot", [Online]. Avalaible: https://www.ericsson.com/en/portfolio/networks/ericsson-radio-system/radio/indoor/radio-dot-system [Accessed 24 September 2020]

[12] 5G-DIVE Project, "Delivrable D1.1 – 5G-DIVE architecure and detailed analysis of vertical use cases", April 2020. [Online]. Avalaible: https://5g-dive.eu/wp-content/uploads/2020/04/D1.1_Final.pdf [Accessed 24 September 2020]

[13] 5G-CORAL Project, "Deliverable D1.1 - 5G-CORAL initial system design, use cases, and requirements", February 2018. [Online]. Avalaible: http://5g-coral.eu/wp-content/uploads/2018/04/D1.1_final7760.pdf [Accessed 24 September 2020]

[14] ETSI GS NFV 002, "Network Functions Virtualisation (NFV); Architectural Framework", October 2013. [Online]. Avalaible: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf [Accessed 24 September 2020]

[15] ETSI GR MEC 017 V1.1.1, "Mobile Edge Computing (MEC): Deployment of Mobile Edge Computing in an NFV environment ", February 2018. [Online]. Avalaible:

https://www.etsi.org/deliver/etsi_gr/MEC/001_099/017/01.01.01_60/gr_MEC017v010101p.pdf [Accessed 24 September 2020]

[16] S. Sukhmani, M. Erol-Kantarci, M. Sadeghi, and A. Saddik, "Edge caching and computing in 5G for mobile AR/VR and tactile internet," IEEE MultiMedia 26, 21–30 (2019).

[17] 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), pp.A28-A37.

[18] Eclipse zenoh, "Zero overhead Pub/sub, Store/Query and compute", [Online]. Avalaible: https://github.com/eclipse-zenoh/zenoh.git [Accessed 24 September 2020]

[19] Rust, "Rust programing language", [Online]. Avalaible: https://www.rust-lang.org/ [Accessed 24 September 2020]

[20] Eclipse Mosquitto, "An open source MQTT broker", [Online]. Avalaible: https://mosquitto.org/ [Accessed 24 September 2020]

[21] Spring, "Spring Java framework", [Online]. Avalaible: https://spring.io/ [Accessed 24 September 2020]

[22] Jinja, "Jinja – templating language for Python", [Online]. Avalaible: https://jinja.palletsprojects.com/en/2.11.x/ [Accessed 24 September 2020]

[23] MongoDB, Inc., "MongoDB – a cross-platform document-oriented database program", [Online]. Avalaible: https://www.mongodb.com/ [Accessed 24 September 2020]

[24] Rancher, "Lightweight Kubernetes", [Online]. Avalaible: https://k3s.io/ [Accessed 24 September 2020]

[25] Eclipse fog05 project, "F0rce – a fog orchestration engine git repository", [Online]. Avalaible: https://github.com/eclipse-fog05/fog05/tree/f0rce [Accessed 24 September 2020]

[26] H2O.ai, "H2O 3 – an open source in- memory, distributed, fast and scalable machine learning and predictive analytics platform official documentation, [Online]. Avalaible: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/faq/java.html [Accessed 24 September 2020]

[27] H2O.ai, "Open source AI and machine learning driven enterprise-ready platforms", [Online]. Avalaible: https://www.h2o.ai/ [Accessed 24 September 2020]

[28] Computational Genetics Lab at University of Pennsylvania, "TPOT- python automated machine learning tool", [Online]. Avalaible: https://epistasislab.github.io/tpot/ [Accessed 24 September 2020]

[29] AutoGluon, "AutoML Toolkit for Deep Learning", [Online]. Avalaible: https://autogluon.mxnet.io/ [Accessed 24 September 2020]

[30] Microsoft, "Neural Network intelligence official git repository", [Online]. Avalaible: https://github.com/microsoft/nni [Accessed 24 September 2020]

[31] Ray project, "Ray – simple and universal API for building distributed applications", [Online]. Avalaible: https://github.com/ray-project/ray [Accessed 24 September 2020]

[32] Docker, "Docker registry official documentation", [Online]. Avalaible: https://docs.docker.com/registry/ [Accessed 24 September 2020]
[33] ONNX, "Open Neural Network Exchange", The Linux foundation projects, [Online]. Avalaible: https://onnx.ai/ [Accessed 24 September 2020]
[34] The HDF Group, "HDF5- High performance data management and storage suite", [Online]. Avalaible: https://www.hdfgroup.org/solutions/hdf5/ [Accessed 24 September 2020]
[35] Joblib, "Tools for lightweight pipelining in Python", [Online]. Avalaible: https://joblib.readthedocs.io/en/latest/ [Accessed 24 September 2020]
[36] A. G´eron, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensor- Flow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019.
[37] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, 2003, pp. 3–10.
[38] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarc´on, M. Sol´e, V. Munt´es-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, et al., "Knowledge-defined networking," ACM SIGCOMM Computer Communication Review, vol. 47, no. 3, pp. 2–10, 2017.
[39] Calastone White paper, "Distributed lager technology", October 2019. [Online]. Avalaible: https://www.theia.org/sites/default/files/2019-10/DLT%20and%20funds%20white%20paper.pdf [Accessed 9 September 2020]
[40] Astri, "Whitepaper on Distributed Ledger Technology", November 2016. [Online]. Avalaible: https://www.hkma.gov.hk/media/eng/doc/key-functions/finanical-infrastructure/Whitepaper_On_Distributed_Ledger_Technology.pdf [Accessed 24 September 2020]
[41] Hyperledger fabric project, "An umbrella project of open source blockchains and related tools", The Linux foundation projects, [Online]. Avalaible: https://www.hyperledger.org/use/fabric [Accessed 24 September 2020]
[42] Corda, "Open-source blockchain platform for business", [Online]. Avalaible: https://www.corda.net/ [Accessed 24 September 2020]
[43] Consensys Quorum, "Open-source blockchain platform for business", [Online]. Avalaible: https://www.goquorum.com/ [Accessed 24 September 2020]
[44] Openchain, "Blockchain technology for the enterprise", [Online]. Avalaible: https://www.openchain.org/ [Accessed 24 September 2020]
[45] Ethereum, "Ethereum for Enterprise documentation", [Online]. Avalaible: https://ethereum.org/enterprise/ [Accessed 24 September 2020]
[46] Huawei, "Huawei LTE CPE B315s-22 product description", [Online]. Avalaible: https://consumer.huawei.com/ie/support/routers/b315s-22/ [Accessed 24 September 2020]
[47] Huawei, "Huawei 5G CPE Pro product description", [Online]. Avalaible: https://consumer.huawei.com/en/routers/5g-cpe-pro/ [Accessed 24 September 2020]
[48] 5G-Coral Project, "Deliverable D3.2 - Refined design of 5G-CORALorchestration and control system and future directions", May 2019. [Online]. Avalaible: http://5g-coral.eu/wp-content/uploads/2019/06/D3.2.pdf [Accessed 9 September 2020]

[49] Lee, Kyungmin & Chu, David & Cuervo, Eduardo & Kopf, Johannes & Wolman, Alec & Degtyarev, Yury & Grizan, Sergey & Flinn, Jason. (2015). Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. GetMobile: Mobile Computing and Communications. 19. 14-17. 10.1145/2867070.2867076.

[50] 5G-CORAL Project, "Deliverable D4.2 - 5G-CORAL Proof of Concept and Future Directions", August 2019. [Online]. Avalaible: http://5g-coral.eu/wp-content/uploads/2019/09/D4.2_FINAL.pdf. [Accessed 9 September 2020]

[51] Nikaein, N., Schiller, E., Favraud, R., Knopp, R., Alyafawi, I., & Braun, T. (2016). Towards a cloud-native radio access network. In Advances in Mobile Cloud Computing and Big Data in the 5G Era. Springer.

[52] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[53] B. Han, V. Gopalakrishnan, L. Ji and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," in IEEE Communications Magazine, vol. 53, no. 2, pp. 90-97, Feb. 2015, doi: 10.1109/MCOM.2015.7045396.

[54] SEMTECH, "LoRa project", [Online]. Available: https://www.semtech.com/lora [Accessed 9 September 2020]

[55] Gr-lora project. "GNU Radio blocks for receiving LoRa modulated radio messages using SDR". [Online]. Available: https://github.com/rpp0/gr-lora

[56] Vijayendra Walvekar, "Virtualizing LoRa baseband functionalities to the Edge", Dissertation, 2019.

[57] 5G-CORAL Project, "Deliverable D4.1 5G CORAL testbed definition, integration and demonstration plans", May 2018. [Online]. Available: http://5g-coral.eu/wp-content/uploads/2018/09/D4.112820.pdf [Accessed 9 September 2020]

[58] GNURadio, "The free & open software radio ecosystem", [Online]. Avalaible: https://www.gnuradio.org/ [Accessed 9 September 2020]

[59] Ettus Reasearch, "Software-defined radio platforms supplier", [Online]. Available: https://www.ettus.com/ [Accessed 9 September 2020]

[60] Pycom Fipy, "Pycom go ivent", [Online]. Available: https://docs.pycom.io/index.html [Accessed 9 September 2020]

[61] Pieter van Schalkwyk, Dr. Shi-Wan Lin, Dr. Somayeh Malakuti, "A Short Introduction to Digital Twins", IIC. [Online]. Available: https://www.iiconsortium.org/news/joi-articles/2019-November-JoI-A-Short-Introduction-to-Digital-Twins.pdf [Accessed 9 September 2020]

[62] Lee, S., Har, D. and Kum, D., 2016, December. Drone-assisted disaster management: Finding victims via infrared camera and lidar sensor fusion. In 2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE) (pp. 84-89). IEEE.

[63] Tanzi, T.J., Chandra, M., Isnard, J., Camara, D., Sebastien, O. and Harivelo, F., 2016. TOWARDS" DRONE-BORNE" DISASTER MANAGEMENT: FUTURE APPLICATION SCENARIOS. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences, 3(8).

[64] Alex, C. and Vijaychandra, A., 2016, December. Autonomous cloud based drone system for disaster response and mitigation. In 2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA) (pp. 1-4). IEEE.

[65] Bejiga, M.B., Zeggada, A. and Melgani, F., 2016, July. Convolutional neural networks for near real-time object detection from uav imagery in avalanche search and rescue operations. In 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS) (pp. 693-696). IEEE.

[66] He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

[67] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

[68] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[69] Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[70] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[71] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T. and Ferrari, V., 2018. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. arXiv preprint arXiv:1811.00982.

[72] ETSI, “European Telecommunications Standards Institute official web page”, [Online]. Avalaible: https://www.etsi.org/ [Accessed 9 September 2020]

[73] ETSI GS NFV 002, "Network Functions Virtualisation (NFV); Architectural Framework", October 2013. [Online]. Avalaible: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf [Accessed 24 September 2020]

[74] Islam, Mohammad Manzurul & Morshed, Sarwar & Goswami, Parijat. (2013). Cloud Computing: A Survey on its limitations and Potential Solutions. International Journal of Computer Science Issues. 10. 159-163.

[75] Zhang, Lixia, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, K. C. Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. "Named data networking." ACM SIGCOMM Computer Communication Review 44, no. 3 (2014): 66-73.

[76] 5G-DIVE project, “Delivrable D1.1: 5G-DIVE architecture and detailed analysis of vertical use cases,” April 2020. [Online]. Avalaible: https://5g-dive.eu/wp-content/uploads/2020/04/D1.1_Final.pdf [Accessed 24 September 2020]

[77] Eclipse Zenoh project, “Zero Overhead Pub/sub, Store/Query and compute”, [Online]. Avalaible: http://zenoh.io [Accessed 24 September 2020]

[78] M. Li et al., "Multipath Transmission for the Internet: A Survey," in IEEE Communications Surveys & Tutorials, vol. 18, no. 4, pp. 2887-2925, Fourthquarter 2016, doi: 10.1109/COMST.2016.2586112.

[79] Eclipse Zenoh project, "Zenoh.net messages specification", [Online]. Avalaible: https://github.com/eclipse-zenoh/zenoh/blob/rust-master/zenoh-protocol/src/proto/msg.rs [Accessed 24 September 2020]
[80] Java, "Variable length encoding", June 2019. [Online]. Avalaible: https://golb.hplar.ch/2019/06/variable-length-int-java.html [Accessed 24 September 2020]
[81] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.
[82] Liu, Wei & Anguelov, Dragomir & Erhan, Dumitru & Szegedy, Christian & Reed, Scott & Fu, Cheng-Yang & Berg, Alexander. (2016). SSD: Single Shot MultiBox Detector. 9905. 21-37. 10.1007/978-3-319-46448-0_2.
[83] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.
[84] Cai, Zhaowei & Vasconcelos, Nuno. (2017). Cascade R-CNN: Delving into High Quality Object Detection.
[85] Latah, Majd & Toker, Levent. (2018). Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview. IET Networks. 8. 10.1049/iet-net.2018.5082.
[86] Azzouni, Abdelhadi & Pujolle, Guy. (2018). NeuTM: A neural network-based framework for traffic matrix prediction in SDN. 1-5. 10.1109/NOMS.2018.8406199.
[87] Latah, Majd & Toker, Levent. (2018). Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview. IET Networks. 8. 10.1049/iet-net.2018.5082.
[88] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.
[89] Yu, Yiding, Taotao Wang, and Soung Chang Liew. "Deep-reinforcement learning multiple access for heterogeneous wireless networks." IEEE Journal on Selected Areas in Communications 37.6 (2019): 1277-1290.
[90] Xu, Zhiyuan, et al. "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs." 2017 IEEE International Conference on Communications (ICC). IEEE, 2017.
[91] Li, Ji, et al. "Deep reinforcement learning based computation offloading and resource allocation for MEC." 2018 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2018.
[92] Yang, Tianyu, et al. "Deep reinforcement learning based resource allocation in low latency edge computing networks." 2018 15th International Symposium on Wireless Communication Systems (ISWCS). IEEE, 2018.
[93] F. Codevilla, et al. "End-to-End Driving Via Conditional Imitation Learning," 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018.
[94] E. Kaufman, et al. "Deep Drone Racing: Learning Agile Flight in Dynamic Environments.", CoRR, 2018.
[95] Duan, et al. "One-Shot Imitation Learning." Advances in Neural Information Processing Systems 30, 2017.
[96] Chelsea Finn, et al. "One-Shot Imitation Learning via Meta-Learning." Advances in Neural Information Processing Systems 30 (NIPS), 2017.

[97] J. Tobin, et al., "Domain randomization for transferring deep neural networks from simulation to the real world," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[98] T. Yu, et al., One-shot imitation from observing humans via domain-adaptive meta-learning. Robotics: Science and Systems, 2018.

7. Appendix – ETSI MEC Reference Architecture

ETSI MEC (Multi-access Edge Computing) is the framework for edge computing developed by ETSI[72]. It defines the components and interfaces enabling the implementation of MEC applications as software-only entities that run on top of a virtualisation infrastructure. Such infrastructure is located in or close to the edge of the network.

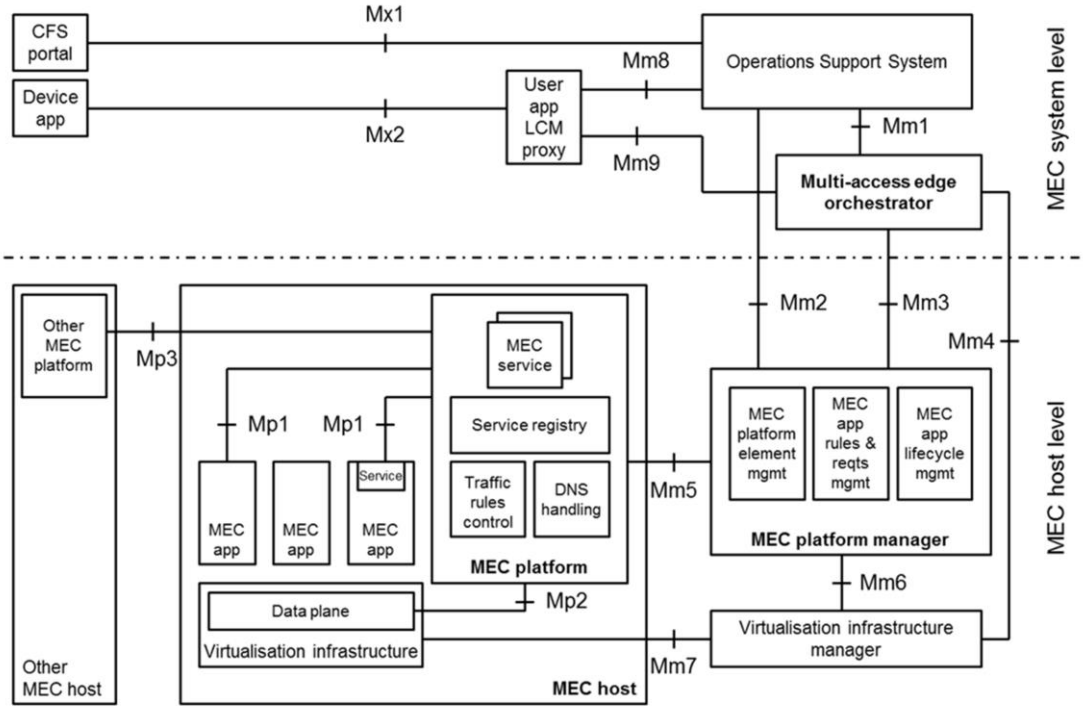


FIGURE 7-1: ETSI MEC REFERENCE ARCHITECTURE.

The MEC framework is composed by several entities as represented in Figure 7-1. This component is divided in two levels. The MEC system level, that comprises all the components and functionalities that are used system-wide, like orchestration and northbound interfaces to existing OSS/BSS. On the other hand the MEC host level comprises all the components and functionalities that resides in each MEC host and are relevant at host level, such as application managers.

7.1.1.1. MEC System Level

At this level all the system-wide functionalities are present and the main component is the Multi-access Edge Orchestrator (MEAO). It provides orchestration functionalities for the whole MEC system and interfaces to existing OSS/BSS as well as interaction with third-party applications via the User app LCM proxy. The MEAO maintains a global view of the MEC system, providing catalogue for MEC application packages and services, onboarding and validation functionalities for such packages.

Allocation and relocation based on both MEC application requirement and operator policies. Keeps track of the overall resource utilisation and running applications and services within the system.

It interoperates with components at the MEC host level when it has to validate or enforce rules and for gathering monitoring information used to keep an update view of the system.

7.1.1.2. MEC Host Level

The MEC host level comprises, the MEC platform manager, the VIM, the MEC platform and MEC applications.

The MEC platform manager is in charge of the lifecycle management of a given MEC platform, this includes configuration of traffic rules, DNS record and registration of new services. It also manages the lifecycle of single MEC applications, including reconfiguration and monitoring.

The VIM provides the virtualised infrastructure used to run MEC application and the MEC platform, it configures, manages and monitor the infrastructure, it provides the same functionalities of the VIM in the ETSI NFV framework[73].

The MEC platform provides an environment where MEC applications can discover, advertise, consume and offer different MEC services, along with configuration of traffic rules and DNS records received via the platform manager and providing access to operator-provided services like persistent storage, time-of-day and RNIS.

MEC applications are virtualized entities that runs on the virtualised infrastructure present within an MEC host. They communicate with the MEC platform to provide and consume MEC services as well as support for some lifecycle operations (e.g. relocation of user state). MEC applications can come with a set of rules and requirements that are validated prior to instantiation.

8. Appendix – Zenoh-based Data Management

8.1. Introduction

Data management is a complex task comprising different approaches and techniques depending on the scenario and application domains. Historically, Information Technology (IT) systems have focused on the capability of storing and querying data over the network. This is usually achieved via **databases** (e.g., SQL, NoSQL, key-value, etc.) that provide all the necessary management for **data at rest**. Conversely, Operation Technology (OT) systems have focused on the capability of distributing data over the network. This is usually achieved via **publish/subscribe message bus** that provide the necessary management for **data in motion.**

Recently, these two domains have started looking into each other thanks to two complementary benefits: (1) IT systems (e.g., Cloud) proved to be economically convenient, and (2) OT systems (e.g., factories) proved to be more scalable and reactive over distributed and constrained network resources. Moreover, on the one hand IT systems are increasingly embracing **reactive** and **event driven** architectures, which are typical of OT systems. On the other hand, OT systems are increasingly embracing **data analytics** (which is typical of IT systems) for production monitoring and forecasting (e.g., Industry 4.0).

The convergence between IT and OT is hence creating an increasing need to integrate the world of **data-at-rest** (storage, queries) with that of **data-in-motion** (pub/sub), notably in Edge and Fog Computing scenarios where decentralisation, scalability, efficiency and location transparent access to geo-distributed data is of paramount importance. This is where the **Data Analytics Support Stratum (DASS)** conceived and designed in 5G-DIVE comes into play.

While the IT world already offers a wide set of solutions for data management, these suffer from several limitations [74] when applied either to (1) Edge and Fog environment or (2) OT systems:

1. There is **sufficient bandwidth** to push data to the Cloud. E.g., a smart factory could produce several TB of data per day which should be pushed to the Cloud every day.
2. **Connectivity** is **not an issue**. A device will (almost) always be connected to the cloud. Remote locations (e.g., oil fields, mines, open fields, etc.) are extremely hard to support.
3. The **latency** induced by cloud-centralised **analytics** and **control** is compatible with the system's dynamic. E.g., autonomous vehicles, drones, etc. require very low latency and jitter.
4. The **connectivity cost** is **negligible.** E.g., cost of connectivity is an issue in Smart Grids as the operator must pay for the 2G/3G/4G datalink**.**
5. **Privacy** sensitiveness of storing their data to the Cloud**.** E.g., storing industrial processes data.

An example of such limitations is illustrated in Figure 8-1 on the left. The figure illustrates an exemplary cloud-based smart home scenario, which is eventually managed by a domestic operator (or application provider). Here, all the data are produced locally and then pushed to the cloud subject to all the limitations listed above. Nevertheless, all the data are centralized in this case allowing the users and/or the service provider to access them from anywhere.

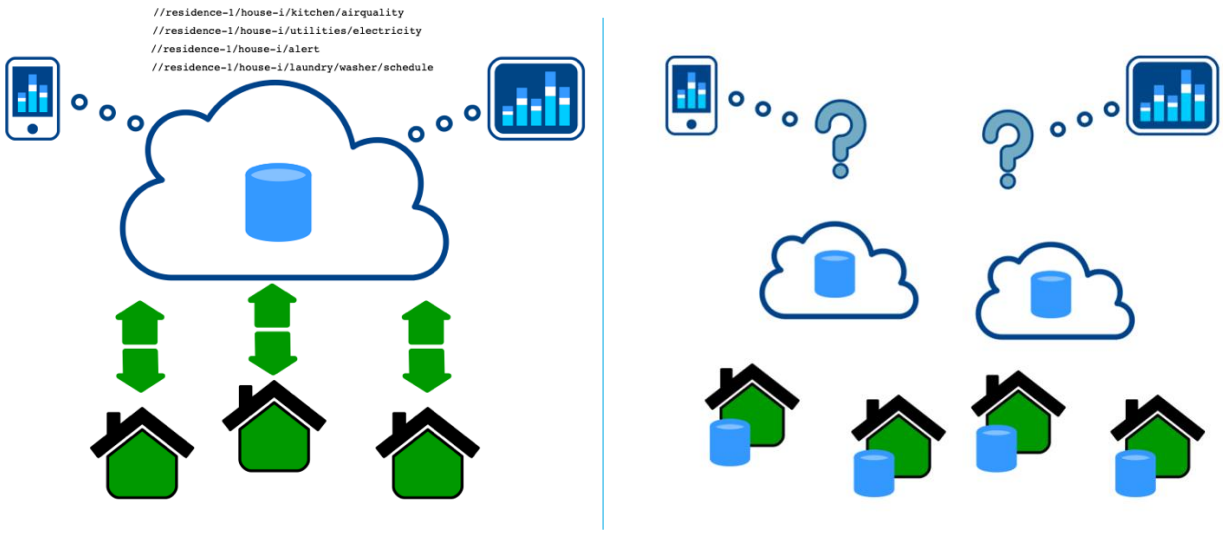


FIGURE 8-1: CLOUD-CENTRIC ISSUES (ON THE LEFT) AND EDGE/FOG ISSUES (ON THE RIGHT).

As the number of connected homes increases, the demand for computing, storage and networking resources increases too forcing therefore the service provider to create a second cloud-centric instance in order to support the demand increase. A possible solution is to keep local all the data as illustrated in Figure 8-1 on the right. However, such distribution creates a scattered data storage which is hard to access in a transparent manner regardless of their geographical location. To tackle this problem, it is necessary to get into patchwork design where multiple protocols and storages are stitched together to provide some meaningful end-to-end semantics. However, such operation is very costly, prone to errors due to the patchwork design, and hard to manage. **It is because of the above reasons that in 5G-DIVE we adopt a clean state approach for the DASS, designing from scratch a data platform and protocol capable of natively support the harmonization of data at rest and data in motion over a decentralized and distributed edge/fog environment, along with the necessary support and hooks to perform data analytics operations.**

8.2. Named Data Networking

DASS adopts **Named Data Networking (NDN)** [75]paradigm to support access transparency of both **data at rest** and **data in motion**. In NDN, applications name data and data names will directly be used in network packet forwarding; consumer applications request desired data by its name. In order to publish data in the network, a value (i.e., the data payload) is associated to a key (i.e., the data name), making de-facto a system based on key-value semantics. NDN, and inherently the DASS, provides a **routing** infrastructure capable of delivering the right data in the right place based on the set of available publishers and subscribers.

Based on the limitations shown in Figure 8-1, an example of how NDN is beneficial to data transparent access is provided in Figure 8-2. In this example, each home is provided with a local storage in addition to some regional storage (e.g., city). By carefully designing the key space (i.e., the key structure

associated to the data), it is possible to aggregate and transparently access the data via specific queries and selectors. In this example, the **keys** take the structure of a **Uniform Resource Identifier (URI)**, which represent a **hierarchical** organization of data. For example, each region comprises several houses identified by unique IDs. This results in a key structure like */region01/house01/*. Moreover, each data produced by each house, can be stored in a specific key, e.g. */region01/house01/room01/temperature* can be used to store the temperature reading of a specific room.

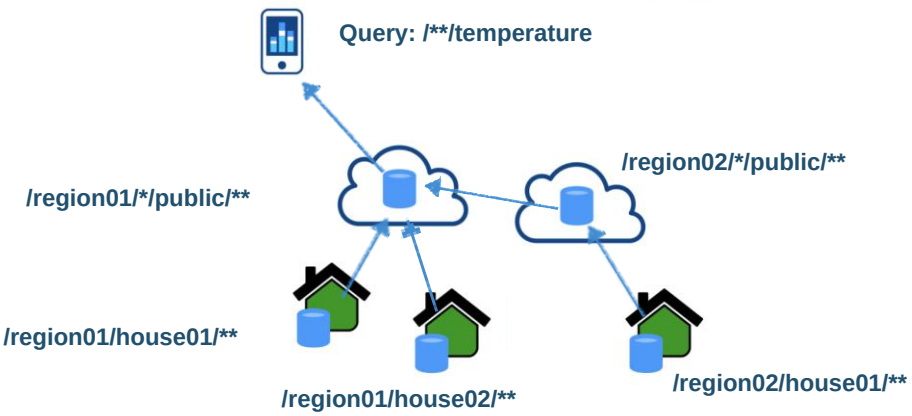


FIGURE 8-2: ADVANTAGES OF NDN APPROACH.

Data can be transparently accessed by the careful usage of **selectors** over the **key space**. For example, the wildcards in the key */region01/*/*/temperature* produce as result that the temperature of every room of every house in region01 is returned, regardless where they are stored. The routing infrastructure takes care of doing the necessary pattern matching between keys, selectors, publishers, and subscribers. By properly designing the key space, it is also possible to achieve the desired level of **privacy** for the data. E.g., data meant to be publicly available could be stored under a specific path (e.g., */region01/*/public/***) and stored only on specific public locations (e.g., regional data centres). Once data is successfully retrieved, **data analytics** can be eventually performed.

8.3. DASS Protocol Design

As described in D1.1[76], DASS comprises three main functionalities: Data Dispatch, Data Pre-processing, and Data Storage. The data dispatcher is responsible for gathering information from the different sources. The data pre-processing comprises the different tasks required to transform raw data into a proper, understandable, and common format. Finally, the data storage is responsible to keep the collected information provided by the different sources.

To provide the above functionalities, DASS needs to put in place a high-level API for pub/sub and distributed queries, data representation transcoding, an implementation of geo-distributed storage and distributed computed values. In addition, DASS requires primitives for efficient pub/sub and distributed queries, supporting fragmentation and ordered reliable delivery. Finally, the DASS requires an adaptive and fault tolerant routing infrastructure to properly dispatch geo-distributed data. It is

worth noticing that the initial design of the DASS described below has been contributed and integrated into the Eclipse Zenoh open source project[77]. The terms DASS and Zenoh are therefore used interchangeably below since Zenoh is an actual implementation of DASS. In order to support a wide heterogeneity of scenarios, networks, and devices, we adopt a two-level protocol design as illustrated in Figure 8-3:

- **Zenoh:** provides a high-level API for pub/sub and distributed queries, data representation transcoding, an implementation of geo-distributed storage and distributed computed values. DASS' data pre-processing and data storage functionalities are provided by Zenoh.
- **Zenoh.net:** Implements a networking layer capable of running above a Data Link, Network or Transport Layer. This protocol provides primitives for efficient pub/sub and distributed queries. It supports fragmentation and ordered reliable delivery. DASS' data dispatcher functionality is provided Zenoh.net.

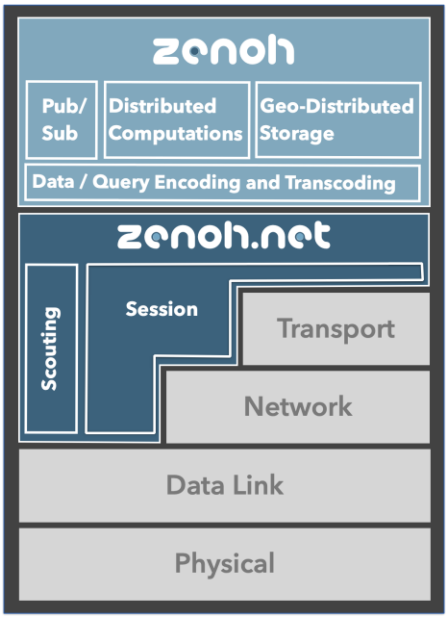


FIGURE 8-3 DASS/ZENOH PROTOCOL STACK.

The separation in two sub-protocols allows for better decoupling of responsibilities and thus to enhance the overall protocol scalability and flexibility. **Zenoh.net** creates the overlay routing infrastructure capable of operating across multiple types of networks (e.g., Ethernet, 5G/4G, Bluetooth, etc.) and compliant with the DASS key concepts. **Zenoh** oversees the exposure of a high-level API focusing on the end-to-end semantics of data management.

In the following, we report the design and progress on Z**enoh.net**, which has been the primary focus of the work undergone during the first year of 5G-DIVE WP2. The design and implementation of **Zenoh** will be addressed in the second year.

8.4. Zenoh.net Abstractions and APIs

Zenoh.net is the protocol layer implementing the data dispatcher functionality in the DASS. It provides four key **abstractions** as shown in Figure 8-4:

1. **resource:** a **named data** (i.e., a key-value). An example of resource is given by the (key, value) := (*/house/room/temp, 25°C*);
2. **publisher**: a **spring** of values for a key expression. It is usually an entity (e.g., an application) publishing values associated to a given resource. E.g., the application publishing the temperature value associated to the resource */house/room/temp*;
3. **subscriber**: a **sink** of values for a key expression. It is usually an entity (e.g., an application) subscribing to a given resource (e.g., */house/room/temp*) that has interest in any new value associated to it. A subscriber can be either **push** or **pull**. A push subscriber receives data as produced. For a pull subscriber, the infrastructure caches the data, as close as possible to allow the subscriber to consume it effectively when ready to pull. **Time schedules** can be negotiated for both push and pull subscribers;
4. **queryable**: a **well** of values for a key expression. It is usually an entity (e.g., an application) that is associated to a resource and that can produce data on demand. E.g., querying */house/room/temp* will return the current temperature value *25°C*.

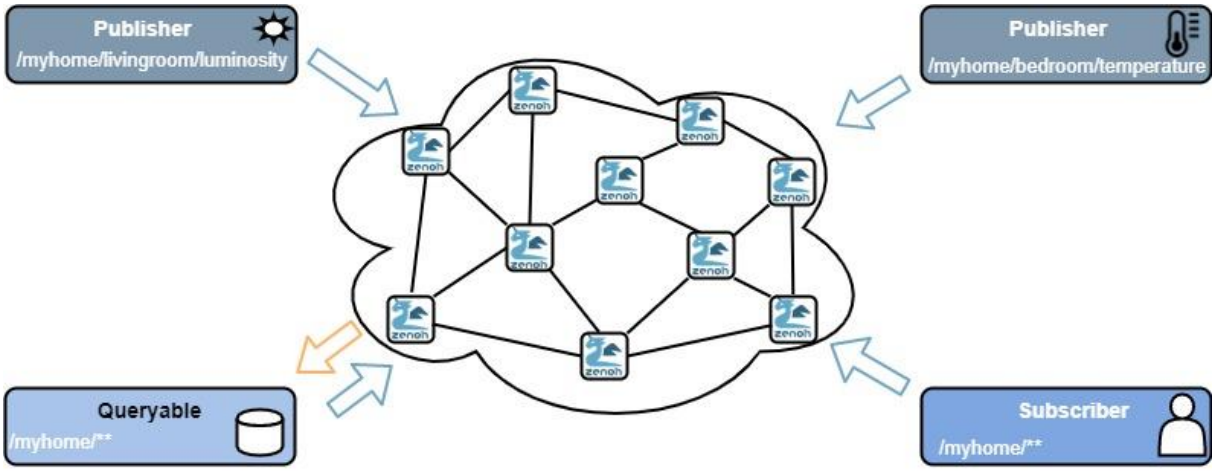


FIGURE 8-4: ZENOH.NET MAIN ABSTRACTIONS WITH ROUTING INFRASTRUCTURE.

These fundamental abstractions can be natively combined to implement a **storage** abstraction. Specifically, a storage abstraction is achieved by combining a **subscriber** and a **queryable** in a single application. E.g., a database (e.g., MariaDB, InfluxDB, etc.) can subscribe to any resource in such a way they can be automatically stored upon publication. The stored values can be then retrieved by issuing a query on the zenoh.net network. Please note that the high-level API for storage is in the scope of the zenoh protocol and not in the lower level zenoh.net protocol.

The above key abstractions are used to build and expose the following key **primitives:**

- **scout:** it performs a discovery on the network looking for other zenoh.net entities (e.g., routers or peers). Once an entity is discovered, it is possible to establish a zenoh.net session.
- **open/close:** it opens/closes a zenoh.net session (e.g., with a router or a peer).
- **declare/undeclare:** it declares/undeclares a resource, a publisher, a subscriber or a queryable. Declarations are used for discovery and various optimisations. For subscribers, the declare primitive registers a user provided call-back that will be triggered when data is available. For queryable, the declare primitive register a user provided call-back triggered whenever a query needs to be answered. The (un)declare primitives can be seen as the *(un)subscribe* operation.
- **write:** writes data for a key expression. It can be seen as the *publish* operation.
- **pull:** pulls data for a pull subscriber.
- **query:** issues a distributed query and returns a stream of results. The query target, coverage and consolidation depend on configured policies.

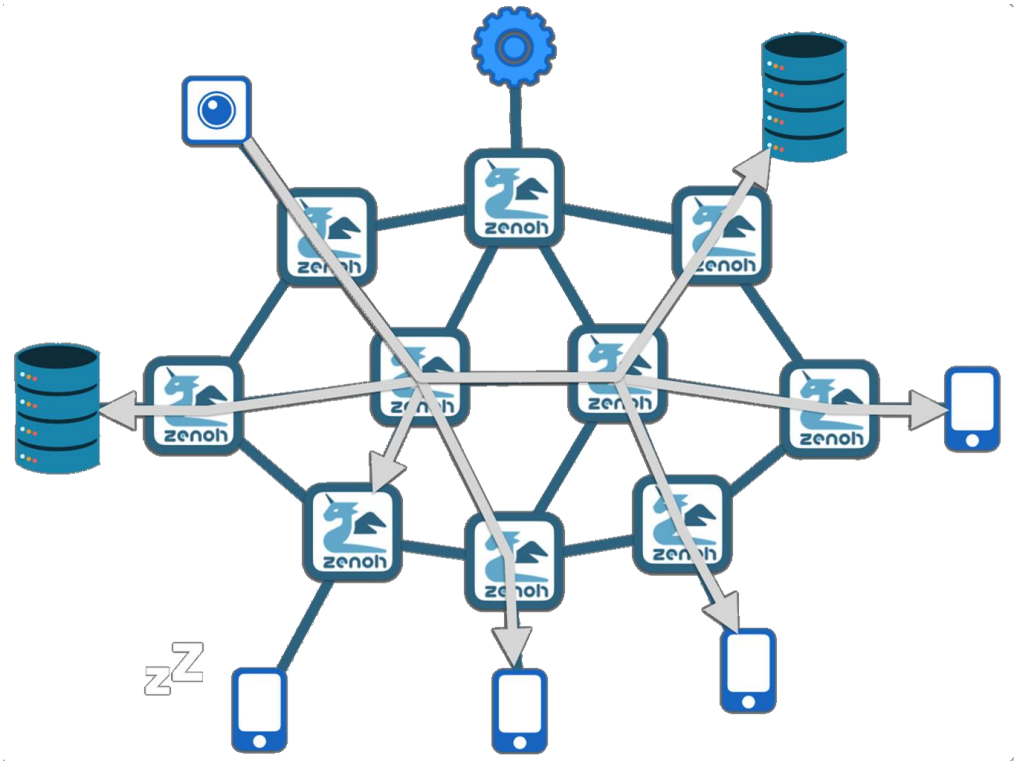


FIGURE 8-5: EXAMPLE OF DATA PUBLICATION AND SUBSCRIPTION IN ZENOH.NET.

Figure 8-5 shows an example of data publication with multiple subscribers, including data storages. In this example, a sensor (e.g., a camera) publishes data (e.g., a video frame) via zenoh.net protocol. The video frame reaches the first zenoh router, which routes and delivers the video frame to every subscriber in a multicast-like fashion. These frames can either directly delivered to a device or cached into the network in case the device is in sleep mode. In this example, two storages are subscribed to the video stream, enabling seamless storage redundancy without requiring any ad-hoc configuration.

8.5. Zenoh.net reliability and channels

Zenoh.net offers three levels of reliability as shown in Figure 8-6:

- **Application-to-application:** data is explicitly acknowledged at application level. This is the strongest type of reliability since data is ensured to be properly delivered at application level. However, it is very costly since it requires a significant amount of state to be kept at application level. It may suffer from scalability problems on the publisher side when the number of subscribers is very large. Finally, given the explicitly end-to-end communication required at application level, it may not be suitable for extremely latency sensitive application.
- **First-to-last broker:** it is similar to application-to-application reliability, but more scalable since it limits the scope to the routing infrastructure. Scalability depends on the number of edge brokers instead of the number of subscribers. The applications are not directly involved in the reliability mechanism. First-to-last broker reliability is useful when an error occurs in the network graph (e.g., a router disappears) and re-routing needs to take place.
- **Hop-to-hop:** it ensures reliability and ordering when there are no failures in the network graph. It is the simplest and most efficient type of reliability capable of coping with transmission errors and out-of-order delivery between two zenoh.net entities (e.g., peers, routers, client-router). This kind of reliability is provided by the reliable channel.

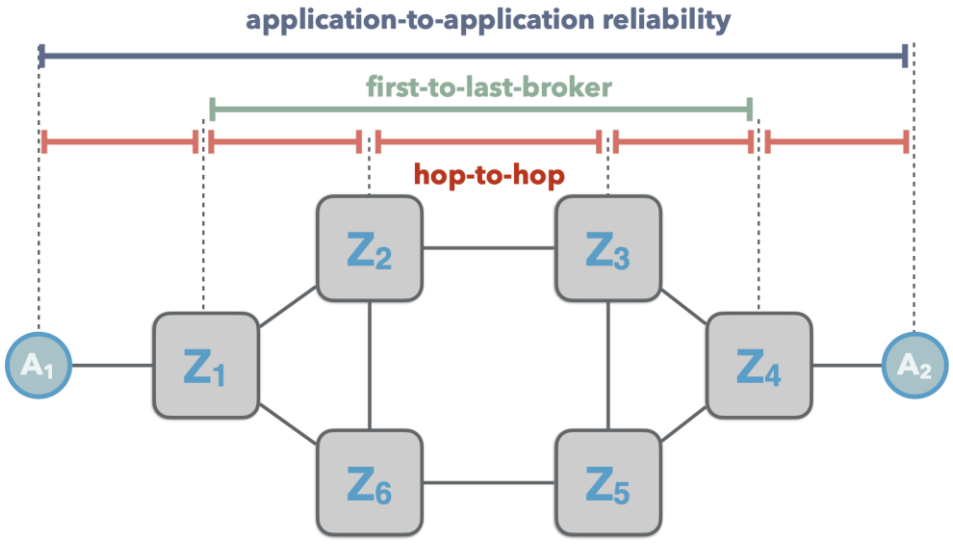


FIGURE 8-6: ZENOH.NET RELIABILITY MODELS.

Zenoh.net provides two **logical channels** for publishing data. These channels have hop-to-hop scope but their semantic is end-to-end, notably application-to-application. The channels are:

- **Reliable:** it actively provides a retransmission and reordering mechanism at zenoh.net level to cope with transmission error. It implements the hop-to-top reliability.

- **Best-effort:** it provides no retransmission and reordering mechanism in zenoh.net. However, reliability and reordering may be provided by the underlying network when e.g. operating zenoh.net on top of a reliable transport protocol like TCP or similar.

8.6. Zenoh.net routing

Zenoh.net routing is based on a combination of link-state approach and multicast distribution trees [78] offering an adaptive and fault tolerant data dispatching. Routing occurs in two phases:

1. **Network graph construction:** routers use the **scout** primitive to discover and connect to each other. Once a new router is added to the network, it transmits the information about its **neighbours** to any discovered routers. This information is then propagated throughout the network via a **link state** approach. By doing so, each router has a global view of the network and it can locally compute the optimal path for each resource.
2. **Routing:** given the NDN approach adopted by Zenoh, routing needs to be done through multicast distribution trees to avoid routing loops and congesting the network. Based on the network global view, each router computes locally a set of multicast distribution trees associated to each known resource subject to the set of known publishers and subscribers.

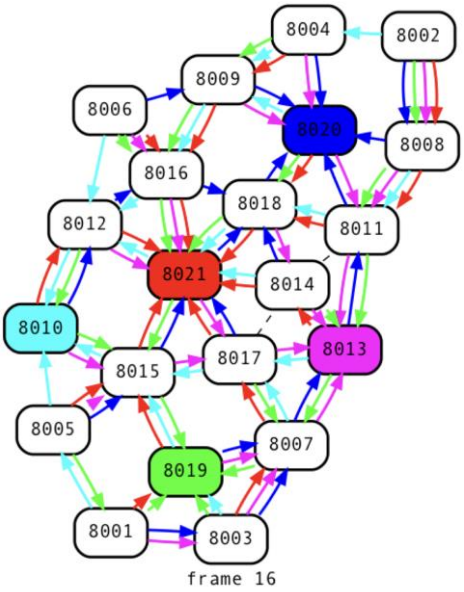


FIGURE 8-7: EXAMPLE OF ZENOH.NET ROUTING AND MULTICAST DISTRIBUTION TREES.

Figure 8-7 shows an example of complex zenoh.net routing network where each node is a router (shown with a unique ID) and each colour is a distinct multicast distribution tree. The coloured nodes are routers that act as a root of the multicast distribution trees while the coloured arrows are the links covered by those. In order to improve scalability and/or reduce the computational requirements on the routers, the number of multicast distribution trees can be limited, although leading sometimes to sub-optimal routing.

8.7. Zenoh.net wire protocol format

For the sake of space, we do not report here the full wire protocol definition, which is directly available in the Zenoh open source repository[79]. Instead, we briefly report on the relevant design choices taking as an example two specific messages: the zenoh.net **Data** and **Frame** messages. The role and scope of Data and Frame messages is the following:

- **Data:** is the message containing the application data and its scope is application-to-application. Every time a data is published, its content is encapsulated in a data message and delivered to the subscribers.
- **Frame:** is the message encapsulating Data messages for hop-to-hop transmission. A Frame can contain either (1) Data message fragment or (2) one or more complete Data messages. The maximum MTU of the Frame depends on the underlying network. In case the Data message is larger than the network Maximum Transmission Unit (MTU), the Data message needs to be fragmented in several Frame messages. Conversely, when multiple small Data messages are ready to be transmitted, these can be batched into a single Frame message to reduce the network overhead and increase performance.

In zenoh.net, each message has 1-byte fixed header indicating the type of message and eventual flags. Specifically, 5 bits are used for indicating the message type and 3 bits for optional flags. For wire-efficiency purposes, zenoh.net uses a **variable length encoding** (VLE) for representing integer values (e.g., sequence numbers, IDs)[80]. This allows to reduce the number of bytes on the wire to represent a small number while keeping the relevant resolution. For instance, the value 0 with a resolution of 64-bits, can still be represented on the wire by using only one byte.

Zenoh.net provides a minimal wire overhead of just 4 bytes for user data as follows:

- Data messages can either contain the full resource URI or a mapping of this URI to a VLE-encoded ID for performance optimization. In this case, the resource ID can be as small as 1 byte. As a result, the minimal overhead introduced by Data messages is 2 bytes.
- Frame messages needs to contain a sequence number (SN) for fragmentation and reordering purposes. The SN resolution can be negotiated when establishing a zenoh.net session. By doing so it can be ensured that the SN always takes 1 byte on the wire. As a result, the minimal overhead introduced by Frame messages is 2 bytes.

Since Data messages are always encapsulated within Frame messages, this leads to a minimal wire overhead of 4 bytes. This results in a very high wire efficiency, especially when transmitting a large amount of small data messages.

9. Appendix – Survey of Relevant Intelligence Techniques

9.1. Artificial Neural Networks

Artificial Neural Networks (ANNs) are learning techniques which artificially created as computing systems inspired by biological neural networks. A Neural Network algorithm contains similar actions and what the biological neural network works, where a collection of connected units called artificial neurons are inter-connected and capable to transmit and receive signals between neurons. Technically, ANN perceive the signals as real numbers and has each neuron to compute the collected signals and translate them into an insightful information. Moreover, ANN techniques fully adopt the complexity of the neural networks and allow them to have some hidden layers, called Deep Neural Networks (DNNs), resulted in more powerful learning capability with higher accuracy and precision. They can also continuously learn and improve, by making use of several learning strategies like supervised learning, unsupervised learning, and reinforcement learning. Artificial Neural Networks (ANNs) can be further classified into different types, such as Feed-forward Neural Networks (FFNN), Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN). In the next paragraphs a set of the most common ANNs will be explained.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are one variant of DNNs which are most applied in the visual imagery related applications, such as, an object detection algorithm to detect person in captured images. Such object detection algorithms have its state-of-the-art techniques divided into two type of approaches: (1) One-stage approaches which prioritize inference speed [81][82]and (2) Two-stage approaches [83][84] which prioritize detection accuracy. The stages consist of, first, a Region Proposal Network to generate region of interests, then, use these region proposals for the object classification and bounding-box regression. In one-stage approaches, the second stage is ignored and has the object detection threated as a simple regression problem. Such approaches are resulting trade-offs between the inference time (faster yet lower accuracy) and detection accuracy (higher accuracy yet slower).

LSTM-RNN

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture which has feedback connections, commonly called loops, allowing information to persist. This way, the loops act as memory functions.

Leveraging this feature, LSTM Networks are capable of learning long-term dependencies and remembering for long periods of time. Besides, LSTM is well-suited to make predictions and classifications based on time-series data as there could exist lags of unknown duration between important events in the time series data and is able to be used to process either single data (i.e. images) or entire sequences of data (i.e. video).

LSTM has been used for different purposes that include SLA enforcement in SDN and NFV, prediction of the size of the different flows when flow counters cannot be placed at the switch or prediction of the traffic matrix using the traffic matrix itself and the current network states [85][86].

MLP-FFNN

Multilayer perceptron (MLP) is a subset of feed-forward neural networks which are included and applied to supervised learning. The information flow in MLPs is unidirectional, so a particular unit sends information to other units from which it does not receive any information. They are used in pattern generation, recognition, regression or classification and they have fixed inputs and outputs.

Leveraging the features of MLP used in forecasting problems, the MLP has been used inside SDN and NFV with different purposes that include, for example, the prediction of the traffic matrix or the detection of the port scan attacks based on Packet-In messages and STATs reports [87].

Both LSTM and MLP are well-suited to be used in our I4.0 and ADS use cases. In all of the use cases, a similar approach can be taken to address the resulting degradation of QoS inside the network, maximizing the throughput and minimizing the mean delay. This would be a potential differentiating element in the successful outcome of the I4.0 and ADS use cases by predicting different network parameters based on resource usage that also could be predicted in 5G-DIVE computing elements.

9.2. Reinforcement Learning

Reinforcement Learning (RL) is a learning technique that can adapts its behavior according to its environment. RL learns by continuously interacts with the environment. A standard RL model has 2 entity: Agent and Environment. In each time step, an Agent can interact with the Environment. The Agent will observe for Reward (r) and State (s) from the Environment. The Agent then decides on which Action (a) to take based on its Policy (p). This Policy is the component that we train. The value of the policy is updated and learned continuously via trial and error. Ultimately, Policy is the behavior that we want the Agent to learn or to have. The optimal policy will be the one that maximizes the Reward in the long run.

Q-Learning

In Q-learning, the agent learns the optimal behavior in an unknown environment from sampled experiences. It is a value-based method, learning how good it is to take a certain action for a particular state of the environment by estimating the return (cumulative reward) for that action. In general, we want the agent to act greedily with respect to its known environment, while also exploring alternate actions in the unknown environment. In order to balance this exploration and exploitation problem in RL, Q-learning uses off-policy learning. This allows the agent, to learn from observing other agents or previous experiences generated from different policies.

But the lack of generalization and dependency on the actual value for each state, action pair in Q-learning, makes it difficult to scale up Q-learning based agents to larger problems. For environments

with large number of states and actions, the memory requirements and number of samples needed for the agent to learn the correct policy also scales up, making Q-learning infeasible for these problems.

The use of function approximators for value function approximation alleviates these issues. With these techniques, instead of updating the value function of a state directly, the weights of the approximators are adjusted towards the new experienced samples. Deep Q-Networks (DQN) has become the preferred choice for approximating the value function. Although, convergence is not guaranteed, recent techniques such as "experience-relay" and "quasi-static target networks" [88] help in ensuring convergence of these methods.

Yu et al [89] presents the use of DQN in managing medium access in presence of heterogenous wireless networks. They formulate the problem of sharing timeslots among multiple networks and ensuring fairness in the medium utilization across multiple nodes as a RL problem. Using DQN networks with experience relay and quasi-static target networks, the paper achieves α-fairness objective and optimal throughput, showing the RL agent is able to coexist harmoniously with heterogeneous networks with specified objective.

Xu et al [90] presents the use of DQN in dynamic power allocation for multiple remote radio heads in a cloud-RAN scenario. The agent reduces the power consumption by learning the optimal beamforming parameters across multiple remote radio-heads. The agent is able to handle dynamic power allocations while keeping up with the user demand.

Deep Q Network

The Deep Q Network (DQN) is a variant of the Q-Learning technique. Both are part of the RL technique. Similar to Q-Learning, DQN Policy is evaluated based on the Q value. The main differences lie in the use of a Neural Network (NN) to approximate the Q value.

In general, DQN is suitable for solving problem that has high number of inputs, problem that is dynamic and changes over time, problem where prior knowledge of the environment is unavailable, and problem where absolute optimal solution is not a must. Although with that said, DQN technique also has its own challenges for adoption. First, DQN solution is tied to the base environment that it is trained on. This means that for every problem, a custom build or modification of the Environment is needed. DQN based solution for one system, does not necessarily means that it will also works for other system. Second, the optimal Policy derivation is time consuming. This is because that the Agent learns only through trial-and-error approach. Third, DQN relies on NN for optimal policy approximation. This itself can be tricky since there are no exact guidelines available on how to design the NN itself. And finally, designing an adequate Reward system can be challenging. A poorly designed reward system can cause failure in training and resulted in inadequate performance.

One interesting use case for DQN is adopting it on Edge Data Center (EDC) use case. DQN can be used to create an intelligent agent that can perform internal resources load balancing and for performing user task scheduling [92]. The motivation behind this is because of the DQN ability as an intelligent system that is adaptable to changes. DQN can be very flexible and is able to cope with environment

variability. For example, variation in user task complexity, and variation in wireless communication channel quality. Moreover, DQN agent can continuously learn throughout its operation. Another motivation is because DQN is able to find for a near optimal solution where traditional optimization theory method fails.

9.3. Imitation Learning

Imitation Learning (IL) [93] is part of the supervised learning family. Here, instead of learning a single action, an agent learns to perform a task (sequence of actions) guided by demonstrations (or an active demonstrator). A huge amount of demonstrations (D) is needed, which will be then translated into state/action pairs (s,a) and on top of that supervised learning or reinforcement learning are applied to find a policy (p) that minimizes the loss (L(s,a)) from the expert demonstrations.

Usually, IL is applied to Markov Decision Processes (MDP) where the reward function is not straightforward. There three main techniques to implement IL:

1) Behavioural Cloning: is the simplest IL algorithm, it is based on finding a policy (using supervised learning techniques) to minimize the 1-step deviation from the expert demonstration. In this way, the agent is able to follow the expert step by step. The main advantages of using behavioural cloning is the simplicity of implementation, on the other hand unpredictable behaviours can occur if the agent visits a state not covered by the expert demonstrations.
2) Direct Policy Learning: it is a generalized version of the behavioural cloning. Here, after finding the policy as in behavioural cloning, the agent rollouts the policy found in the environment and queries an active demonstrator to fix the errors in the policy. In the end, the model is training with the old dataset combined with the new dataset received by the demonstrator.
3) Inverse Reinforcement Learning: follows a different approach from the previous two techniques. Differently from RL, here the aim is to find a reward function. It starts by estimating a reward function as a linear combination of the known parameters, then reinforcement learning is applied to find the optimal/sub-optimal policy for the selected reward function. The policy found is compared with the one from the expert, in this process the reward function is updated and the cycle starts again. The iterations stop when the loss between the policy found and the expert demonstrations are smaller than epsilon.

In the recent years, IL has gathered much attention by both industry and academy. It can be very useful for applications like autonomous driving cars (ref driving), smart manufacturing and autonomous drone navigation[94]. Moreover, IL has been combined with others AI/ML techniques to achieve better learning rate and accuracy. As an example, the combination of IL with Domain Randomization meta learning techniques, gives the IL agent the ability of "learning how to learn". In fact, many works are done in order to infer to robot and robotic manipulators the ability to execute tasks with few demonstrations (e.g. by using video, images, digital twins) leveraging the use of domain randomization meta learning [95][96] [97][98].

10. Appendix – Survey of AutoAI/ML Tools

In this appendix, we provide a survey of the different AutoAI/ML tools and/or techniques which will be used as part of the proposed IESS described in section 2.3.3. This survey can be seen in Table 10-1.

TABLE 10-1: SURVEY OF AUTOAI/ML TECHNIQUES.

Project	Pre-process.	Distrib comp.	Interfacing options	GUI	Usage	GPU accel.	K8s Integ.
H2O[2]	Yes	Yes*	R, Python, Scala, Java, JSON, Flow notebook, Hadoop, Spark	Yes**	GBM, Random Forest, Deep Neural Networks, Word2Vec and Stacked Ensembles	Yes***	Yes*
NNI[3]	Yes	Yes	CLI, Web, Python	Yes	Feature Engineering, Neural Architecture Search, Hyperparameter Tuning and Model Compression.	Yes	Yes*
AutoKeras[4]	No	No	Python	No	Image classification and regression, Text classification and regression Structured data classification and regression, Multi modal and multitask	Yes****	N/A
TPOT[5]	No*****	Yes+	CLI, Python	No	Classification and Regression	No	N/A

[2] https://github.com/h2oai/h2o-3, https://github.com/h2oai/h2o-kubeflow
[3] https://github.com/Microsoft/nni
[4] https://github.com/keras-team/autokeras
[5] https://github.com/EpistasisLab/tpot

TransmogrifAI[6]	Yes	Yes++	Scala, Spark Shell, Jupyter Notebook	No	Classification and Regression	No	N/A
Project	**Pre-process.**	**Distrib comp.**	**Interfacing options**	**GUI**	**Usage**	**GPU accel.**	**K8s Integ.**
MLBox[7]	Yes	No	Python	No	Classification and Regression with auto feature selection and hyperparameter tuning	No	N/A
Advisor[8]	No	No	API, SDK, WEB and CLI	Yes	Hyperparameter tuning	No	Yes
AutoWeka[9]	No	No	Java, GUI, CLI	Yes	Tabular models and Hyperparameter tuning	No	N/A
AdaNet[10]	No	Yes+++	Python	Yes+++ +	Neural networks architecture search for regression and classification	Yes+++++	N/A
Auto-SKLearn[11]	No	No	Python	No	Classification and Regression	No	N/A
Featuretools[12]	Yes^	Yes+	Python	Yes	Feature engineering	No	N/A

[6] https://github.com/salesforce/TransmogrifAI
[7] https://github.com/AxeldeRomblay/MLBox
[8] https://github.com/tobegit3hub/advisor
[9] https://github.com/automl/autoweka
[10] https://github.com/tensorflow/adanet
[11] https://github.com/automl/auto-sklearn
[12] https://github.com/FeatureLabs/featuretools

AutoGluon[13]	Yes	Yes	Python	No	Text classification, Object detection, Tabular prediction and Image classification	Yes+++++	N/A
Automl-gs[14]	Yes^^	No	Python, CLI	No	Deep Learning	Yes+++++	N/A
Project	**Pre-process.**	**Distrib comp.**	**Interfacing options**	**GUI**	**Usage**	**GPU accel.**	**K8s Integ.**
Ray[15]	No	Yes	Python, HTTP (With Ray-Serve)	Yes ++++	Distributed tasks, Reinforcement Learning, Neural Networks and Hyperparameter Tuning	Yes+++++	Yes
Analytics-Zoo[16]	No	Yes++	Java/Scala, Python	No	The AutoML uses Ray for Time Series Forecasting	Yes	Yes
ATM[17]	No^^^	No	Python, CLI, REST API	No	Best model selection for classification	No	N/A
MindsDB[18]	No^^^^	No	Python, GUI	Yes	Prediction, Classification and Time series support	Yes+++++	N/A
AutoViml[19]	Yes	No	Python	No	Prediction	No	N/A
Aikit[20]	Yes	No	Python	No	Classification and Regression	No	N/A

[13] https://github.com/awslabs/autogluon
[14] https://github.com/minimaxir/automl-gs
[15] https://github.com/ray-project/ray
[16] https://github.com/intel-analytics/analytics-zoo
[17] https://hdi-project.github.io/ATM/
[18] https://github.com/mindsdb/mindsdb
[19] https://github.com/AutoViML/Auto_ViML
[20] https://github.com/societe-generale/aikit

Aethos[21]	Yes	No	Python	No	Classification and Regression	No	N/A
Katib[22]	No	Yes	Python	Yes	Hyperparameters tuning and Neural architecture search	No	Yes

* Using Kubeflow.

** Using H2O Flow.

*** Using CUDA through H2O4GPU.

**** Using Tensorflow with CUDA support.

***** Only Feature Selection.

+ Using Dask.

++ Using Apache Spark

+++ Using TensorFlow

++++ Using TensorBoard

+++++ Using CUDA.

^ Requires Structured Data.

[21] https://github.com/Ashton-Sidhu/aethos

[22] https://github.com/kubeflow/katib

^^ Built-in ETL.

^^^ Requires data in CSV format.

^^^^ Only offers insights about data columns.

Then, once described in detail the different AutoAI/ML platforms and tools which could be part of the IESS. Table 10-2 explains which of them have been pre-selected to be used as part of the IESS platform which is going to be developed in this project.

TABLE 10-2: AUTOAI/ML TECHNIQUES PRE-SELECTION.

Project	Pre-selection	Explanation
H2O	**Yes**	It has a large community, allows distributed computing, supports GPU and is available to be run in K8s. Lots of algorithms available and includes feature engineering.
NNI	**Yes**	The most complete software of the list at a first glance, although it does not support ARM architecture, but it even supports Windows workers.
AutoKeras	No	It does not satisfy the minimum requirements of the DEEP platform.
TPOT	**Yes**	Although it does not support GPU workloads, it is widely used and supports distributed computing. It is the to-go solution for classification/regression solutions.
TransmogrifAI	No	It does not satisfy the minimum requirements of the DEEP platform.
MLBox	No	It does not satisfy the minimum requirements of the DEEP platform.
Advisor	No	It does not satisfy the minimum requirements of the DEEP platform.
AutoWeka	No	It does not satisfy the minimum requirements of the DEEP platform.
AdaNet	No	It does not satisfy the minimum requirements of the DEEP platform.
Auto-SKLearn	No	It does not satisfy the minimum requirements of the DEEP platform.
Featuretools	No	It does not satisfy the minimum requirements of the DEEP platform.
AutoGluon	**Yes**.	Its strong points are that it supports GPU and ARM architecture. It handles very well the pre-processing, at the level of H2O
Automl-gs	No	It does not satisfy the minimum requirements of the DEEP platform.
Ray	**Yes**	Specifically built with workload distribution in mind. This platform is even used by other distributed computing platforms.
Analytics-Zoo	No	It does not satisfy the minimum requirements of the DEEP platform.
ATM	No	It does not satisfy the minimum requirements of the DEEP platform.
MindsDB	No	It does not satisfy the minimum requirements of the DEEP platform.
AutoViml	No	It does not satisfy the minimum requirements of the DEEP platform.
Aikit	No	It does not satisfy the minimum requirements of the DEEP platform.
Aethos	No	It does not satisfy the minimum requirements of the DEEP platform.
Katib	No	It does not satisfy the minimum requirements of the DEEP platform.