



UNIVERSITY OF CATANIA

DEPARTMENT OF ELECTRICAL, ELECTRONIC AND  
COMPUTER ENGINEERING

PhD IN SYSTEM, ENERGY, COMPUTER AND  
TELECOMMUNICATIONS ENGINEERING

XXXIII CYCLE

---

**Network softwarization and smart  
services in 5G ecosystems**

---

CHRISTIAN GRASSO

Coordinator: Prof. Paolo Pietro Arena

Tutor: Prof. Giovanni Schembra

Company Tutor: Xenia Network Solutions



# Contents

<b>List of Terms and Abbreviations</b>	<b>iii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xiv</b>
<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Enabling technologies for 5G networks</b>	<b>7</b>
2.1 Network Function Virtualization (NFV) . . . . .	7
2.2 Software Defined Networking (SDN) . . . . .	14
2.2.1 SDN-NFV integration . . . . .	17
2.3 Network Slicing . . . . .	19
2.4 Cloud Technology . . . . .	23
2.4.1 Cloud vs Edge . . . . .	23
2.4.2 Multi-Access Edge Computing . . . . .	29
<b>3 Management and orchestration of network slices</b>	<b>35</b>
3.1 5G-Handler for handover detection . . . . .	36
3.1.1 Definition and implementation . . . . .	39
3.1.2 Testbed description and results . . . . .	41
3.2 UAV for Slice extension . . . . .	47
3.2.1 UAV for Monitoring System applications . . . . .	48
3.2.2 Analytical System Model and results . . . . .	53
3.2.3 Event-based Matlab simulation and results . . . . .	59
3.2.4 MEC UAV managed with Reinforcement Learning . . . . .	65
<b>4 Network slicing for vertical applications</b>	<b>98</b>
4.1 The TSE for Tactile Internet . . . . .	98

## CONTENTS

---

4.1.1	Tactile Internet Architecture . . . . .	99
4.1.2	TSE Implementation . . . . .	101
4.2	Multi-Layer Offloading in Vehicular Networks . . . . .	114
4.2.1	Vehicular Networks . . . . .	114
4.2.2	Multi-Layer offloading and RL . . . . .	118
<b>5</b>	<b>5G-enabled smart services</b>	<b>131</b>
5.1	DiMoViS: Distributed Mobile Video Surveillance System . . . . .	132
5.1.1	Triangle Testbed . . . . .	132
5.1.2	DiMoViS architecture and performance evaluation . . . . .	135
5.2	TouristEyes: a 5G-based platform for blind Tourist . . . . .	146
5.3	5Gamer: a 5G-assisted online game . . . . .	160
5.4	VISION: a platform for smart-city video surveillance services . . . . .	167
5.4.1	Flame Project . . . . .	168
5.4.2	VISION architecture . . . . .	174
<b>6</b>	<b>Conclusion</b>	<b>187</b>
	<b>List of publications</b>	<b>189</b>
	<b>References</b>	<b>191</b>
	<b>Appendix</b>	<b>200</b>
Appendix A:	OpenFlow . . . . .	200
Appendix B:	SDN Controllers . . . . .	203

# List of Terms and Abbreviations

<b>AI</b>	Artificial Intelligence
<b>ANDSF</b>	Access Network Discovery and Selection Function
<b>AP</b>	Access Point
<b>API</b>	Application Programming Interface
<b>ASP</b>	Application Service Provider
<b>BIP</b>	Backhaul Infrastructure provider
<b>BBU</b>	BaseBand Unit
<b>BS</b>	Base Station
<b>CapEx</b>	CAPital EXpenditure
<b>CE</b>	Computing Element
<b>CFS</b>	Customer Facing Service
<b>CLMC</b>	Cross-Layer Management and Control
<b>CNIT</b>	National Inter-University Consortium for Telecommunication
<b>CoMP</b>	Coordinated MultiPoint
<b>C-RAN</b>	Centralized RAN
<b>DC</b>	Docker Containers
<b>DHCP</b>	Dynamic Host Configuration Protocol
<b>DMC</b>	Digital Media Controller
<b>DMS</b>	Digital Media Server

## LIST OF TERMS AND ABBREVIATIONS

---

<b>DNS</b>	Domain Name System
<b>DPI</b>	Deep Packet Inspection
<b>D-RAN</b>	Distributed RAN
<b>DT</b>	Digital Twin
<b>e2e</b>	end-to-end
<b>EA</b>	Edge Acquirer
<b>eICIC</b>	Inter Cell Interference Coordination
<b>eMMB</b>	Enhanced Mobile Broadband
<b>EN</b>	Edge Node
<b>eNB</b>	eNodeB
<b>EPC</b>	Evolved Packet Core
<b>EPDG</b>	Evolved Packet Data Gateway
<b>ES</b>	Edge Storage
<b>ETSI</b>	European Telecommunication Standards Institute
<b>E-UTRAN</b>	Evolved Universal Terrestrial Radio Access Network
<b>FANET</b>	Flying Ad-hoc NETWORK
<b>FCAPS</b>	Fault management, Configuration management, Accounting, Performance monitoring and Security
<b>FiaB</b>	Flame-in-a-Box
<b>FIFO</b>	First-In-First-Out
<b>FQDN</b>	Fully Qualified Domain Names
<b>FR-EA</b>	FlowReplicator EA
<b>FR-PA</b>	FlowReplicator PA
<b>FW</b>	Firewall
<b>GDPR</b>	General Data Protection Regulation

## LIST OF TERMS AND ABBREVIATIONS

---

<b>GUI</b>	Graphic User Interface
<b>GUMMEI</b>	Global Unique MME Identity
<b>HSI</b>	Human-System Interface
<b>HSS</b>	Home Subscriber Server
<b>IaaS</b>	Infrastructure as a Service
<b>IC</b>	Infrastructure SDN controller
<b>IDS</b>	Intrusion Detection System
<b>InP</b>	Infrastructure Provider
<b>IoT</b>	Internet of Things
<b>ISG</b>	Industry Specification Group
<b>ITS</b>	Intelligent Transportation System
<b>ITU-R</b>	International Telecommunication Union Radiocommunication Sector
<b>KPI</b>	Key Performance Indicators
<b>LB</b>	Load Balancer
<b>LOS</b>	Line-Of-Sight
<b>LPR</b>	Lying Person Recognition
<b>LSTM</b>	Long Short Term Memory
<b>LTE</b>	Long Term Evolution
<b>MANO</b>	NFV Management and Orchestration
<b>MCC</b>	Mobile Country Code
<b>MDP</b>	Markov Decision Process
<b>MEAO</b>	MEC Application Orchestrator
<b>MEC</b>	Multi-Access Edge Computing
<b>MEO</b>	MEC Orchestrator

## LIST OF TERMS AND ABBREVIATIONS

---

<b>MEPM</b>	MEC Platform Manager
<b>MEPM-V</b>	MEC platform manager-NFV
<b>MIMO</b>	Multiple-input and Multiple-output
<b>ML</b>	Machine Learning
<b>MME</b>	Mobility Management Entity
<b>MMORPG</b>	Mass Multiplayer Online Role-Play Game
<b>mMTC</b>	Massive Machine Type Tommunications
<b>MNC</b>	Mobile Network Code
<b>MPS</b>	MEC Platform Server
<b>NAT</b>	Network Address Translation
<b>NF</b>	Network Function
<b>NFV</b>	Network Function Virtualization
<b>NFVI</b>	Network Function Virtualization Infrastructure
<b>NFVI-POP</b>	Network Function Virtualization Infrastructure Point of Presence
<b>NFVO</b>	NFV Orchestrator
<b>NIST</b>	National Institute of Standards and Technology
<b>NN</b>	Neural Network
<b>NS</b>	Network Service
<b>NSD</b>	Network Service Descriptor
<b>NSO</b>	Network Service Orchestrator
<b>OBU</b>	On Board Unit
<b>ONF</b>	Open Network Foundation
<b>ONOS</b>	Open Network Operating System
<b>OpEx</b>	OPerating EXpenditure



## LIST OF TERMS AND ABBREVIATIONS

---

<b>OSM</b>	Open Source MANO
<b>OSS</b>	Operation/business support system
<b>PA</b>	Personal Acquirer
<b>PaaS</b>	Platform as a Service
<b>PCRF</b>	Policy and Charging Rules Function
<b>pdf</b>	Probability Density Function
<b>PGW</b>	Packet Data Network Gateway
<b>PIML</b>	Personalization, Interactivity, Mobility and Localisation
<b>PLMN</b>	Public LandMobile Network
<b>PNF</b>	Physical Network Function
<b>PoI</b>	Point of Interests
<b>PSNR</b>	Peak Signal-to-Noise Ratio
<b>QoE</b>	Quality of Experience
<b>QoS</b>	Quality of Service
<b>RAN</b>	Radio Access Network
<b>RL</b>	Reinforcement Learning
<b>RMSP</b>	RAN/MEC Service Provider
<b>RNN</b>	Recurrent Neural Network
<b>RO</b>	Resource Orchestrator
<b>RRH</b>	Remote Radio Head
<b>RSRP</b>	Reference Signal Received Power
<b>RSSI</b>	Received Signal Strength Indication
<b>RSU</b>	Road Side Unit
<b>SaaS</b>	Software as a Service

## LIST OF TERMS AND ABBREVIATIONS

---

<b>SBBP</b>	Switched Batch Bernoulli Process
<b>SC</b>	System Controller
<b>SCTP</b>	Stream Control Transmission Protocol
<b>SDN</b>	Software Defined Networking
<b>SeV</b>	Service Vehicle
<b>SFC</b>	Service Function Chain
<b>SFEMC</b>	Service Function Endpoint Management and Control
<b>SGW</b>	Security Gateway
<b>SMF</b>	Session Management Function
<b>TAP</b>	Test Automation Platform
<b>TaV</b>	Task Vehicle
<b>TC</b>	Tenant SDN Controller
<b>TO</b>	Telco Operator
<b>TSE</b>	Tactile Support Engine
<b>UE</b>	User Equipment
<b>UAV</b>	Unmanned Aerial Vehicles
<b>URLLC</b>	Ultra-Reliable and Low Latency Communications
<b>V2V</b>	Vehicular-to-Vehicular
<b>V2X</b>	Vehicular-to-Everything
<b>VC</b>	Virtualization Containers
<b>VCC</b>	Vehicular Cloud Computing
<b>VDI</b>	Virtual Decoder Interface
<b>VEC</b>	Vehicular Edge Computing
<b>VF</b>	Virtual Function

## LIST OF TERMS AND ABBREVIATIONS

---

<b>VIM</b>	Virtualized Infrastructure Manager
<b>VNFD</b>	Virtual Network Function Descriptor
<b>VM</b>	Virtual Machine
<b>VNF</b>	Virtual Network Function
<b>VNF-FG</b>	VNF Forwarding Graph
<b>VNFM</b>	VNF managers
<b>VPS</b>	Vehicular Platform Server
<b>vsSTB</b>	video-surveillance set-top box
<b>WAN</b>	Wide Area Network
<b>WIM</b>	WAN Infrastructure Manager

# List of Figures

1.1	Structure of this thesis . . . . .	6
2.1	Traditional network functions vs NFV paradigm [11] . . . . .	8
2.2	ETSI high level NFV framework definition [12] . . . . .	9
2.3	ETSI End-to-end Network Service definition [12] . . . . .	11
2.4	ETSI NFV Architecture definition [12] . . . . .	12
2.5	Traditional Networking vs SDN [19] . . . . .	14
2.6	Software Defined Networking . . . . .	15
2.7	SDN and NFV integration [26] . . . . .	18
2.8	Network slice . . . . .	20
2.9	Network slice architecture [29] . . . . .	21
2.10	Network slicing deployment integrating SDN and NFV [27] . . . . .	22
2.11	Cloud Architecture Model [30] . . . . .	24
2.12	Differences between a User Managed, IaaS, PaaS and SaaS deployment . . . . .	26
2.13	Cloud Computing and Edge Computing Architecture . . . . .	28
2.14	ETSI MEC framework [38] . . . . .	30
2.15	ETSI MEC reference architecture [38] . . . . .	31
2.16	ETSI MEC reference architecture in NFV [38] . . . . .	33
3.1	5G-Hander. Distributed RAN scenario . . . . .	38
3.2	5G-Hander. Centralized RAN scenario . . . . .	38
3.3	5G-Hander. Algorithm 1: Handover Sniffer pseudo-code . . . . .	40
3.4	SCTP packet structure . . . . .	41
3.5	SCTP Data chunk structure . . . . .	41
3.6	Header of the X2 and S1 packets . . . . .	41
3.7	5G-Hander. Testbed infrastructure . . . . .	42
3.8	5G-Hander. Network topology of Backhaul emulator . . . . .	43
3.9	5G-Hander. Experiment scenario . . . . .	44
3.10	5G-Hander. RSRP values throughout the duration of the experiment . . . . .	46

## LIST OF FIGURES

---

3.11	5G-Hander. RSRP values around the handover events . . . . .	46
3.12	UAV for Video Surveillance. Proposed Architecture . . . . .	49
3.13	Drone Functional Architecture . . . . .	50
3.14	Mutual Help Policy implementation algorithm . . . . .	53
3.15	UAV for Video Surveillance. System Model diagrams . . . . .	55
3.16	UAV for Video Surveillance. Numerical Results . . . . .	59
3.17	FANET topology used in the use case . . . . .	60
3.18	UAV for Video Surveillance. Application-level QoS parameters . .	62
3.19	UAV for Video Surveillance. Dependence of application-level QoS parameters on the job arrival rate during alarm states . . . . .	63
3.20	UAV for Video Surveillance. Performance parameters for a be- haviour analysis of the proposed framework . . . . .	64
3.21	UAV for Video Surveillance. Normalized histograms of the job queue length . . . . .	65
3.22	An IoT network zone covered by MEC UAV . . . . .	67
3.23	UAV and RL. Time diagram of the sequence of the events in a state transition . . . . .	73
3.24	i7-2600K power consumption vs. clock speed for different numbers of active CPUs . . . . .	77
3.25	Job loss probability for the IoT devices in LA state, against the job service rate for each CPU . . . . .	79
3.26	Job processing latency for the IoT devices in LA state . . . . .	80
3.27	Mean number of active CPUs for the IoT devices in LA state . . .	80
3.28	Job loss probability for the IoT devices in HA state, against the job service rate for each CPU . . . . .	81
3.29	Job processing latency for the IoT devices in HA state . . . . .	82
3.30	Mean number of active CPUs for the IoT devices in HA state . .	82
3.31	UAV cooperation scenario. Functional architecture of the helped and helping MEC UAVs . . . . .	83
3.32	UAVs collaboration and RL. Time diagram of the sequence of the events in a state transition . . . . .	85
3.33	UAVs cooperation and RL. Loss probability for $L = 3$ available CPUs . . . . .	93
3.34	UAVs cooperation and RL. Mean delay for $L = 3$ available CPUs	93
3.35	UAVs cooperation and RL. Mean number of active CPUs for $L = 3$ available CPUs . . . . .	94

## LIST OF FIGURES

---

3.36	UAVs cooperation and RL. Loss probability (on both non-offloaded and offloaded jobs) . . . . .	95
3.37	UAVs cooperation and RL. Mean delay . . . . .	95
3.38	UAVs cooperation and RL. Power consumption gain . . . . .	95
3.39	UAVs cooperation and RL. Reduction percentage of flight duration for $L = 3$ . . . . .	96
3.40	UAVs cooperation and RL. Reduction percentage of flight duration for $L = 4$ . . . . .	96
4.1	Tactile Internet scenario . . . . .	100
4.2	TSE Reference System . . . . .	101
4.3	TSE implementation. Network in Good state . . . . .	102
4.4	TSE implementation. Network in Bad state . . . . .	102
4.5	Architecture of the TSE . . . . .	103
4.6	A standard LSTM unit . . . . .	105
4.7	TSE. Diagram of AI Engine life cycle . . . . .	106
4.8	TSE Use Case: Local Setup . . . . .	107
4.9	TSE Use Case: Distributed Setup . . . . .	109
4.10	TSE. Obstacle positions drawn for the Level 1, Lap 1 . . . . .	110
4.11	TSE. Force and position traces during a game in the local setup . . . . .	111
4.12	TSE. Model comparison for different look_backs $L_B$ , at different levels . . . . .	112
4.13	TSE Performance analysis . . . . .	113
4.14	Loss probability measured without TSE . . . . .	114
4.15	VEC Network Architecture . . . . .	116
4.16	Vehicular Scenario: multy layer system architecture . . . . .	119
4.17	Vehicular Scenario: job arrival process . . . . .	122
4.18	Multi-layer offloading in a Vehicular Network: numerical results . . . . .	129
5.1	High level architecture of the Triangle testbed [81] . . . . .	132
5.2	Transport layer in the Triangle testbed [81] . . . . .	134
5.3	The vsSTB service chain . . . . .	136
5.4	DiMoViS deployment when user is at home . . . . .	137
5.5	DiMoViS deployment when user is in mobility . . . . .	138
5.6	DiMoViS in Triangle. Network level measurement scenario . . . . .	139
5.7	DiMoViS in Triangle. Application level measurement scenario . . . . .	140
5.8	DiMoViS in Triangle. VFs migration in 5 steps . . . . .	141

## LIST OF FIGURES

---

5.9	DiMoViS in Triangle. Frame rate degradation when IP Camera changing events during live video streaming . . . . .	143
5.10	DiMoViS in Triangle. Frame rate vs time jumping on the seek bar during payout of a pre-recorded event . . . . .	143
5.11	DiMoViS in Triangle. RX Data Rate . . . . .	144
5.12	DiMoViS in Triangle. TX Data Rate . . . . .	144
5.13	DiMoViS in Triangle. RSRP and RSSI values . . . . .	145
5.14	Tourist Eyes. State of the art . . . . .	147
5.15	Tourist Eyes. New testbed . . . . .	147
5.16	Tourist Eyes architecture . . . . .	148
5.17	Tourist Eyes Manager . . . . .	151
5.18	Tourist Eyes app . . . . .	152
5.19	Tourist Eyes. Data Flow . . . . .	153
5.20	Tourist Eyes. VNFD in OSM . . . . .	154
5.21	Tourist Eyes. NSD in OSM . . . . .	154
5.22	Tourist Eyes. Position of the installed camera in Millennium Square	155
5.23	Tourist Eyes. Path during experiment execution . . . . .	155
5.24	Tourist Eyes. Forward and backward patch during the experiment	156
5.25	Tourist Eyes Console Screen - High Quality Scenario . . . . .	158
5.26	Tourist Eyes. Average transmission bitrate from IP Cams in P1 .	158
5.27	Tourist Eyes. Delay and Loss Percentage in P1 . . . . .	158
5.28	Tourist Eyes Console Screen - Poor Quality Scenario . . . . .	159
5.29	Tourist Eyes. Average transmission bitrate in P2 . . . . .	159
5.30	Tourist Eyes. Delay and Loss Percentage in P2 . . . . .	159
5.31	Tourist Eyes Console Screen - Good Quality Scenario . . . . .	161
5.32	Tourist Eyes. Average transmission bitrate in P3 . . . . .	161
5.33	Tourist Eyes. Delay and Loss Percentage in P3 . . . . .	161
5.34	5Gamer. Application scenario . . . . .	163
5.35	5Gamer. System Architecture . . . . .	164
5.36	5Gamer. Comparison with the state-of-the-art in case of malfunction	167
5.37	Flame Platform Architecture [87] . . . . .	169
5.38	FLAME: Possible node status and transitions from one status to another . . . . .	171
5.39	Flame-in-a-Box architecture [88] . . . . .	172
5.40	Topology of Sandpit with clusters (green), emulated UE (red) and SDN switch (blue) [88] . . . . .	173
5.41	Bristol Flame platform topology [90] . . . . .	174

## LIST OF FIGURES

---

5.42	FLAME. Coverage area in Millennium Square [90] . . . . .	175
5.43	VISION Architecture and interaction with the FLAME Platform .	176
5.44	VISION. DiMoViS service: cam registration . . . . .	180
5.45	VISION. DiMoViS service: user access and cam selection . . . . .	180
5.46	VISION. DiMoViS service: power on, if required, of the other components . . . . .	180
5.47	VISION. DiMoViS service: video flow transmission . . . . .	181
5.48	VISION: interactions between DiMoViS platform and Tourist Eyes service . . . . .	182
5.49	VISION: interactions between DiMoViS platform and LPR service	183
5.50	Vision. System Architecture and Stakeholders . . . . .	184
A.1	Structure of an OpenFlow switch [92] . . . . .	201
A.2	Structure of the flow table [19] . . . . .	201
A.3	Pipeline process inside a OpenFlow switch [92] . . . . .	202
A.4	ONOS architecture [93] . . . . .	203
A.5	ODL architecture [95] . . . . .	205



# List of Tables

3.1	5G-Hander. Representative Time Instants and Events during Handover Detection . . . . .	47
4.1	Comparison between Vehicular Edge Computing and Vehicular Cloud Computing [72] . . . . .	115
5.1	DiMoViS in Triangle. Delay in the Personal Acquirer . . . . .	139
5.2	DiMoViS in Triangle. Delay in the Edge Acquirer . . . . .	139
5.3	DiMoViS in Triangle. Latency due to IP Camera changing events during live video streaming . . . . .	142
5.4	DiMoViS in Triangle. Latency due to time jumping on the seek bar	143
5.5	Tourist Eyes. Distance between IP Cams and Access Point . . . . .	153



---

## ABSTRACT

Internet data traffic will maintain its rapid growth in the next future due to the ever increasing development of new application services and devices. The 4G network is not able to meet this explosive growth in traffic demand. For this reason, 5G mobile network represents an optimal solution due to its designed architecture, modifying the actual ossified infrastructure thanks to the use of paradigms like NFV, SDN and MEC. A key feature of 5G is Network Slicing, a technology that allows the creation of virtual networks able to guarantee different applications requirements.

In this context, this thesis was realized with the aim of studying the main aspects of network softwarization in 5G environment, from the management and orchestration of network slices to their use for vertical services. In particular, as regards the management and orchestration, a first work concerns the definition of a framework for the prediction of handovers, aimed at optimizing resources in scenarios characterized by high variability of the access point by the users. In the same context, the use of UAVs to extend network slices is proposed, providing networking and computing resources at the edge of the network. Using Reinforcement Learning, it is possible to offload part of the data to be processed from one UAV to another nearby, optimizing parameters such as latency, loss probability and energy consumption.

About the use of network slices for vertical services, two works are presented: the first concerns the definition of the functional architecture of the Tactile Support Engine in the Tactile Internet network slice. The second proposed work concerns hierarchical offloading in vehicle networks based on Reinforcement Learning techniques.

Finally, thanks to the participation in European projects, it was possible to define and implement some application scenarios characterizing a 5G ecosystem, taking care of three areas that are now considered strategic: distributed video surveillance, development of assistive technologies for patients suffering from various kinds of diseases and online gaming.

**Keywords:** *5G, Network Slicing, MEC, Reinforcement Learning, Resources Orchestration.*

# Chapter 1

## Introduction

Internet data traffic shows no decreasing signs and is expected to maintain its rapid growth in the next future due to the ever increasing development of new application services, and the proliferation of devices to support them. According to [1], nearly two-thirds of the global population will have Internet access by 2023, there will be 5.3 billion total Internet users (66 percent of global population) by 2023, up from 3.9 billion (51 percent of global population) in 2018. The number of devices connected to IP networks will be more than three times the global population by 2023. Due to the huge amounts of data traffic that is expected to be produced by new applications like high-resolution games on line, 4K and 8K video streaming, augmented reality and virtual reality, and the emerging “mission-critical” scenarios, like traffic control, robotic control, production and automation, robotic surgery, and autonomous driving cars [2], the 4G wireless communication system is not able to meet this explosive growth in traffic demand. Consequently, the development of a new telecommunications network has become necessary.

The 5G mobile network represents a revolution if compared with previous mobile network generations due to its designed architecture. The main feature of this new network is that it makes changes not only to the Radio Access Network (RAN), but also to the fixed network. It is deployed with the aim of offering greater use of Cloud, Edge and Fog solutions, softwarization of the hardware and direct end-to-end (e2e) communication between devices. This allows 5G to address the wide range of needs of the vertical industries and support new services that are currently unthinkable (for example, telesurgery), but also services already developed for current telecommunications networks which, thanks to the introduction of new technologies, will be able to achieve a significant increase in performance.

Even end users change: until nowadays, mobile networks had always the only

need to provide connectivity to human consumers, through the use of smartphones, tablets and personal computers. Thanks to an unlimited mobile broadband experience and with the progressive development of information technology, the Internet of Things (IoT) [3] is emerging as a huge paradigm shift by connecting a versatile and massive collection of smart objects to the Internet. The IoT enables physical objects to see, hear, think and perform jobs by having them “talk” together, to share information and to coordinate decisions, transforming them in smart objects. This means that not only humans will exploit 5G technology to be able to communicate or exchange data better than now, but there will be an increase in communications between humans and objects, or between everything from human-held smart devices to sensors and machines and, most importantly, supporting critical machine communications with instant action and ultra-high reliability [4].

Also telcom industry is changing, moving from a "horizontal" toward a "vertical" service delivery model. The first one is characterized by the idea to define services independent of their consumers' needs, while in the vertical model the services are tailored to specific industry sectors, each one with very diverse and extreme requirements [5].

To achieve these challenges, network is modifying its ossified infrastructure through some innovations [6]:

- *New Radio*, a new air interface designed to greatly improve the performance of the access network;
- *Network Softwarization Process*, which will be supported by three main paradigms that, although born in different contexts and for different purposes, are converging together in the 5G system: Network Function Virtualization (NFV), Software Defined Networking (SDN) and Multi-Access Edge Computing (MEC);
- *Network Slicing*, that consists in the creation of a dedicated virtual network architecture with the aim of providing specific functionalities for certain services exploiting the real available physical network;
- full exploitation of *in-network computing*, following the recent trend that is transforming the Internet in a network of data centres in which the prevailing communication paradigm is becoming device-to-computing-to-device, rather than device-to-device;
- a *Service-Based Architecture*, for which network control functions expose an

Application Programming Interface (API) based on HTTP/2 and RESTful technologies, thus providing an unprecedented flexibility, simplifying the deployment and the evolution of the network and harmonizing the entire network control plane with Web technologies.

Due to the possibility of different types of communications and applications requirements, the International Telecommunication Union Radiocommunication Sector (ITU-R) has defined three usage scenarios to expand and support diverse families of applications [7]:

- *Enhanced Mobile Broadband (eMMB)*, addressing human-centric use cases for access to multimedia content, services and data;
- *Ultra-reliable and low latency communications (URLLC)* with strict requirements, especially in terms of end-to-end latency and reliability;
- *Massive machine type communications (mMTC)* for a huge number of connected devices, typically transmitting a relatively low volume of non-delay-sensitive information.

This thesis work aims to analyze aspects related to the network softwarization in the context of 5G ecosystems. In particular, the main objectives of this thesis are:

- management and orchestration of network slices: in this context, the first work regards the management of handover in the RAN portion of a network slice, with the implementation of a network service that is able to detect handover packets between physical and virtual base stations. The captured messages are analyzed and communicated to the Network Orchestrator using publish/subscribe approach. In this way, the Orchestrator is able to allocate resources to prepare the network to manage the handover, or block this last one, sending an alert trigger to the Mobility Management Entity block of the RAN, if the resources are limited or the network behind the new access point presents congestion or faults. The second work is about the use of Unmanned Aerial Vehicles (UAVs) to extend network slices to provide computing and network facilities to IoT devices for area monitoring applications. This problem is addressed proposing the analytical model realized in a first step, then implemented in a Matlab simulator, and finally improved with Reinforcement Learning techniques;

- network slicing for vertical applications: in this field, two different scenarios are studied. The first regards the architectural definition of the Tactile Support Engine (TSE), a component that supports the network to respect application requirement in a Tactile Internet scenario, implementing Artificial Intelligence techniques. The second scenario regards Vehicular Networks and the use of MEC servers installed along the roadway to provide facilities to the vehicles during the processing of the data collected from the environment in a context of a smart city.
- 5G-enabled smart services: in this part, four frameworks have been implemented in the context of smart services for smart cities. In particular, a distributed mobile video surveillance system was realized to allow end-users to receive the video stream from cameras to which they are registered, wherever they are. After that, two services for people with visual and motor disabilities have been deployed: TouristEyes and Vision: using machine learning techniques, video flows coming from cameras are analyzed with the aim of recognizing a blind person (helping him/her to walk in an unusual city context) or laying person (helping him/her by sending alerts to relatives and/or law enforcement agencies). Another use case regards a gaming online scenario, where the 5Gamer application was realized: it is a revival of the famous Pong game, with the addition of Artificial Intelligence that trains according to the gaming habits of the physical player. It has made possible to understand the necessary requirements to be able to guarantee the user to play at best in an online context, even during network outage periods.

This thesis work is structured as shown in Fig. 1.1. Chapter 2 will present an overview about the enabling technologies for 5G networks: SDN, NFV, Cloud/Edge Computing, MEC and Network Slicing. In Chapter 3, the management and orchestration of network slice will be addressed, and the 5G-Handler network service and the use of UAVs to extend a network slice will be presented. The Network Slicing technology for vertical applications will be described in Chapter 4. In Chapter 5, four frameworks implemented in the context of European Projects during the whole period of the Ph.D. course will be shown; they represent typical 5G-enabled smart services. These projects have allowed to work on the main issues typical of the network softwarization and orchestration in 5G network, not limiting the discussion to only the theoretical aspects, but also being able to address them from a practical point of view. Finally, in Chapter 6 the conclusions of this thesis will be drawn.

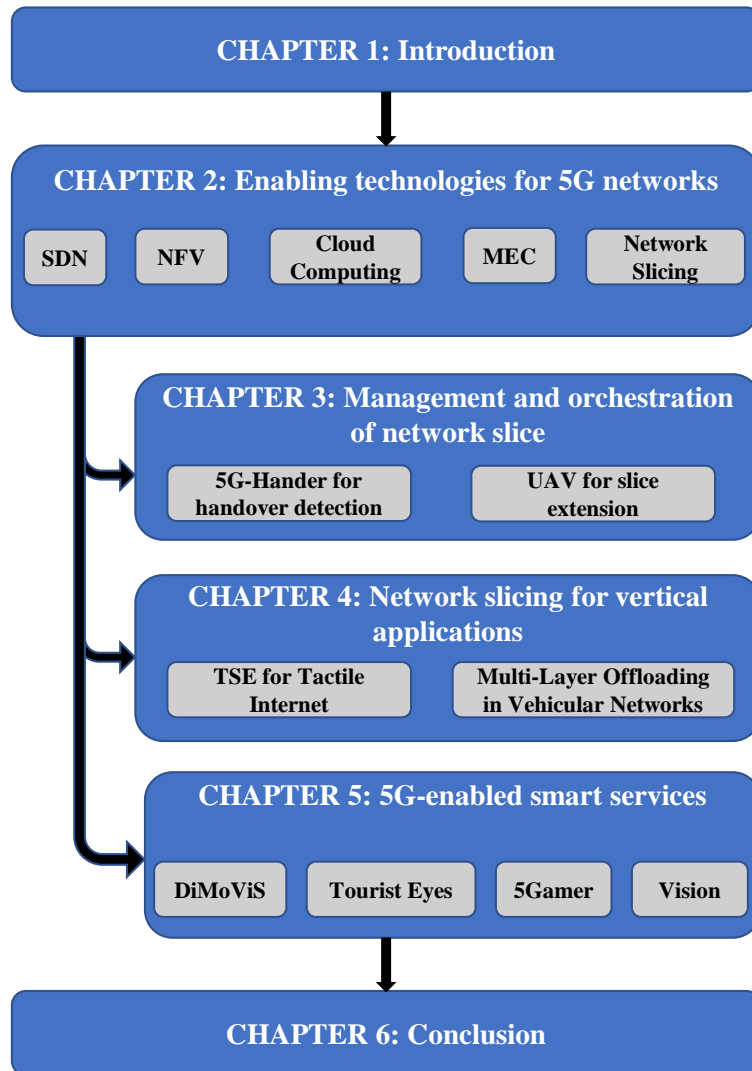


Figure 1.1: Structure of this thesis



# Chapter 2

## Enabling technologies for 5G networks

### 2.1 Network Function Virtualization (NFV)

Traditionally, network operators have deployed physical proprietary equipment and devices for every part and function of the network. However, with the increase of user demands on network functions, they are required to acquire many specialized network hardware (known as middleboxes or hardware appliances) to satisfy the plethora of heterogeneous user requirements, and it is difficult to manage these devices when integrated with the network.

Middleboxes embody a large variety of specialized functions (for example L2 Switching, Routing, Network Address Translation (NAT), Firewall (FW), Deep Packet Inspection (DPI), Intrusion Detection System (IDS), Load Balancer (LB)) and recent studies [8] show that they are ubiquitous in enterprise networks and their number is comparable with the number of routers and switches needed to maintain the operation of the network. This results in high Capital Expenditure (CapEx) and Operational Expenditure (OpEx). Another problem is that middleboxes have vendor-specific interfaces and, consequently, suffer of a slow protocol standardization [9]. Deploying new network services (NSs) is also a tedious process, as technicians are required to visit specific sites and place the middleboxes in a pre-defined order to form the correct service function chains (SFCs).

One of the proposed solutions to deal with these difficulties is Network Functions Virtualization (NFV). NFV is a term used to represent the implementation of hardware appliances providing data-plane network functions (firewalls, gateways, etc.) in software, executed in commodity hosts. To this purpose, it leverages on virtualization technology, widely adopted in the recent past to data centers and

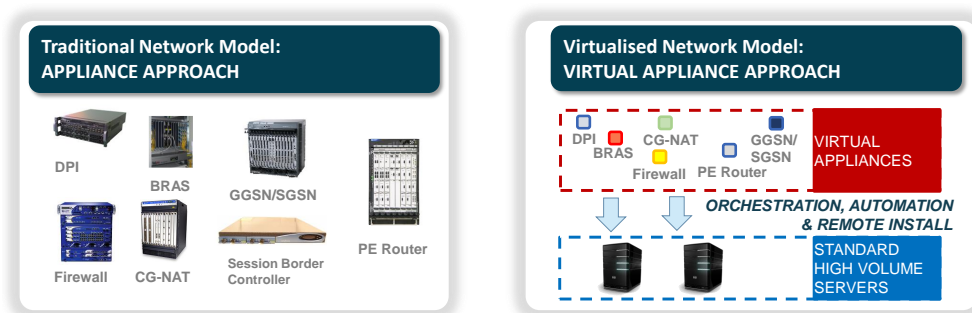


Figure 2.1: Traditional network functions vs NFV paradigm [11]

clouds, offering a new way to design the networks and to provide flexible network functions deployment [10].

An illustrative example contrasting NFV paradigm with traditional network is shown in Fig. 2.1. Compared to the traditional service provisioning paradigm based on hardware middleboxes, NFV manages to achieve cost-effectiveness by consolidating multiple instances of VNFs on high-volume yet inexpensive servers, routers, or storage. Service provisioning in NFV is also highly simplified as the previous troublesome tasks, such as middlebox deployment, monitoring, migration, and scaling, can be optimally automated through software control mechanisms.

Thanks to the virtualization, traditional middleboxes are managed as single independent modules of software running into virtual machines (VMs), programmed to play the role of a particular VNF, allowing modularity and isolation of each function. Moreover, being VNFs deployed on general-purpose servers, dynamic migration from one server to another one is possible, with the aim of “following the users” or to be consolidated on few servers for energy saving purposes [12]. This makes the setup of the network more flexible based on the user needs as well as making the process of scaling the network easier.

NFV promises many benefits to network operators. The most significant ones are [12]:

- *independence*: software is no longer integrated with hardware in NFV. As a result, their evolution will be independent from each other;
- *flexibility*: the decoupling of software from hardware helps to reassign and share the same infrastructure resources, which allows to perform different functions at various times. As a result, the deployment of network functions and their connections becomes more flexible;
- *scalability*: NFV optimizes the NS provisioning and provides scalable and elastic automation of services;

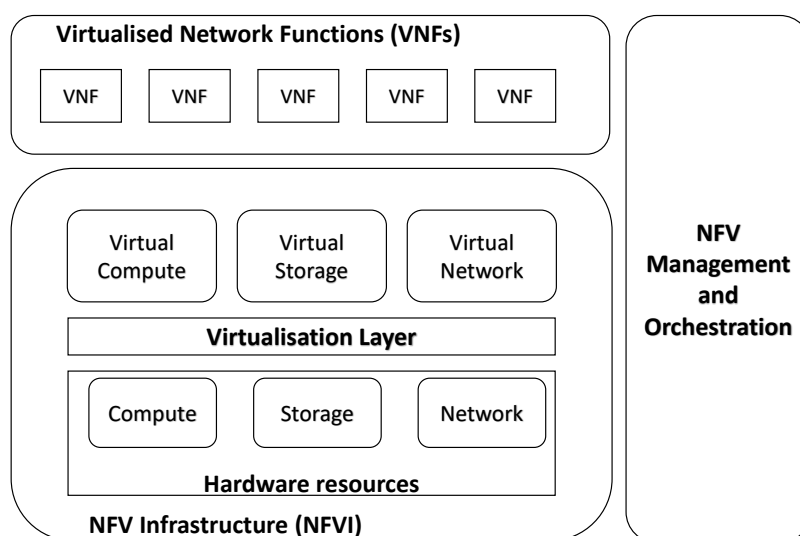


Figure 2.2: ETSI high level NFV framework definition [12]

- *reduced energy consumption*: with the ability of scaling resources up and down, it is possible to reduce the OpEx needed to run network devices;
- *time-to-market and service instantiation reduction*: the deployment time becomes comparable with that of software tools, with a reduction from few years to few months, with better integration and testing abilities;
- *open-source*: being software, development of network functions can be done according to open-source paradigms, so simplifying dissemination and deployment of network functions.

The main architectural components of the NFV model (see Fig. 2.2) are the *Network Function Virtualization Infrastructure (NFVI)*, the *Virtualised Network Functions (VNFs)*, and the *Management and Network Orchestration (MANO)* [12].

The NFVI includes all hardware (a single computer or a computer cluster that can be located in data centers, network nodes or close to end user premises) and a software framework (offers functions that are commonly required by NFs, such as NF placement, dynamic scaling, etc) on which to deploy, manage and execute the VNFs, through the creation of a *Virtualisation layer*. It sits right above the hardware with the objective of abstracting the hardware resources, so they can be logically partitioned and provided to the VNFs to perform their functions. In the case of NFVI-PoP located over several areas, the NFV Infrastructure has a decentralized structure, being able to virtualize resources physically distant but which appear to be located in the same point to the VNFs that use them. Ob-

viously, the network resources that connect the various NFVI-PoPs will also be part of the NFVI. The network resources are composed by switching functions and wired/wireless links. NFV groups all networks into two classes: *NFVI-PoP network*, to indicate the network that interconnects the computing and storage resources inside a NFVI-PoP, and the *Transport network*, that interconnects NFVI-PoP to other network appliances or terminal. The Virtualization layer abstracts the hardware resources. Its aims are: abstract and logically partition the resources in the physical layer, providing virtualised resources to the VNF, after enabling the latter to use these resources.

The Virtualised Network Functions component performs the virtualization of a network function (NF). A NF is a piece of software that implements data-plane network functions ranging between basic packet forwarding to complex middlebox functions, such as intrusion prevention systems. They are run on servers belonging to the NFVI. It is worth highlighting that the general concept of decoupling NFs from dedicated hardware does not necessarily require virtualization of resources. This means that the network operator could provide software NFs running them on physical machines not using virtualization. Nevertheless, running them in Virtualization Containers (VC), e.g. in VMs or Docker Containers (DC), makes the difference and realizes the gain of NFV. In fact, if NFs are virtualized, they can be run on commodity servers, and present many advantages in terms of flexibility, dynamic resource scaling, and energy efficiency [13]. Needless to mention, it is also possible to have hybrid scenarios as it was said so far. When NFs are executed on VMs, they are referred to as VNFs. In Fig. 2.3 there is an example of NS, in which it is possible to see all the levels that take part to its deployment. A NS is defined through a *VNF Forwarding Graph* (VNF-FG), that is the list of all the VNFs that compose it, in the exact order in which the flow managed by the NS must pass through them. In the figure it is represented the case of a *nested VNF-FG* for the VNF2.

The third element constituting the NFV model is MANO, whose target is management and orchestration of the whole network. It includes functions as Fault management, Configuration management, Accounting, Performance monitoring and Security (FCAPS), and orchestrators, which manage service chains of multiple NFs [14]. Research regarding NFV in the last years has focused on MANO, and specifically on its task of NF placement, also referred to as resource allocation, for optimally locating NFs in physical servers and/or VMs. The survey in [15] focused on virtual network embedding, and reported on several optimization techniques for ideal NF placement.

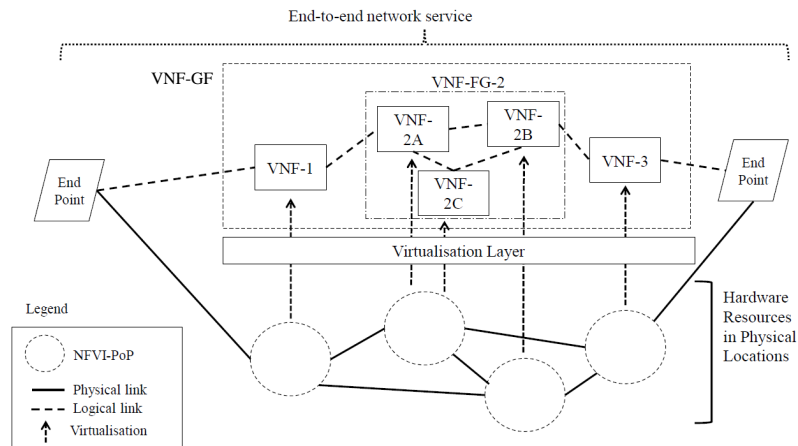


Figure 2.3: ETSI End-to-end Network Service definition [12]

Fig. 2.4 shows a more detailed view of the NFV architecture. Inside the components already described, there are [12]:

- *Element Management System (EMS)*: placed on top of VNF, manages the functionality for one or more VNFs;
- *Virtualised Infrastructure Manager (VIM)*: inside the MANO, it deals with controlling and managing the interactions between VNF and computing, storage and network resources;
- *NFV Orchestrator (NFVO)*: inside the MANO, its rule is to orchestrate and manage the NFV infrastructure and software resources, and realize NS on NFVI;
- *VNF Manager (VNFM)*: inside the MANO, it manager the VNF lifecycle, from the instantiation to the termination, through update, query and scaling phases;
- *Service, VNF and Infrastructure Description*: inside MANO, it provides information about VNF deployment, VNF-FG, services information and NFVI infrastructure information;
- *Operation/business support system (OSS/BSS)*: the objective of this block is to behave as a management system that allows the network providers to deploy and manage various end-to-end telecom services like ordering, billing, trouble-shooting, etc. OSS is in charge of functions of infrastructure planning, such as service provisioning, network inventory, network configuration, and fault management. BSS, on the other hand, deals with operations like ordering, billing, and revenue management aspects [16].

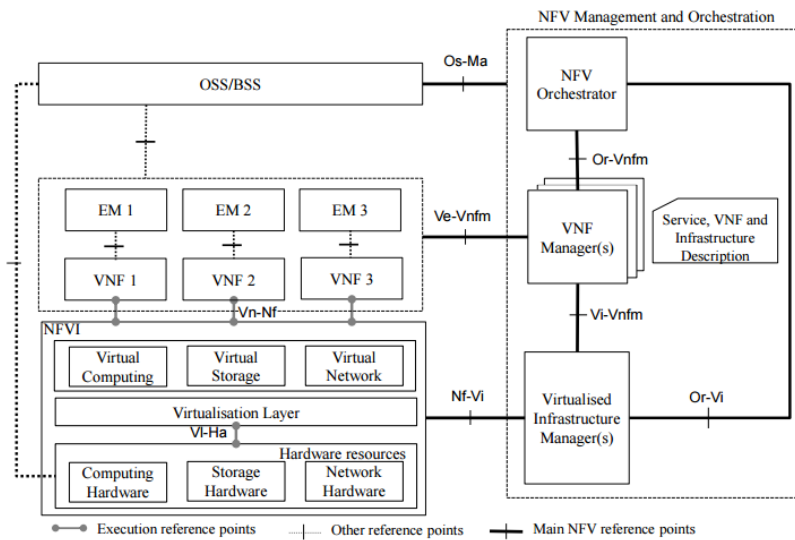


Figure 2.4: ETSI NFV Architecture definition [12]

To carry out its task, the NFV Orchestrator also manages four catalogues:

- *VNF Catalogue*: it represents the repository of all the on-boarded VNF Packages. The properties of each VNF are described using a VNF Descriptor in which the resources needed by the VNF (in terms of virtual compute, storage and networking) are listed. About the connectivity, it is possible to specify external connection points, internal virtual links and internal connection points of each VNF. The definition of these connection points is important because, otherwise, the VNF will not be able to connect to the Internet and to other VNFs of a NS. Also the image used to deploy the VNF is specified, image that has to be available inside the VIM. Both NFVO and VNFM can query the VNF Catalogue for finding and retrieving a VNF descriptor, to support different operations;
- *NS Catalogue*: it represents the repository of all of the on-boarded NSs, supporting the creation and management of the NS deployment templates. Each NS is defined by a NS Descriptor, in which all the necessary VNFs that compose the NS are listed, together with specifications about the virtual links used to connect them (through the connection point of each VNF described in the VNF descriptor). For each virtual link, if two different communication channels are necessary, it is possible to specify if it is a management virtual link or a data virtual link;
- *NFV Instances Repository*: it holds information of all VNF instances and Network Service instances. Records saved on it are updated during the life-

cycle of the respective instances, reflecting changes resulting from execution of NS lifecycle management operations and/or VNF lifecycle management operations. This supports NFVO's and VNFM's responsibilities in maintaining the integrity and visibility of the NS instances, respectively VNF instances, and the relationship between them;

- *NFVI Resources Repository*: it holds information about available/reserved/allocated NFVI resources as abstracted by the VIMs across different operator's Infrastructure Domains, thus supporting information useful for resources reservation, allocation and monitoring purposes. Therefore, this repository plays an important role in supporting NFVO's Resource Orchestration and governance role, by allowing NFVI reserved/allocated resources to be tracked against the NS and VNF instances associated with those resources.

Another important activity was carried out by the European Telecommunications Standards Institute (ETSI) in defining the following set of relevant use cases [17], as described in [18]:

- *Network Functions Virtualization as a service*: NFV infrastructure, platform and even a single VNF instance can be provided as a service by a network operator, based on models similar to the cloud computing service models;
- *Virtualization of Mobile Core Network and IMS*: mobile networks and the IP Multimedia Subsystem are populated with a large variety of proprietary hardware appliances, which costs and complexity can be reduced introducing NFV (specially for the coming 5G);
- *Virtualization of Mobile Base Station*: mobile operators can apply NFV in order to reduce costs as well as continuously develop and provide better service to their customers;
- *Virtualization of the Home Environment*: installation of new equipment can be avoided in the home environment with the introduction of VNFs, reducing maintenance and improving service provision;
- *Virtualization of CDNs*: Content Delivery Networks use cache node to improve the quality of multimedia services, but it comes with lots of disadvantages (e.g., waste of dedicated resources) that could be mitigated by NFV;

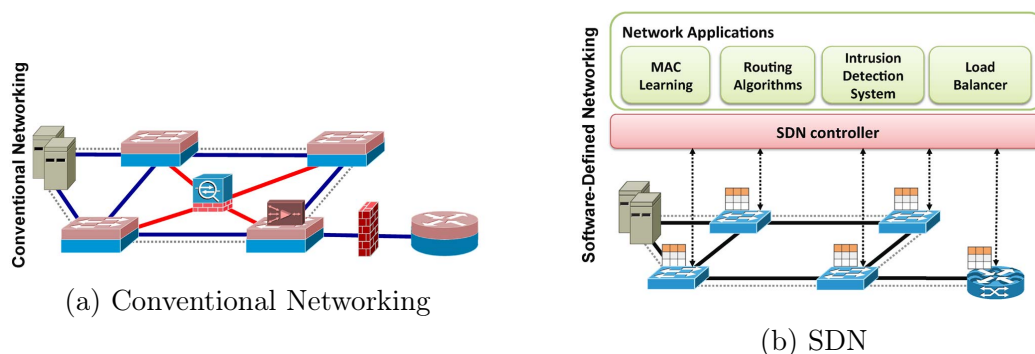


Figure 2.5: Traditional Networking vs SDN [19]

- *Fixed Access Network Functions Virtualization*: virtualization supports multiple tenancy in access network equipment, whereby more than one organizational entity can either be allocated, or given direct control of, a dedicated partition of a virtual access node

## 2.2 Software Defined Networking (SDN)

The traditional IP network is complex and hard to manage: this is due to the presence of control and transport protocols inside routers and switches, forcing network operators to configure each device using the specific vendor protocol. Another problem is the possible failures that occur within the network. Lacking someone who has an overall view of the network, if a device stops working due to a failure, the other components are not informed of the damage in time, but must find out themselves, causing service disruptions and malfunctions [19].

SDN is a computer networking approach that allows network administrators to manage services through the abstraction of low-level functionalities. This is done by decoupling the system that makes decisions about where the traffic is sent from the system that forwards the traffic to the selected destination. A comparison between traditional and SDN networks is shown in Fig. 2.5.

Fig. 2.6 shows a simplified logical view of the SDN architecture. The lowest level is the *Infrastructure layer* (or data plane), constituted by the physical structure of the network. It includes software and hardware devices, such as routers, switches, access points. It has the task of dealing with the various operations on the data, such as reassembly and fragmentation, as well as forwarding. As the routing control is removed from the devices, in this layer there are only programmable switches [20]. These switches are characterized by one or more flow tables, a secure channel for connection with the controller, and the OpenFlow protocol,



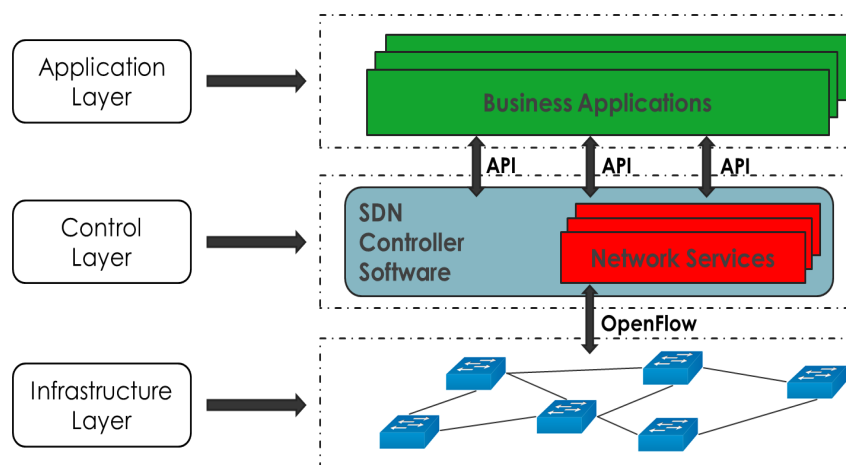


Figure 2.6: Software Defined Networking

defined by Open Network Foundation (ONF), installed inside. As for hardware switches, when an SDN switch receives a packet, it first checks in the flow tables by matching its packet header. If it finds a flow entry against this matching, it follows the policy described in the table entry (for example, forward the packet through a specific port). Instead, if it does not find any entry, it forwards it to an external controller, which can add a new entry in the switch table to manage this type of packet. Each rule in the flow tables specifies an action (dropping, forwarding, modifying, etc) for a subset of the traffic. Depending on the rules, an OpenFlow switch can act like a router, switch, firewall, or perform roles like load balancer, traffic shaper, and so on [19].

The central level is the *Control layer*, where is located the *SDN controller*. It has the task of establishing flow tables and data management rules. It realizes the abstraction of the network complexity and collects information through the Southbound APIs; through the latter, the controller manages the network devices and chooses the optimal path for the data to be transmitted. The SDN controller allows dynamic management of the network to adapt the flow to the needs of the moment. It is possible to say that a relevant part of the network intelligence is located inside this component. It is the task of this level to have a global and correct view of the network and how it is working in real-time. A logically centralized programmatic model does not mean that the system is physically centralized too: scalability and reliability are required and a centralized system is not the better solution. For these reasons, a physically distributed control plane can be implemented [19].

The communication between the SDN control layer and the Infrastructure layer is entrusted to the OpenFlow. Through control messages, the controller can com-

municate with the devices to send, for example, new flow entries for routing and forwarding tables. There are two types of communications with Southbound APIs [21]:

- *in-band*: control messages are sent on the same channel used to transport data traffic;
- *out-of-band*: control messages are sent using a different channel than that used for data.

The out-of-band is easier to design than in-band solution because the controller is directly connected to each switch. However, this solution requires an extra physical port on each SDN device. In fact, in this case, OpenFlow defines a virtual port (reserved) which enables remote entities to interact with the switch.

Note that, in addition to the Southbound APIs, there are also the Northbound APIs (used for communications between applications and controllers), Eastbound and Westbound APIs that allow multiple controllers to exchange information about the data flow in the Data Plane. The functions of these last two interfaces include import/export data between controllers, algorithms for data consistency models, and monitoring/notification capabilities (e.g., check if a controller is up or notify a take over on a set of forwarding devices) [19]. An important issue regarding Eastbound and Westbound APIs is heterogeneity: an SDN controller has to be able to communicate not only with peer controllers but also with subordinate controllers (in the case of a hierarchy of controllers).

Some controllers like NOX and POX have centralized architecture, while controllers like FloodLight, Open Network Operating System (ONOS) and OpenDaylight are distributed. Compared to most controllers, OpenDaylight not only supports OpenFlow but also other protocols (i.e. OvSDB, SNMP, NetConf) of southbound interfaces [20].

The highest level, that is the *Application layer* (also called Application Plane), has the task of establishing the rules. It also provides various services, such as firewall, access control, security services for intrusion detection system and intrusion prevention system (used to identify unauthorized access), routing and proxy services, balancing monitoring (i.e. load control and balancing). This level is responsible for the abstraction of the control management of SDN networks. In fact, applications can build an abstract view of the network by collecting information from the SDN controller, via the Northbound API, and can act on the physical infrastructure by communicating with the control layer via the API. The ability to use a network abstraction, leaving out the details of the topology, means that

networks become *application-customized* and applications are *network-capability-aware*.

The benefits brought about by the use of SDN technology are both in terms of CapEx and OpEx thanks to a simplification of the network architecture and management procedures.

SDN aims to make the control plane programmable, as well as the network nodes (like routers and switches), by introducing appropriate levels of abstraction accessible through API control interfaces, as described before [22]. Business applications and network services perceive the network as a logical entity; this allows the definition of network services by abstracting from the specificity of the single device. The decisions regarding switching and flow engineering, first implemented according to a "monolithic" approach by the device itself, are now taken by the SDN controller, that can create flow entry in such a way as to manage two different flows in a different way, even if the source and destination are the same. Application and network functions become independent from the vendors and from the characteristics of each device. This results in a simplification of network planning and operational management. Routers and switches become easier to build and configure, having to perform only forwarding functions and having to implement only the protocol to communicate with the SDN controller. In this way, the innovation and automation processes are speeded up, allowing the creation of new network functions and services, in a simpler and faster way without having to configure individual devices or modify their firmware. The control plane can be enriched with new functions more quickly. It is possible to apply policies with different levels of granularity at the session, user, device and application levels with a consequent increase in the reliability and security of the network. Furthermore, different applications can take advantage of the same network information to take more consistent and effective policy decisions. Finally, the integration of different applications becomes more straightforward: for instance, load balancing and routing applications can be combined sequentially, with load balancing decisions having precedence over routing policies [19].

For the sake of completeness, the OpenFlow protocol and the two main SDN Controllers (ONOS and OpenDaylight) are covered in the Appendix.

### **2.2.1 SDN-NFV integration**

SDN and NFV, given their characteristics, are strongly linked and complementary, although they were born as completely independent protocols and have distinct architectures and purposes. As described in [23], SDN aims to achieve a

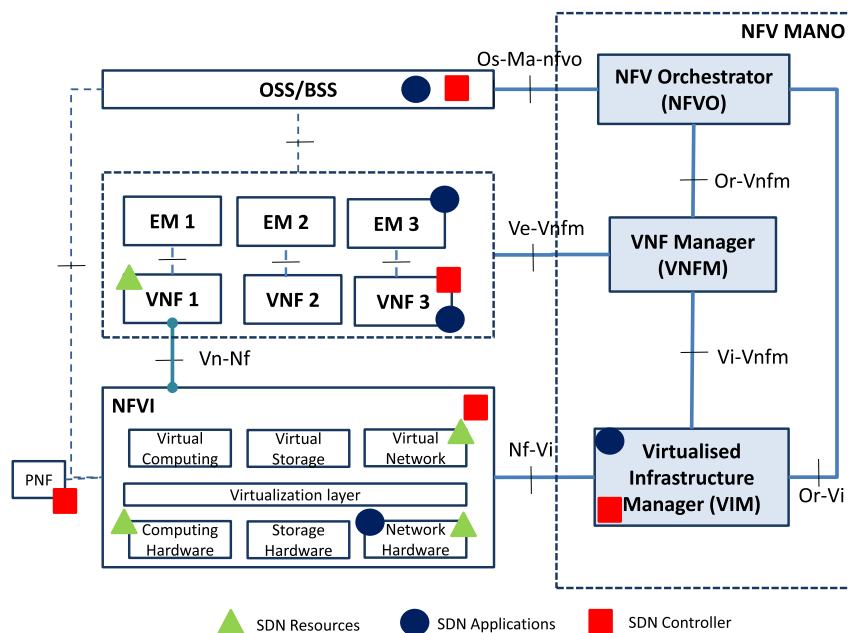


Figure 2.7: SDN and NFV integration [26]

centrally controlled and programmable network architecture, providing network abstractions by decoupling network control from the data forwarding plane, while NFV was born to virtualize network functions, decoupling them from proprietary hardware, reducing CapEx, OpEx and energy consumption as much as possible. However, both can benefit from each other:

- the SDN controller, if interpreted as a network function, can be virtualized through the NFV approach to adapt it more easily to a cloud context. By becoming a VNF, its dynamic migration to optimal locations becomes much easier and above all increases its performance, being able to be placed closer to devices;
- the SDN paradigm, given its ability to control flow tables of network elements and thus being able to manage traffic optimally, helps NFV by providing programmable network connectivity between VNFs [24].

For these reasons, ETSI NFV has published a report [25] in which various possible options of integration between SDN and NFV are proposed, trying to place SDN elements inside NFV architecture. In [26], the Authors proposed some solution, and the result is summarized in Fig. 2.7.

In particular, there are the following cases:

- the SDN Resources are located in four possible points, because they could be physical switch or router (Network Hardware), virtual switch or router (Vir-

- tual Network), e-switch, software-based SDN enabled switch in a server NIC (Computing Hardware) or switch/router implemented as VNF (VNF1);
- the SDN Controller is located in five points: merged with the VIM, virtualized as a VNF (VNF3), part of the NFVI, part of the OSS/BSS, or it is a Physical Network Function (PNF);
  - the SDN Applications are located in five points: part of a PNF (Network Hardware), part of the VIM, virtualised as a VNF (VNF3), part of an EM (EM3) or part of the OSS/BSS.

For more details, the reader can refer to the technical report in [25].

## 2.3 Network Slicing

As mentioned in Chapter 1, the 5G network will provide innovation opportunities for many sectors of common life, industries and communications between humans and/or devices. Today's networks use a "one size fits all" approach, which makes them unsuitable to encounter the performance requirements of future use cases in terms of latency, scalability, availability and reliability. This implies the need to provide programmability, flexibility and modularity required by all these possible use cases. For this purpose, 5G must be able to create multiple logical networks, each capable of guaranteeing the requirements of a specific use case, on top of a common network infrastructure. These logical networks are called *network slices*. In [27], the Authors define network slices as e2e logical networks running on a common underlying (physical or virtual) network, mutually isolated, with independent control and management, and which can be created on-demand. Such self-contained networks must be flexible enough to simultaneously accommodate diverse business-driven use cases from multiple players on common network infrastructure.

In Fig. 2.8, there is an example of network slicing, in which it is possible to see how the same elements of the physical infrastructure can be used as "virtual" elements of each slice, depending of the requirements of the specific application layer. Thanks to this structure, each particular type of application will be able to "see" a network configured in the best way to manage its traffic. For example, a self-driving car application will need to guarantee low-latency services but without the need of high throughput. Instead, a streaming service will need both high throughput and low latency.

A network slice is composed by two types of elements [27]:

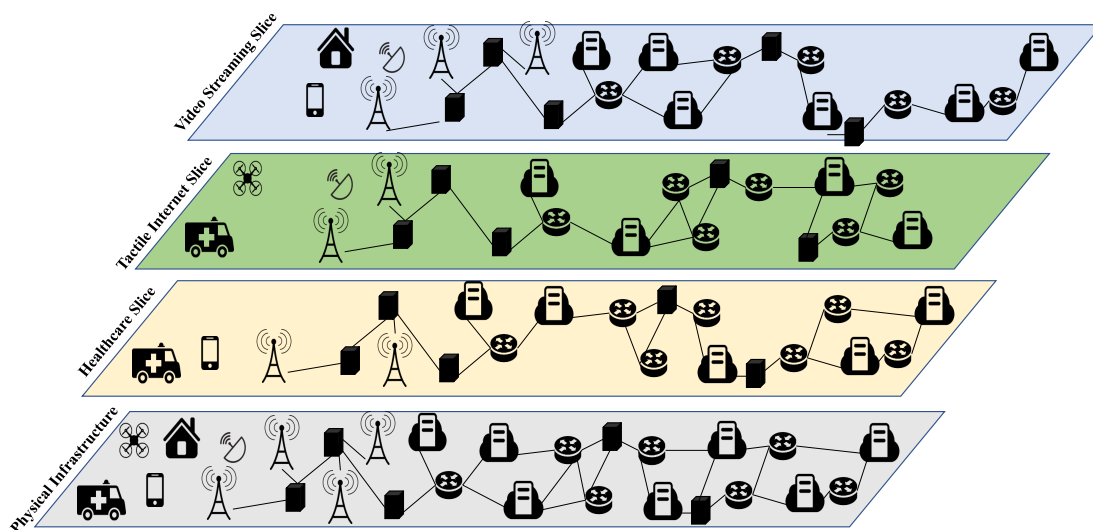


Figure 2.8: Network slice

- *NFs*: functional blocks aimed to provide network capabilities to encounter the application requirements. Generally, *NFs* are implemented as software running on infrastructure resources.
- *Infrastructure Resources*: heterogeneous hardware (wireless access network, edge/cloud computing server, satellites, unmanned aerial vehicles, Wi-Fi access, switches/routers, links) and necessary software for hosting and connecting *NFs*;

Generally, it is possible to consider a network slice architecture as composed of two blocks [28] [29] (see Fig. 2.9): one for the slice implementation and the other for the slice management and configuration. The first block is also composed of a multi-tier architecture with three layers: *service layer*, *network function layer* and *infrastructure layer*. More in details:

- the service layer interacts with the network business entity providing a unified vision of the network. The actor of this layer are operators, verticals, enterprise and third parties;
- the network function layer is in charge of the creation of each network slice according to service instance requests coming from the upper layer. It is composed of a set of network functions (characterized by a well-defined behaviour and interface to exchange data with the others). An e2e network slice instance can be composed by multiple network functions chained together. The configuration of the network functions is performed using a set of network operations that allow management of their full life-cycle

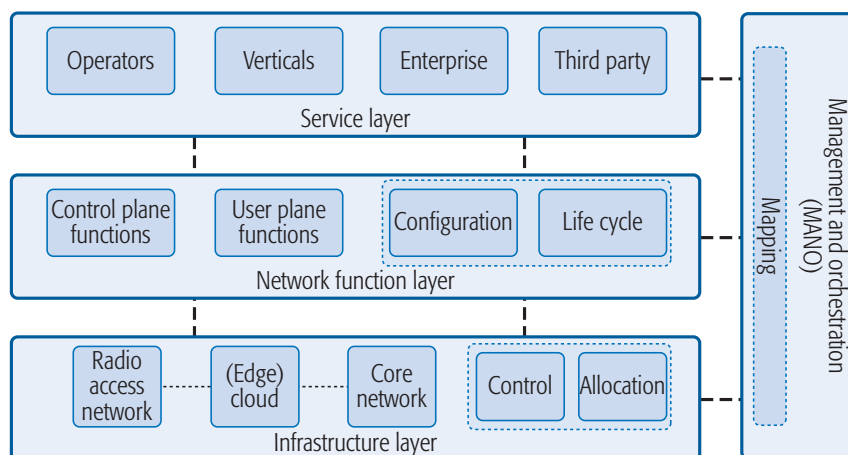


Figure 2.9: Network slice architecture [29]

(from their placement when a slice is created to their de-allocation when the function provided is no longer needed). The same network function can be simultaneously shared by different slices: this comports an increment in resource usage efficiency, but an increase in the complexity of operations management;

- the infrastructure layer represents the actual physical network topology (radio access network, transport network and core network) upon which every network slice is multiplexed and it provides the physical network resources to host the several network functions composing each slice.

The network slice controller is a network orchestrator, which interfaces with the various functionalities performed by each layer to manage each slice request. It enables an efficient and flexible slice creation that can be reconfigured during its life-cycle.

The characteristic of the slices of being mutually isolated from each other guarantees not only that there is maximum security of communications within the slice, but also the possibility of modifying the operation of one slice without impacting that of the others. This means, for example, that an operator can drastically change an offered service to the customers by acting only on that in a targeted way. In each layer described before, isolation of slice is managed in three different ways: language-based isolation (at the network function layer and service layer), physical-based isolation (applied at the infrastructure layer) and virtual machine-based isolation (performed at the resource layer and network slice instance layer).

A key role in the creation of network slices is played by SDN and NFV, and

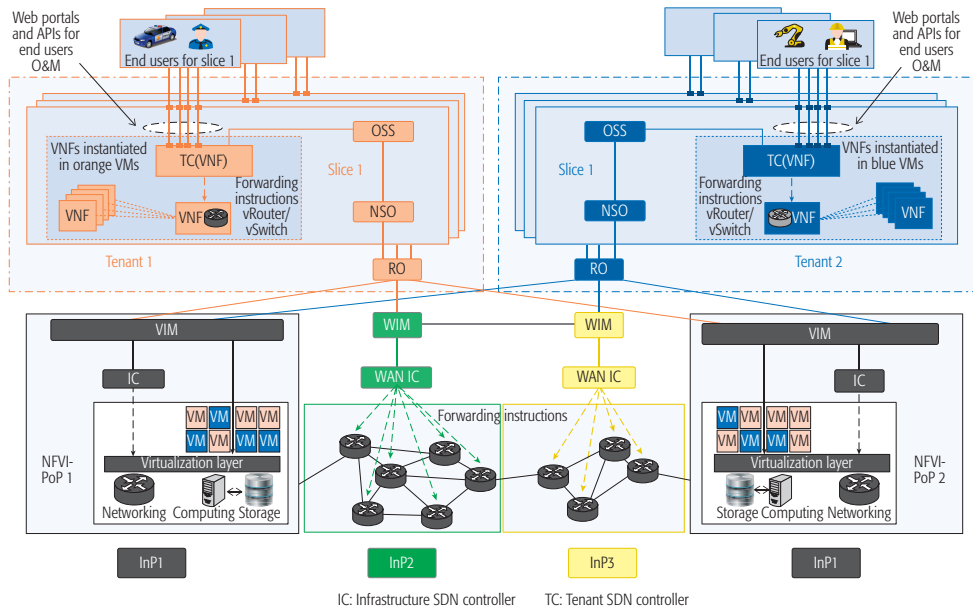


Figure 2.10: Network slicing deployment integrating SDN and NFV [27]

techniques like edge computing, cloud computing that, combined appropriately, help the network to guarantee application requirements.

Fig. 2.10 shows an example of deployment of a network slice [27], considering the integration of NFV and SDN paradigms. There are three *Infrastructure Providers* (InP), that own and manage physical network and its constituent resource. In this case, InP1 provides compute and networking resources, while InP2 and InP3 make available SDN-based Wide Area Network (WAN) as transport network. Inside the InP1 block there are the following elements:

- *VIM*: it is in charge of controlling and managing the Network Functions Virtualization Infrastructure, a set of resources used to host and connect VNFs;
- *Infrastructure SDN controller* (IC): it manages the networking resources, and is controlled by the VIM. It may modify the infrastructure behaviour on demand according to VIM specifications.

About InP2 and InP3, also in these cases there are two blocks:

- *WAN Infrastructure Manager* (WIM): it plans, develops, and manages the WAN and overall network infrastructure to deliver the required connectivity, availability, and performance;
- *WAN IC*: with the same role of the previous IC, but in the WAN transport network.



At the top level there are the *tenants*. This entity leases virtual resources from InP in the form of a virtual network that uses to realize, manage, and provide network services to its users. A tenant can realize one or more network slices using the underlying resources. Inside the tenant, there is one *Resource Orchestrator* (RO) that communicates with VIM/WIM and orchestrates the resources in an efficient way to satisfy the required performance of each slice. To do this, the RO interacts with the *Network Service Orchestrator* (NSO). This component has the role to perform the life cycle management of each network slice. This structure helps the RO to maintain isolation among slices.

Finally, there are the *OSS* and the *Tenant SDN Controller* (TC). The OSS is an element of the general Network Management System and represents a collection of systems and management applications used by each NSO to provide services. The TC manages the VNF life cycle and the composition of the VNF using the capability of virtual switches/routers (in the southbound interface), based on the instruction received by the OSS in the northbound interface.

## 2.4 Cloud Technology

### 2.4.1 Cloud vs Edge

The technological evolution of devices connected to the network, together with their use in applications with very specific requirements, has created the need for guarantee very stringent characteristics as regards the interaction and connection between where the data are produced and where they are processed. The evolution of modern IT systems has led to the deployment of the current reference architecture for Cloud environments. The philosophy behind the Cloud establishes that the resources needed by the users, from simple data storage capacity to the provision of entire operating systems, must be on the network without knowing how they are organized or where they physically reside. The National Institute of Standards and Technology (NIST) has defined *Cloud Computing* as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [30].

The main architecture of Cloud systems derives from a general taxonomy, drawn up by NIST, which lists all the interacting parts of the model completely. Although it is not the only one [31], it comprehensively classifies all the main types of Cloud. The reference framework that presents a high-level architecture that serves as a model for developing cloud platforms is shown in Fig. 2.11.

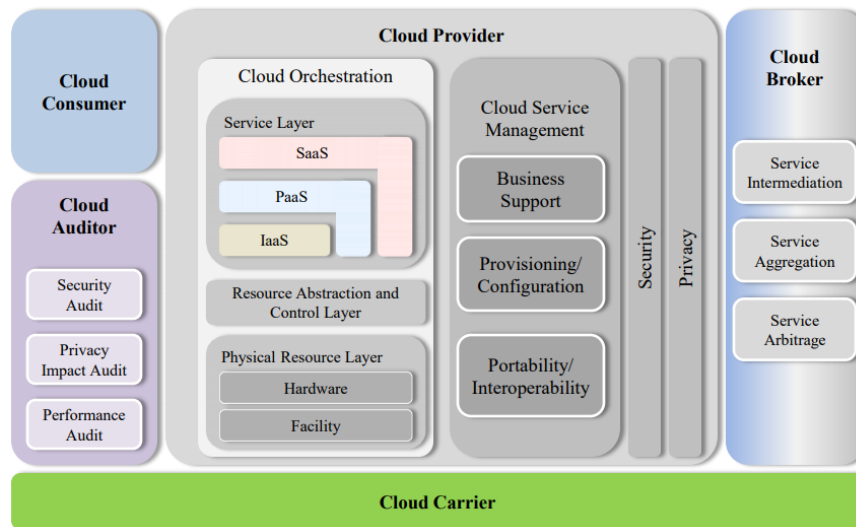


Figure 2.11: Cloud Architecture Model [30]

As shown in the figure, in a cloud environment it is possible to consider five actors [30] [32]:

- *Cloud Consumer*: it is the principal stakeholder for the cloud computing service. It could be either a person or an organization that can browse a list of all the services available in a Cloud Provider and, through the use of Service Level Agreements, specify the requirements (for example, in terms of quality of service and security) that have to be met by this one, during the use of a particular service. Each Cloud Consumer is different from another, based on the type of required service. This means that the interactions between Cloud Provider and Cloud Consumers may differ from each other;
- *Cloud Broker*: it deals with the relationship between the Cloud Provider and Cloud Consumer, negotiating the agreement and managing the performance and the delivery of services to Consumer. In this way, the Cloud Consumer may request cloud services from the Broker, avoiding to contact the Provider directly. In particular, the Cloud Broker can provide services in three different ways: 1) service intermediation, when the Broker enhances a service by improving specific capabilities in term of managing access, performance reporting, etc; 2) service aggregation, in which the Broker combines and integrates multiple services into one or more new service; 3) service arbitrage, when a Broker provides a flexible composition of service chosen possibly from multiple providers selected statically or dynamically based on technical as well as cost aspects;

- *Cloud Carrier*: it provides connectivity and transport of services between the Provider and the Customers;
- *Cloud Auditor*: it can perform an independent examination of services offered by the Provider, to verify conformance to standards in terms of security controls, privacy impact, performance, etc;
- *Cloud Provider*: it acquires and manages the computing infrastructure needed to provide services, choosing the necessary policies to deliver these services to the Consumer through the network. Its main activities regard the following areas: Cloud Orchestration, Cloud Service Management, Security and Privacy. More in detail, about the Cloud Orchestration, it is composed by three layers: 1) *Physical Resource Layer*, that includes the computing resources (hardware resources, network resources, storage components and other physical computing infrastructure elements; 2) *Resource Abstraction and Control Layer*, containing the system components needed to provide and manage access to the Physical Resource Layer (hypervisor, virtual machines, etc); 3) *Service Layer*, where the interfaces to allow to Customers to access the computing services are defined.

About the Service Layer, it is possible to define three different types of services provisioning [33]:

- *Infrastructure as a Service* (IaaS): it provides virtualized computing resources over the Internet. Instead of having to purchase the hardware outright, users can purchase IaaS based on consumption, similar to electricity or other utility billing. They are responsible for managing applications, data, runtime, middleware, and OS. What users gain with IaaS is infrastructure on top of which they can install any required platform;
- *Platform as a Service* (PaaS): it provides a framework in which the user can build upon to develop or customize applications. PaaS makes the development, testing, and deployment of applications quick, simple, and cost-effective;
- *Software as a Service* (SaaS): the Provider deploys, configures, maintains and updates the software applications on cloud infrastructure so that the services are provisioned at the expected service levels to Cloud Consumers through a web browser.

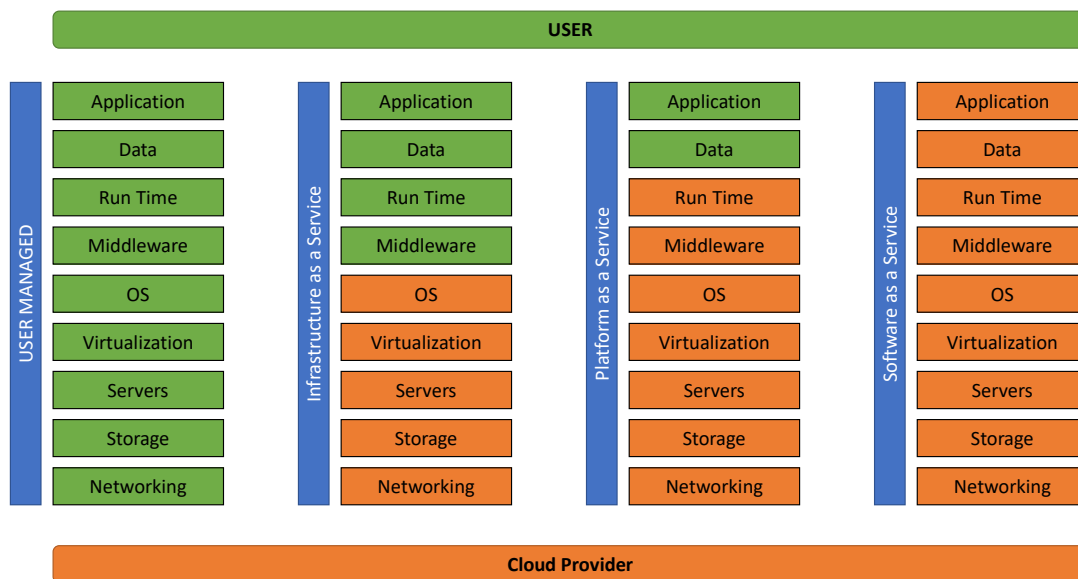


Figure 2.12: Differences between a User Managed, IaaS, PaaS and SaaS deployment

In Fig. 2.12, a comparison between the above types of described service provisioning and the case in which the user deploys the application without cloud technology, is shown.

The increasing use of Cloud Computing is also to be associated with the continuous development of the IoT, of which the Cloud is the fundamental enabling technology because it allows the execution of tasks otherwise impossible to complete with the use of only the available resources on devices (both in human-held smart devices and sensors/actuators).

Moving part of the computing to the Cloud is a fundamental application choice to mitigate the negative effects of resource scarcity. However, IoT means also continuous and fast real-time processing that concerns, in some cases, big data, with very low response times requirements. This means that some limits of the Cloud are highlighted [34]:

- latency: novel applications in the IoT scenario have high real-time requirements. Sending data to the Cloud and receiving a response increases the system latency; furthermore, the Cloud acts on extended WAN that could contain bottlenecks or interruptions along the path that connects the user to the Cloud platform, due to both malfunctions and the presence of shared access resources between multiple users. All these aspects cause an increment of the system latency;

- bandwidth: some type of applications can produce huge amounts of data, that have to be transmitted through the network, causing great pressure on network bandwidth. Just as an example, Boeing 787 generates more than 5 GB/s of data, but the bandwidth between an aircraft and satellites is insufficient to support real-time transmission;
- availability: more services are deployed on the Cloud, and users want to be free to use it at any time of day. For example, let image if an application like Siri or Google Maps is unavailable for a short time, the Quality of Experience (QoE) of the users will decrease. Therefore, it is a big challenge for Cloud Providers to keep the  $24 \times 7$  promise;
- energy: data centers consume a lot of energy and with the increasing amount of computation processes and transmission, energy consumption will become a relevant problem;
- security and privacy: data, having to travel to remote clouds, can be subject to interception along the way. Being able to guarantee security and privacy all along the path can represent a huge challenge for the Cloud Provider. In applications such as video surveillance, this is a major problem that can discourage the end-user from using a given application. The application of the EU General Data Protection Regulation (GDPR) has introduced rules and responsibilities on data processing that must be respected by the Cloud Provider;
- mobility support: the Cloud is not suitable for mobile users who often change access points to the network, with continuous handover to switch from one base station to another, which involves managing the transfer of big data that must be as accurate as possible, so as not to have data loss [35].

These described limits lead to the impossibility of using some applications that require stringent requirements of latency and available bandwidth on cloud platforms. An example is augmented reality which, through sensors, microphones and video cameras, requires a continuous sending of a flow of information to a central server with unlimited resources, which, after having processed and enriched them with further details, sends answers that allow the reconstruction of entire real environments in the user's device. Also telesurgery, assistive applications for users with disabilities, autonomous driving in a vehicular network, are examples of applications that require very low latency and high computational resource

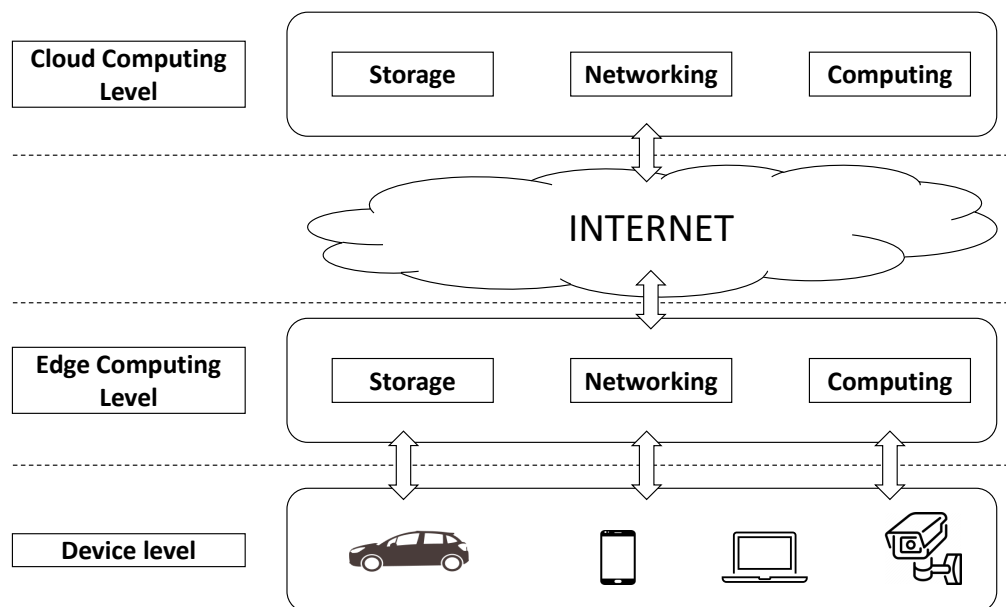


Figure 2.13: Cloud Computing and Edge Computing Architecture

available. In [36] the Authors show, with numerical results, how an extension of the Cloud towards the data production point, and therefore towards the users, is a necessary and effective solution, which allows to significantly lower network latency, increasing not only the Quality of Service (QoS) but also the QoE of the end-users. This extension to the user is called *Edge Computing*.

Edge Computing refers to a wide range of techniques designed to move computing, processing and storage resources out of the remote cloud (public or private) and closer to the user, decreasing latency and reducing the traffic load on the core network. The edge of the network is usually located just one hop away from end devices: for this reason, it is able to offer ideal placement for low-latency offload infrastructure to support emerging applications such as augmented reality, connected and autonomous driving, smart manufacturing, and healthcare. The Edge Computing architecture is able to reduce the amount of data to be sent to the Cloud as the data are processed by the device itself (smart device) or by edge server, instead of being transmitted to the data center; the less 'time-sensitive' data can instead be transmitted to the cloud infrastructure or to the company's data center, to allow more complex processing, such as big data analysis, training activities to refine the learned model of machine learning (ML), long-term storage and historical data analysis.

Fig. 2.13 shows the three-level architecture generated by the introduction of Edge Computing.

It is important to underline that, in some cases, the same devices constituting

the Device Level can provide edge computing functionality. For example, in a vehicular scenario with vehicles equipped with sensors for the collection of information from the surrounding environment, a vehicle that is in difficulty in processing all its data, could ask for help from a nearby vehicle, which providing its available computing resources, behaves like an edge computing server. The same thing can happen in the use of UAVs for particular application scenarios. If a UAV is too loaded, it could exploit the resources of a nearby UAV. The ability for devices to send all or part of the computational load to other nearby devices or servers at the edge, gives rise to the need to implement appropriate offloading policies. Some solutions about this aspect have been proposed in the UAV and vehicular contexts, and will be presented in Chapters 3 and 4.

### 2.4.2 Multi-Access Edge Computing

In December 2014, ETSI initiated the standardization of Mobile Edge Computing to promote and accelerate the advancement of Edge Computing in mobile networks by launching the MEC Industry Specification Group (ISG). Since September 2016, the ETSI ISG has removed "Mobile" from the name and renamed *Multi-access Edge Computing* (MEC), with the aim of expanding its applicability to heterogeneous networks, which not only concern mobile but also WiFi and fixed access technologies [37].

ETSI ISG MEC aims to standardize a framework to run third-party applications at the edge of the network, providing a dynamic environment based on cloud technology. Furthermore, by deploying applications at the edge level, it is possible to also benefit from services provided locally such as knowledge of the network access point and information on the radio network. Based on Edge computing, the MEC can be thought of as a Cloud Server, installed near mobile devices, to carry out specific operations that are not possible to complete with traditional Cloud infrastructures. The technological development of the components installed on the network equipment has allowed a considerable expansion of the number and type of services that can be installed at the Edge. Furthermore, mobile operators are able to open their infrastructure to third parties, such as developers who want designing their applications aware of interfacing with an environment characterized by low latency. For their part, the providers have to supply evolved RAN devices, in order to increasingly enrich the functionality of the services.

The introduction of MEC is connected to the natural evolution of the mobile base stations toward 5G systems, enabling software-based mobile edge applications and cloud computing services at the network edge. The MEC platform

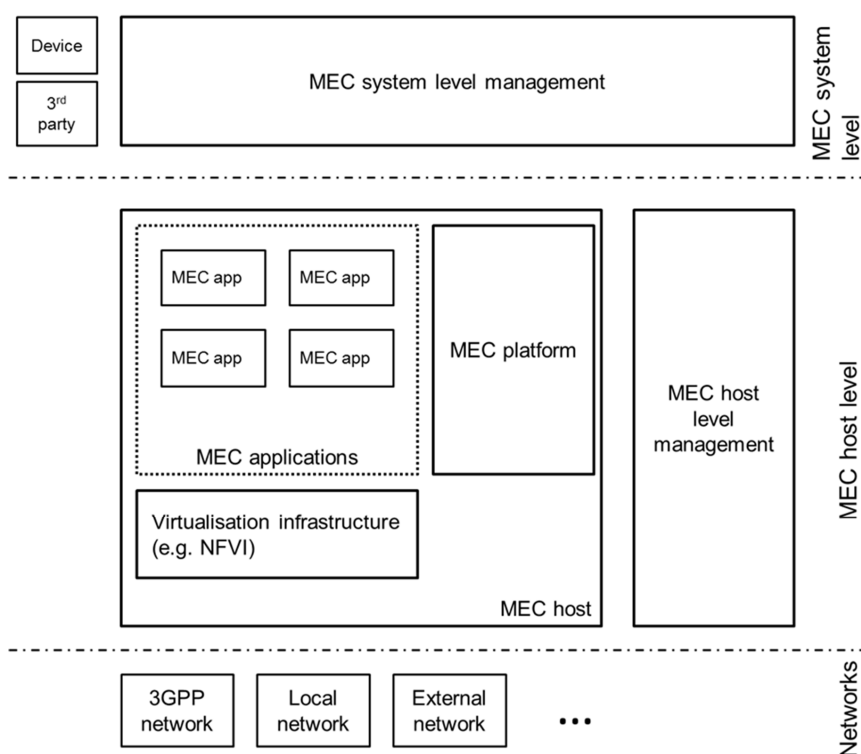


Figure 2.14: ETSI MEC framework [38]

can provide cloud storage, caching, computing, proximity benefits of resource provisioning, context and location awareness, agility and speed to the mobile applications [37].

The enrichment of network functionalities is transparent to the end-user, who benefits from it seeing an increase in QoS and QoE. In particular, thanks to MEC technology, there is the possibility to offer a programmable ecosystem that changes the user experience. The ability to customize the offered services based on the context and network conditions allows applications to self-configure if and when needed. The applications become software-only entities that run on top of virtualized infrastructure.

In Fig. 2.14 there is the MEC framework proposed by ETSI, in which it is possible to see how all the entities and functionalities of MEC are grouped in three levels:

- *MEC system Level*: it provides an abstraction of the underline MEC environment;
- *MEC host Level*: it is constituted by the core components of the MEC framework;
- *Network Level*: it aims at providing the network resources to the top levels.



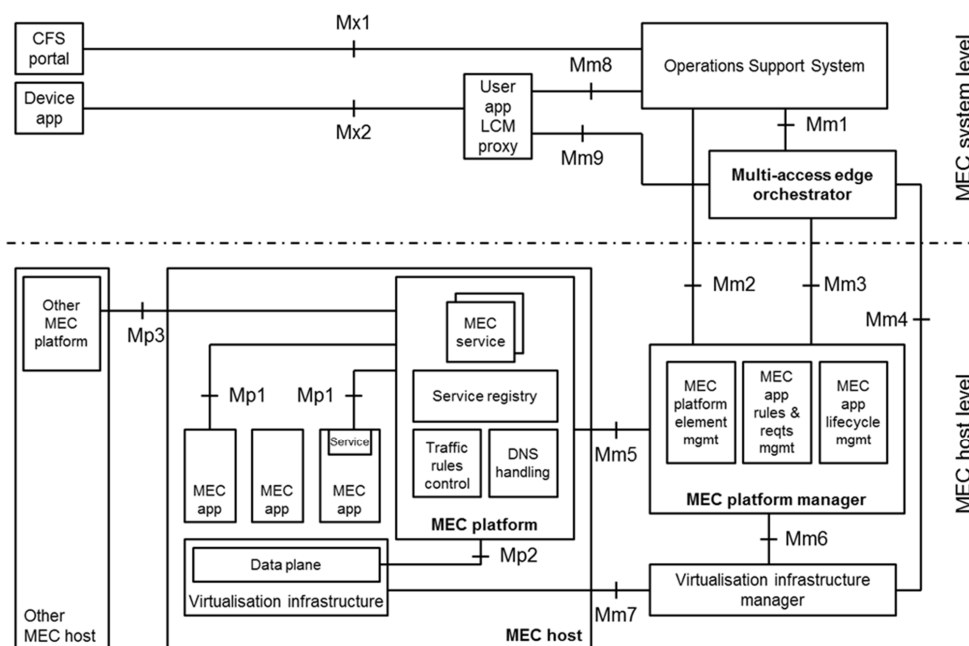


Figure 2.15: ETSI MEC reference architecture [38]

Let see that, following the definition of "multi-access", in this level different types of networks are present.

The MEC reference architecture is shown in Fig. 2.15, in which more details about the first two levels described before are presented [38]. In the figure also the standardized interfaces (or reference points) are highlighted: the interface is named Mp if it regards MEC platform functionalities, Mx if it allows communication with external entities, otherwise Mm if it is used to exchange management information.

About the MEC system level, it is composed by the *Multi-access edge orchestrator* (MEO) and the OSS.

The first one is responsible for: 1) maintaining an overall view of the MEC system in terms of MEC hosts, topology, MEC services and available resources; 2) managing the entire life cycle of the applications, starting from the on-bording of application packages (checking the integrity and authenticity of the packages, validating application rules and requirements), then preparing the virtualization infrastructure manager of the underline level (using the Mm4 interface) to handle the applications, and selecting the MEC host(s) in which run the application (based on constraints like latency, available resources and services), triggering application instantiation and relocation (if needed) until its termination.

The OSS, as described before, is an operator's block that receives requests about the instantiation or termination of an application through the Customer Facing

Service (CFS) portal and from device applications, decides if accept or not these requests and, if so, forwards these to the MEO using the Mm1 interface.

In the MEC host layer there are the *MEC host*, and the *MEC host level management* that comprises the *Virtualization Infrastructure Manager* and the *MEC platform manager* (MEPM). More in deep, the MEC host is composed by:

- *MEC platform*: it offers an environment where the MEC applications can discover, use and offer MEC services, even if these services are available in other platforms (through Mp3 reference point);
- *Virtualization infrastructure*: it provides compute, storage and network resources for MEC applications and include a data plane that executes the traffic rules received by the MEC platform;
- *MEC applications*: they run as VM on top of the virtualized infrastructure. They interact with the MEC platform through the Mp1 reference point) to offer or use services already deployed. Each application is characterized by particular requirements as latency, required service and bandwidth: as said before, the MEO of the top level validates these requirements based on the available resources before to start the instantiation of the service in a selected MEC host. Each MEC application can also provide services to other MEC applications.

The MEPM manages the life cycle of applications and informs the Multi-access edge orchestrator about particular events of these applications. Moreover, it manages the application rule and requirements and provides management functions to the MEC platform. It uses the Mm5 interface to communicate with MEC Platform inside the MEC host, the MM2 to exchange information with the OSS about fault, configurations and performance management purposes and the MM3 to communicate with MEO.

About the Virtualization Infrastructure Manager, it allocates and manages the virtualized resources to run a software image, and collects and reports performance and fault information about the applications that use these resources. This information is communicated to the MEPM through the Mm6 interface. If supported, this component performs application relocation and rapid provisioning of new applications. When needed, it manages the release of the used resources.

In the MEC context just described, NFV technology plays an important role in the management of virtualized MEC applications within the MEC host, considering also that these applications exploit the virtualized resources of the Virtualization Infrastructure. NFV introduces the possibility to increase flexibility,

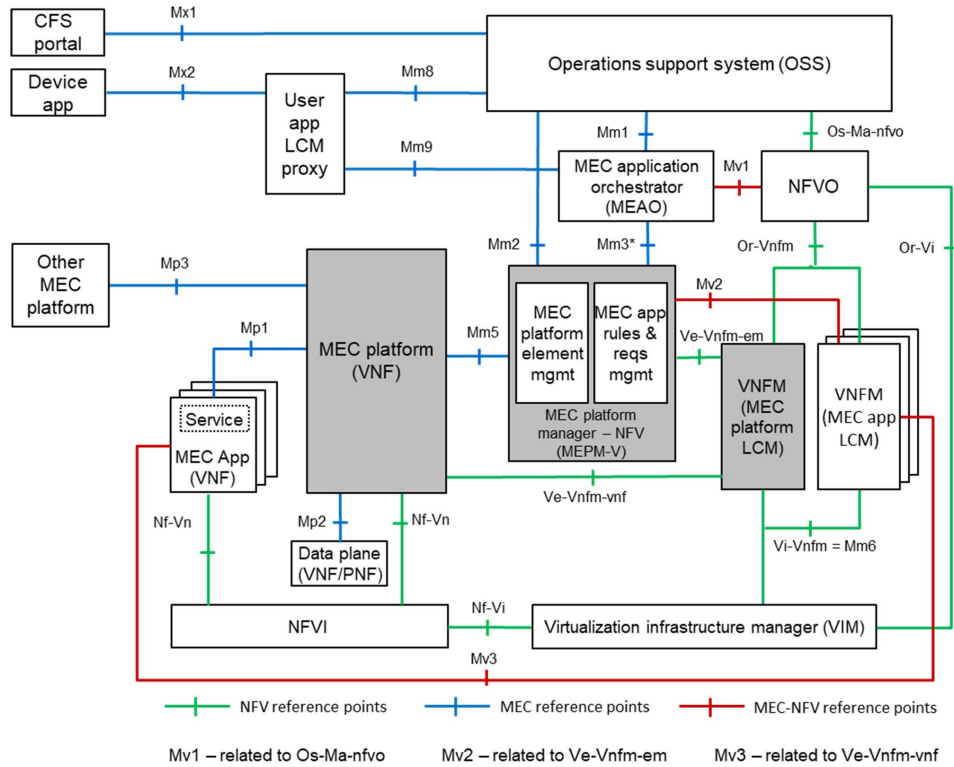


Figure 2.16: ETSI MEC reference architecture in NFV [38]

scalability and to migrate the service, especially in contexts where the user who asks for a service is moving from a MEC server to another. For this reasons, ETSI in [38] also provides a MEC architecture variant (2.16). Thanks to this architecture, MEC applications and NFV functions could be deployed on top of the same virtualized infrastructure. This integration between MEC and NFV adds the possibility to reuse ETSI NFV MANO components to fulfil part of the MEC management and orchestration. NFV dynamic aspects that can benefit MEC services include: 1) portability, because independent blocks of all services can be easily moved to another cloud environment; 2) federation support, that allows deploying portable functions over inter-operable geographically distributed virtual networks; 3) slicing through partitioning of virtual network resources for particular applications; 4) sharing a pool of configurable resources for on-demand access [37].

In a comparison between Fig. 2.15 and Fig. 2.16, the following differences can be highlighted [39]:

- MEC platform is deployed as a VNF;
- MEC applications appear as VNFs towards the ETSI NFV MANO components. This allows re-use of ETSI NFV MANO functionality. It is, however,

expected that ETSI MEC might not use the full set of MANO functionality, and requires certain additional functionalities;

- the Virtualization infrastructure is deployed as an NFVI and is managed by a VIM as defined by ETSI GS NFV 002 [12];
- the MEPM is replaced by a *MEC platform manager-NFV* (MEPM-V) that delegates the VNF lifecycle management to one or more *VNF managers* (VNFM);
- the MEO is replaced by a *MEC application orchestrator* (MEAO) that relies on the *NFV orchestrator* (NFVO) for resource and MEC application VNFs orchestration.

The reference points Mv1, Mv2 and Mv3 are introduced between elements of the ETSI MEC architecture and the ETSI NFV architecture to support the management of MEC application VNFs.

## Chapter 3

# Management and orchestration of network slices

In recent years, the IoT has transformed objects of everyday life into communicating devices. 5G networks design and develop new management capabilities to meet the stringent requirements of future use cases. In a scenario in which there will be an enormous number of antennas and a lot of new functionalities, the management and orchestration of resources play a central role. Placing resources on the edge of the network, extending the cloud computing paradigm, is useful to deal with the eminent growth of connected devices [40]. Due to the limited availability in the current deployments, computing power, storage and memory capacity are moved closer to access nodes, sensors and actuators. All this is possible only with a proper resource allocation performed by the Network Orchestrator, but it is not an easy task.

Techniques of orchestration of the resources inside the data center are for a long time studied and developed, reaching by now the optimal maturity. The same cannot be said regarding the implementations of such techniques in the edge nodes: the limited resources availability but the need to ensure a lot of requirements and constraints (like ultra-low latency, delay, bandwidth, energy efficiency and so on) complicate the application of the same data center techniques in an edge scenario. In a smart city scenario, resources should be distributed within the network ensuring that they are allocated and instantiated close to the end device that is requesting an application.

This chapter addresses the problem of management and orchestration of network slices. In particular, in Section 3.1 the handover management in the RAN portion of a network slice is described. In this context, 5GHander was realized. It is a NS that, with the aim of detecting handovers performed by the end-users or by

any device connected to the mobile network, can help the Network Orchestrator to implement resource allocation technique in a preventive way, or block the handover if there is a low availability of resources and the handover is not really necessary, for example if the user measures a little higher receiving power from a cell rather than the one in which he is currently located. In Section 3.2 the use of UAV to extend a network slice is proposed. This solution will provide computing and network facilities to IoT devices in area monitoring applications.

### 3.1 5G-Hander for handover detection

For 5G networks, as in 4G communications systems, one of the most challenging problems will be managing handovers and their consequences in the status of the network. In fact, each handover event related to a mobile device causes a redirection of both uplink and downlink traffic flows generated by and directed to that device. In a softwarized network with a centralized control, knowing that handover of a given flow is upcoming will allow the Network Orchestrator to reorganize resources or deciding the allocation of new resources in terms of computing, networking and storage. Alternatively, the Network Orchestrator, when informed about an incoming handover, can split flows on different parallel paths with different throughput performance or, if necessary and possible, delaying it by intercepting messages between two base stations. Independently of the nature of the RAN, which can be either Distributed (D-RAN) or Centralized (C-RAN), detecting handover events without the cooperation of the involved devices will allow Telco Operators to use legacy and vendor-independent self-consistent devices.

To this purpose, in the paper [41], accepted for publication on a Special Issue on Internet Technology Letters (Wiley), a virtual NS called *5G-Hander* (5G Handover Detector) was presented. It is aimed at capturing autonomously, i.e. with no cooperation with other physical or virtual devices, information relating to the handover events in the RAN or Backhaul Network where it is running.

The system considered as reference is a 5G transport network consisting of integrated fixed and wireless network infrastructures, as specified by ETSI-NFV-Group [42]. The wireless part of the framework is constituted by the mobile core network and a set of RANs deployed on the same geographic zone. On the other hand, the fixed portion of the network comprises core, metro and fixed access domains. While the fixed core network and the metro network are usually realized with optical transport, the access network uses a heterogeneous set of transport technologies.

Both kinds of handover are supported, that is, X2 [43] and S1 [44]. The X2 handover occurs when the involved eNodeBs (eNB) are connected to the same Mobility Management Entity (MME) in the Core Network; the S1 handover occurs when the eNBs are connected to either different MMEs, or to the same MME but have not established the X2 interface.

Fig. 3.1 and Fig. 3.2 depict two relevant applications scenarios: a D-RAN, made of a set of eNBs connected to each other and to the mobile core network through a Backhaul Network, and a scenario constituted by a portion of C-RAN coexistent with a D-RAN. While in a D-RAN the same eNB contains both a Remote Radio Head (RRH) and a BaseBand Unit (BBU), in a C-RAN, the antenna sites of the RRH elements are simplified given that base-band processing is performed by virtualized instances running on servers centrally placed on a data center in the C-RAN cloud. This allows C-RANs to achieve an easier installation of small RRHs, also allowing reducing the overall base-band processing resources due to the statistical gain of centralization.

In the first scenario shown in Fig. 3.1, there are three eNBs managed by the MME1, while the fourth eNB is managed by the MME2. For this reason, eNB1, eNB2 and eNB3 communicate to each other through their X2 interfaces, while communicate with the eNB4 through their S1 interface.

The second scenario shown in Fig. 3.2, is similar to the previous one, but with the first two base stations virtualized as RRH1/BBU1 and RRH2/BBU2. Therefore, the first two base stations communicate to each other through X2, and with eNB4 through S1. Let us notice that, although not in Fig. 3.2, this scenario also captures the case of Coordinated MultiPoint (CoMP) transmission from more than one RRH to the same BBU.

The 5G-Handler proposed in the paper is highlighted in Fig. 3.1 and Fig. 3.2 for both the above scenarios. It is realized as a virtual NS constituted by the chain of two VNFs, the Handover Sniffer and a Broker. We assume that the networks inside the C-RAN data center and the Backhaul Network are SDN compliant. In this way traffic flows can be routed programmatically in such a way that all traffic carrying X2 and S1 information are also routed towards the 5G-Handler installed in the same network.

In the case of the first scenario, the VNFs composing 5G-Handler are installed on a server connected to the SDN network infrastructure of the Backhaul Network. Instead, in the second scenario, besides their installation as in the first scenario, two instances are run on a server of the C-RAN data center, which has been installed together with the other servers of the BBU Pool. In this way, it is able

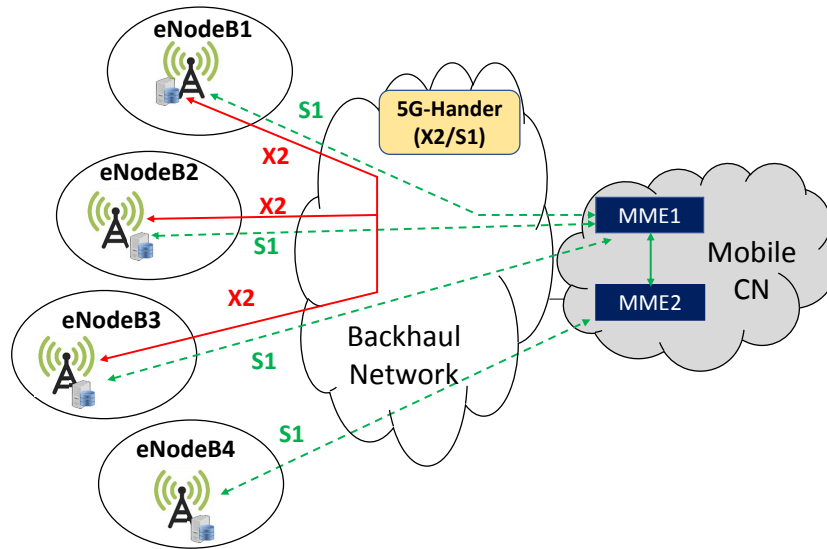


Figure 3.1: 5G-Handler. Distributed RAN scenario

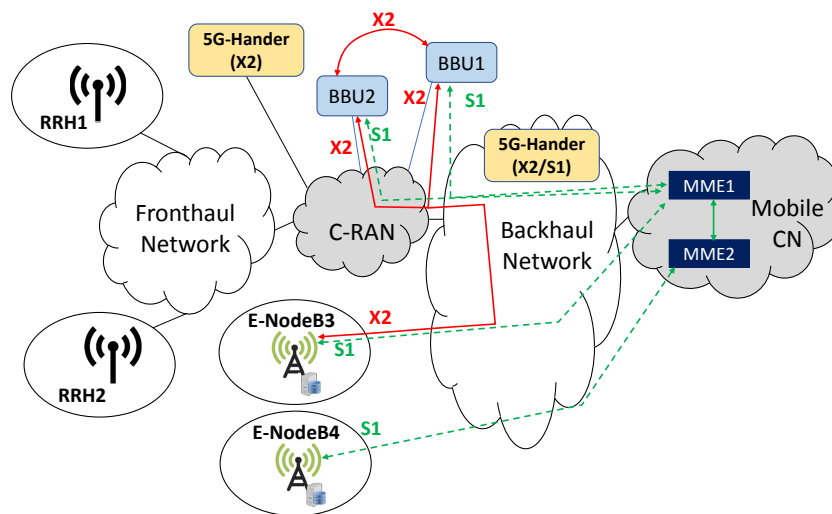


Figure 3.2: 5G-Handler. Centralized RAN scenario

to capture the X2 traffic flowing between BBUs running in the same C-RAN. Instead, the 5G-Handler running in the Backhaul Network allows to capture two types of traffic: all the S1 traffic among virtual and physical base stations managed by different MMEs, and X2 traffic among the base stations virtualized in the C-RAN, and the physical ones directly connected to the Backhaul Network.



### 3.1.1 Definition and implementation

As already said, the 5G-Hander network service has the goal of analyzing packets generated by all the base stations in order to find packets reporting handover events of both S1 and X2 handover types. This information is redirected, via the Broker within the 5G-Hander, to the Network Orchestrator, which analyze them and can decide whether to intervene by inhibiting the handover or not. The Handover Sniffer works by analyzing all the traffic generated by the base stations on their X2 and S1 interfaces. To this purpose, the SDN switches that are present in both the backhaul and the C-RAN networks have to be configured by setting up a specific rule on the local SDN Controller, in such a way that all the X2/S1 traffic is also routed to the Handover Sniffer.

The Handover Sniffer VNF captures data traffic packets by using the functions of the *pcap.h* library [45]. More specifically, it works according to the algorithm shown in Fig. 3.3. First (lines 1-2)), through the functions *pcap\_compile* and *pcap\_setfilter*, a filter is applied in order that the Handover Sniffer VNF focuses on Stream Control Transmission Protocol (SCTP) packets only, because SCTP is the protocol used by the application-layer protocols S1 and X2 [43, 44, 46]. The SCTP packet structure is shown in Fig. 3.4

For each SCTP packet, the Handover Sniffer needs to analyze the chunks contained in its payload (lines 7-14). For all the chunks labeled as *data chunks*, that is, identified by a *chunk-type* field equal to 0, the *Payload protocol identifier* field is analyzed (lines 15-16). Its value gives the information regarding the specific type of packet contained in the *Data* field: X2 handover packets are identified with the number 452984832, while S1 handover packets with the number 18. The data chunk structure is shown in Fig. 3.5

The handover packet is contained in the SCTP *Data* field. The information regarding the handover phase during the complete handover cycle (from the *handover request* to the *context release*) is contained in the *message\_type* field in the handover packet header (see Fig. 3.6 for both X2 and S1 packets), while the other information elements are contained in its payload (lines 17-18). Detection of the specific protocol message, i.e. Handover Request, Status Transfer and Context Release for the X2 protocol, and Handover Preparation, Handover Notification and Path Switch Request for the S1 protocol, is described in lines 19-30.

Finally, the last step is sending the results of the traffic analysis to the Broker VNF. Specifically, the following list of information is sent: the User Equipment (UE)-Id identifiers used by both the source base station and the target base station of the handover event, the information elements regarding the cell (the

```

Begin
1. input interface
2. filter ← “sctp”
3. do
4.   input packet
5.   ih ← pointer to header of IP packet
6.   sh ← pointer to header of SCTP packet
7.   if chunk != null then
8.     sc ← pointer to the chunk of SCTP packet
9.     c_type ← chunk_type
10.    if c_type = 0 then goto 15
11.    else
12.      chunk ← next_chunk
13.      goto 7
14.    else goto 32
15.  sd ← pointer to chunk DATA
16.  protocol ← sd.payload protocol identifier
17.  m ← pointer to message header
18.  mt ← m.message_type
19.  if protocol = X2 then
20.    if mt = 0 then output Handover Request Msg
21.    else
22.      if mt = 4 then output Status Transfer Msg
23.      else
24.        if mt =5 then output Context Release Msg
25.  if protocol = S1 then
26.    if mt = 0 then output Handover Preparation Msg
27.    else
28.      if mt = 2 then output Handover Notification Msg
29.      else
30.        if mt = 3 then output Path Switch Request Msg
31.  packet ← next_packet
32. while next_packet != null
End

```

Figure 3.3: 5G-Handler. Algorithm 1: Handover Sniffer pseudo-code

Evolved Universal Terrestrial Radio Access Network (E-UTRAN) cell identifier and the Tracking Area Identifier), the Global Unique MME Identity (GUMMEI), the Public Land Mobile Network (PLMN) identity, the Mobile Country Code (MCC) and Mobile Network Code (MNC), and the User Equipment (UE) Context Information. All the other information elements that are captured by the Handover Sniffer are not sent to the Broker in the current implementation, but can easily be included.

The Broker VNF works as a gateway at the application level by using a *pub-*

Bits	Bits 0-7	8-15	16-23	24-31
+0	Source port		Destination port	
32	Verification tag			
64	Checksum			
96	Chunk 1 type	Chunk 1 flags	Chunk 1 length	
128	Chunk 1 data			
...	...			
...	Chunk <i>N</i> type	Chunk <i>N</i> flags	Chunk <i>N</i> length	
...	Chunk <i>N</i> data			

Figure 3.4: SCTP packet structure

Bits	Bits 0-7	8-11	12	13	14	15	16-31
+0	Chunk type = 0	Reserved	I	U	B	E	Chunk length
32	TSN						
64	Stream Identifier					Stream Sequence Number	
96	Payload Protocol Identifier						
128	Data						

Figure 3.5: SCTP Data chunk structure

Bits	Bits 0-7	8-15	16-23
+0	Flag	Message type	Criticality

Figure 3.6: Header of the X2 and S1 packets

*lish/subscribe* approach. More specifically, the Handover Sniffer sends information retrieved during its analysis as messages to be published by the Broker. Then the Broker sends this information to the Network Orchestrator running in the Core Cloud, in such a way that it can allocate resources and take its decisions knowing the current position of the entrance/exit points of all the flows. Thanks to this approach, more than one Orchestrator can be registered to the same Broker to receive information relating to handover events, and specific filters can be applied to differentiate messages to be sent to each Orchestrator.

### 3.1.2 Testbed description and results

With the aim of verifying the behavior of the proposed 5G-Header, it was realized a testbed representing the D-RAN scenario described so far. It is constituted by

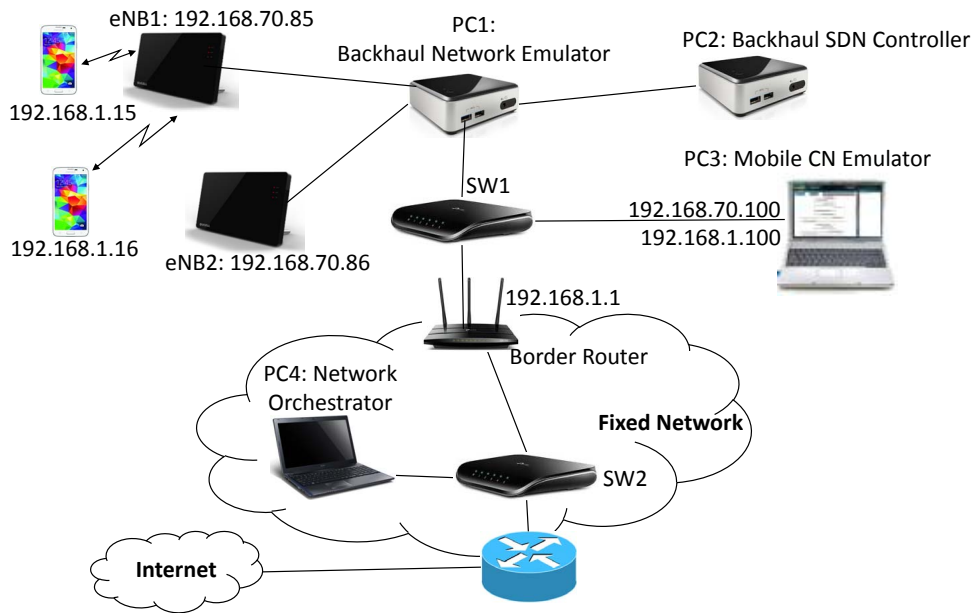


Figure 3.7: 5G-Hander. Testbed infrastructure

the following devices, as shown in Fig. 3.7:

- 2 4G Smartphones (Samsung Galaxy S5 with dummy sim card);
- 2 SmallCells Qualcomm LTE Band 3;
- 2 Computers (PC1 and PC2) INTEL NUKE miniPC, core I5, running respectively the backhaul emulator realized through the Mininet network emulator and OpenDaylight as the backhaul SDN Controller;
- 1 Computer (PC3) HP Laptop, core I5, with XCore tool emulating the Evolved Packet Core (EPC) Mobile Core Network;
- 1 Computer (PC4) ACER Aspire 5755G, INTEL core I7, running the Network Orchestrator;
- 1 Border Router for the connectivity from/to the Infrastructure Network;
- 2 network switches.

The two smartphones work as UE to realize the handover from one small cell to the other one. Their IP addresses are directly assigned by the Border Router of the Infrastructure Network. The two small cells work as eNBs to provide UEs with cellular connectivity. As specified by the Long Term Evolution (LTE) standard [47], they communicate to each other directly through the X2 protocol interface, with no involvement of the MME block inside the Mobile Core Network. Instead,

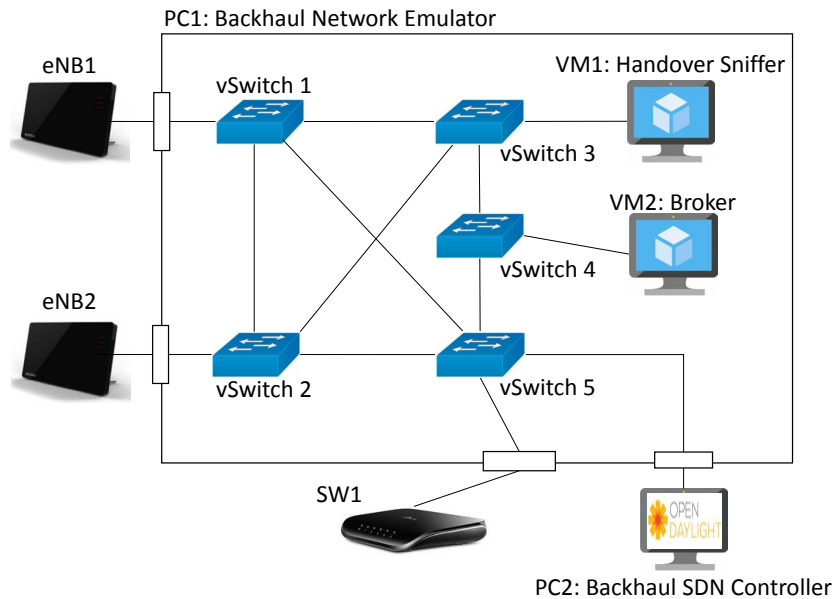


Figure 3.8: 5G-Hander. Network topology of Backhaul emulator

their communications with all the other components of the system follow the S1 protocol interface.

The PC1 node emulates the Backhaul Network. Its network topology, realized with the Mininet network emulator, is shown in Fig. 3.8. All the switches are based on OpenVSwitch, and are controlled by a SDN Controller through the OpenFlow protocol. More specifically, five virtual SDN switches are used in order to emulate the most general scenario where small cells, Handover Sniffer, Broker and Controller are installed remotely to each other. Of course, easier scenarios can be considered. Virtual Switches 1 and 2 are used to connect the small cells to the network. The link between these virtual switches is used for message exchanges between the small cells according to the X2 protocol.

Two Virtual Machines (VMs) are used: one running the Handover Sniffer VNF, connected to the Virtual Switch 3, and the other running the Broker VNF, connected to the Virtual Switch 4. Virtual Switch 5 is used to connect the Backhaul Network to the SDN Controller, and to the rest of the network through the physical switch SW1.

In PC3, simultaneous IP connections of the same physical network interface towards two different networks (i.e. 192.168.1.0 network of the Border Router and 192.168.70.0 network of the small cells) are realized through Linux IP port aliasing.

The Border Router allows the connection between the Mobile Core Network and

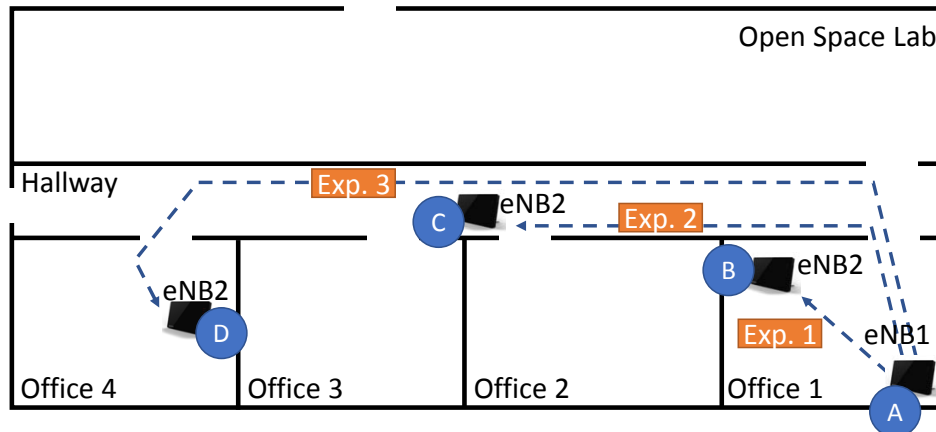


Figure 3.9: 5G-Hander. Experiment scenario

the Fixed Network. An important element of the Fixed Network is the Network Orchestrator, which manages and orchestrates the whole Telco Operator network. It is the destination of the information generated by the 5G-Hander network service. As shown in Fig. 3.7, it runs in the PC4 node.

Three different experiments have been executed to test the proposed 5G-Hander network service and, in particular, to highlight the behavior of the Handover Sniffer. The considered scenario is shown in Fig. 3.9. It represents the CNIT Riltus 5G Lab in the Campus of the University of Catania. In all the experiments, it was considered that eNB1 is placed in position A, while the position of the eNB2 is changed in the three cases, respectively in positions B, C and D. In these experiments, eNBs are connected to the same MME because we have only one EPC emulator, so next results are related to X2 handover type. In order to test the functionality of our 5G-Hander with S1 handover, we set a firewall for the X2-AP protocol, blocking communications on ports 36422 and 36423 used by this protocol to establish X2 connection and exchange messages. In this way, the eNBs were forced to communicate using S1 protocol even if connected to the same MME.

The purpose of these experiments has been to measure, during user's movements from one eNB to the other one, the Reference Signal Received Power (RSRP) values and the time instants in which the signaling messages due to the handover events are generated. RSRP represents one of the key measurement parameters of signal quality for the modern networks. As defined in [48], RSRP is the linear average over the power contributions (in W) of the resource elements that carry cell-specific reference signals within the considered measurement frequency bandwidth. During the experiments, a video transmission was started on the mobile

terminal via the YouTube app.

The smartphone continuously performs measurements relative to the channel currently used and the neighboring channels it can receive, and sends these measurements to the eNB where it is connected. This one analyzes the value of the RSRP field and, if it detects that the UE is receiving a signal with higher power from another cell, it starts the handover procedure with the eNB of that cell. During the experiments, the XCal software by Accuver was used to get the RSRP measurements that smartphone performs continuously. At the same time, the 5G-Hander network service captures the handover messages, also providing the time instants when these messages were exchanged among the eNBs and the EPC through X2 and S1 interfaces.

Fig. 3.10 show the RSRP values captured in a range of 90 seconds, during which a handover was performed by the UE. More in deep, in Fig. 3.10a it is possible to note that the smartphone is able to measure the RSRP values of both the cells since the beginning of the experiment. This is due to the fact that both the eNBs are located in the same room. On the contrary, in the first part of the other two experiments, the RSRP measurements of the eNB2 signal are not available because the source is too far from the UE. Moreover, we can observe that, as expected, the measured RSRP from the eNB1 (the starting point of the UE path) decreases in time, while the one measured from the eNB2 (the end point) increases.

Figs. 3.11a, 3.11b and 3.11c focus on the time period when the handover event occurs, ranging between 5 seconds before and 5 seconds after it. The time instants in which the handover messages are exchanged, sent by the VNF Hander to the Broker VNF, are highlighted in the figure. In particular, the attention was focused on four messages: *Handover Request* at  $t_1$ , *Status Transfer* at  $t_2$ , *Path Switch Request* at  $t_3$  and *UE Context Release* at  $t_4$ . For example, in Fig. 3.11a, it is possible to notice the first instant ( $t_1 = 28.38s$ ) in which the UE detects a stronger signal from the eNB2. This is notified to the eNB1, and the handover procedure starts on the X2 interface.

The other messages are exchanged with the Broker (at the instants  $t_{1,1}$ ,  $t_{1,2}$ ,  $t_{2,1}$ ,  $t_{2,2}$ ,  $t_{3,1}$ ,  $t_{3,2}$ ,  $t_{4,1}$  and  $t_{4,2}$ ), and this is done by using the publish/subscribe approach. Although during the execution of this procedure there are some instants in which the eNB1 signal is stronger than the signal coming from the eNB2, the procedure goes on to the end, until the instant  $t_4$ , when the eNB1 releases the resources and the UE connects to the eNB2. A similar behavior characterizes the other two experiments, as shown in Figs. 3.11b and 3.11c. In Table 3.1, messages

# CHAPTER 3. MANAGEMENT AND ORCHESTRATION OF NETWORK SLICES

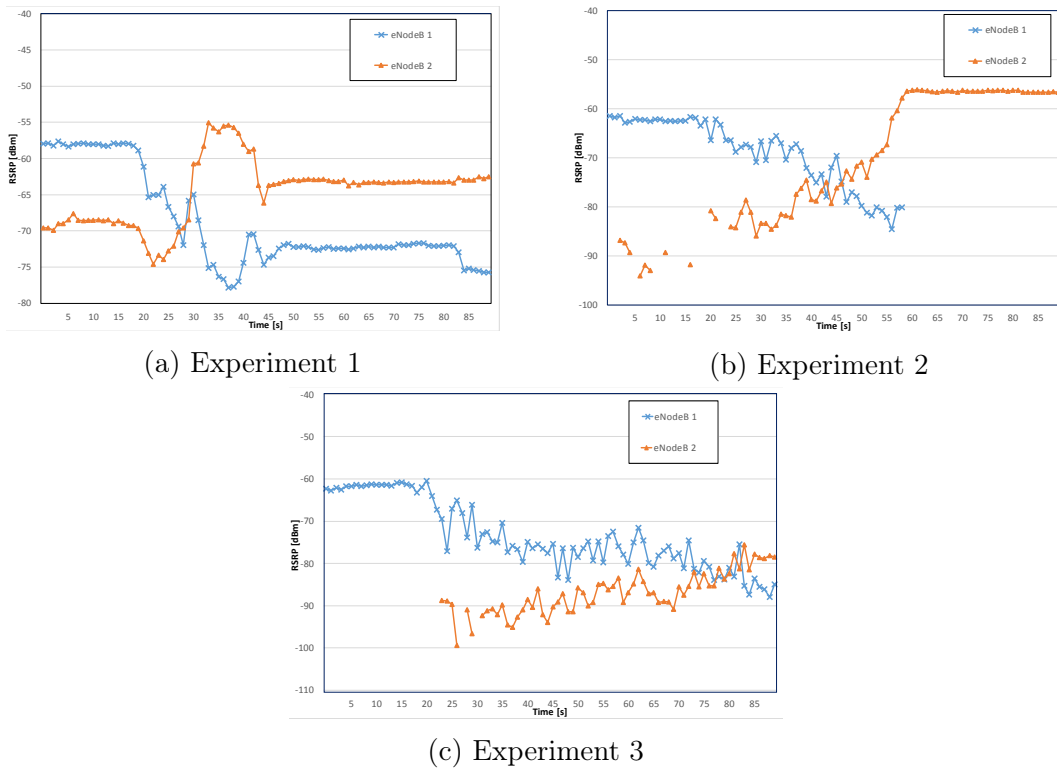


Figure 3.10: 5G-Hander. RSRP values throughout the duration of the experiment

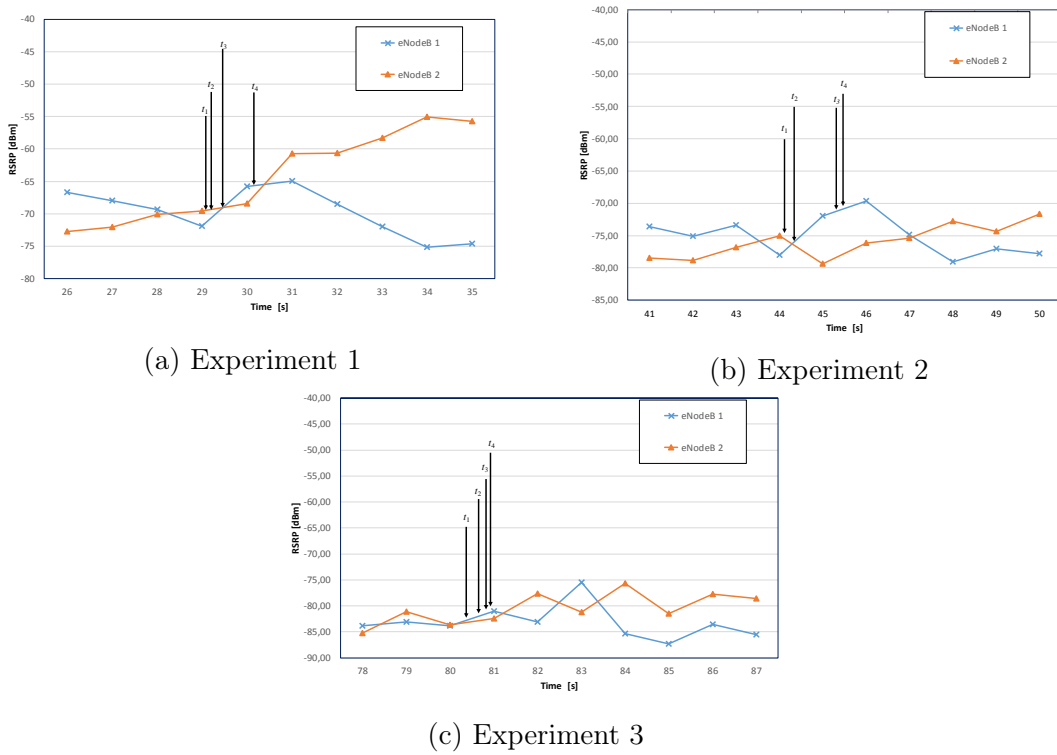


Figure 3.11: 5G-Hander. RSRP values around the handover events



Time	Event
<b><math>t_1</math></b>	<b>HandoverRequest<sub>eNB1→eNB2</sub></b>
$t_{1,1}$	HandoverRequestInformation <sub>HandoverSniffer→Broker</sub>
$t_{1,2}$	HandoverRequestInformation <sub>Broker→NetworkOrchestrator</sub>
<b><math>t_2</math></b>	<b>StatusTransfer<sub>eNB1→eNB2</sub></b>
$t_{2,1}$	StatusTransferInformation <sub>HandoverSniffer→Broker</sub>
$t_{2,2}$	StatusTransferInformation <sub>Broker→NetworkOrchestrator</sub>
<b><math>t_3</math></b>	<b>PathSwitchRequest<sub>eNB2→MME</sub></b>
$t_{3,1}$	PathSwitchRequestInformation <sub>HandoverSniffer→Broker</sub>
$t_{3,2}$	PathSwitchRequestInformation <sub>Broker→NetworkOrchestrator</sub>
<b><math>t_4</math></b>	<b>UEContextRelease<sub>eNB2→eNB1</sub></b>
$t_{4,1}$	UEContextReleaseInformation <sub>HandoverSniffer→Broker</sub>
$t_{4,2}$	UEContextReleaseInformation <sub>Broker→NetworkOrchestrator</sub>

Table 3.1: 5G-Hander. Representative Time Instants and Events during Handover Detection

exchanged between the main components of the proposed testbed are reported. The bold messages are the same shown in Fig. 3.11a, 3.11b and 3.11c. Among the instants of time  $t_1$ ,  $t_2$ ,  $t_3$  and  $t_4$ , the Handover Sniffer exchanges captured information with the Broker which, subsequently, sends them to the Network Orchestrator.

It is important to stress that the proposed network service is able to provide the Orchestrator with information not only useful to timely allocate/deallocate resources, but also to intervene for example to avoid handover events only by blocking some signaling messages among eNBs through an opportune dynamic configuration of the SDN switches in the radio access network.

## 3.2 UAV for Slice extension

This section describes the idea of using UAVs to extend a 5G network slice characterized by low-latency constraint. The capacity of a network to guarantee very severe application's requirements, like ultra low-latency, does not only depend on the length of the physical path that information should follow flowing from sensors to actuators, but also on the size of each information.

For contexts in which a frequent or continuous check of an interest area is required, small-scale UAVs can be used, thanks to their rapid deployment time and their capability to perform low-altitude flights. For persistent monitoring tasks when limited duration of batteries could limit the mission lifetime, use of multiple small-scaled UAVs organized in flocks has been considered in [49].

The problem has been tackled from different perspectives. In particular, after a study of the possible architecture that best suits the proposed idea (Subsection 3.2.1), the first step was the implementation of an analytical model of the single UAV (Subsection 3.2.2) and then a Matlab simulator was realized to analyze the performance considering a fleet of UAVs (Subsection 3.2.3). After that, Reinforcement Learning (RL) was introduced to help the UAVs to take decisions with the aim of managing the resources in an efficient way. For this reason, in Subsection 3.2.4 two cases are proposed: the first case concerns the use of the RL technique applied to a single UAV while, in the second case, RL is applied to neighboring UAVs.

### 3.2.1 UAV for Monitoring System applications

In the papers [50], presented at the IEEE Conference on Network Softwarization in 2018, and [51], published on MDPI Special Issue on Softwarization at the Network Edge for the Tactile Internet, the use of UAVs in a video surveillance system scenario was addressed. In the last decade, thanks to a diffusion of cheap small UAVs and a decrease in price of video camera devices, video monitoring has become very popular especially in the context of surveillance. Since information received from a surveillance UAV may include high-volume sensor data such as live video, and since processing of these data is computationally expensive to be performed locally by the same small-scaled UAVs, they should continuously offload the information to an external data center that can process them. This means that high uninterrupted communications bandwidth should be required, but the maximum communication range is typically limited, especially in the case of smaller and lower-cost UAVs, so they tend to require line-of-sight (LOS) communications with a fixed network access point. The problem of achieving LOS, especially in mountainous or urban areas, cannot be alleviated by increasing altitude when small UAVs are used, because they are unable to ascend to sufficient altitude to achieve LOS to both the target and the access point. This is why the architecture shown in Fig. 3.12 was proposed in the context of network slice extension.

Considering a large area that has to be monitored, it is possible to subdivide it in zones, each monitored by a given set of Monitoring UAVs equipped with cameras and sensors installed on ground. In the same zone a number of actuators are installed to implement actions triggered by some alarm generated from the analysis of data received by the monitoring devices. The set of data produced by one or more monitoring devices that have to be processed together will be

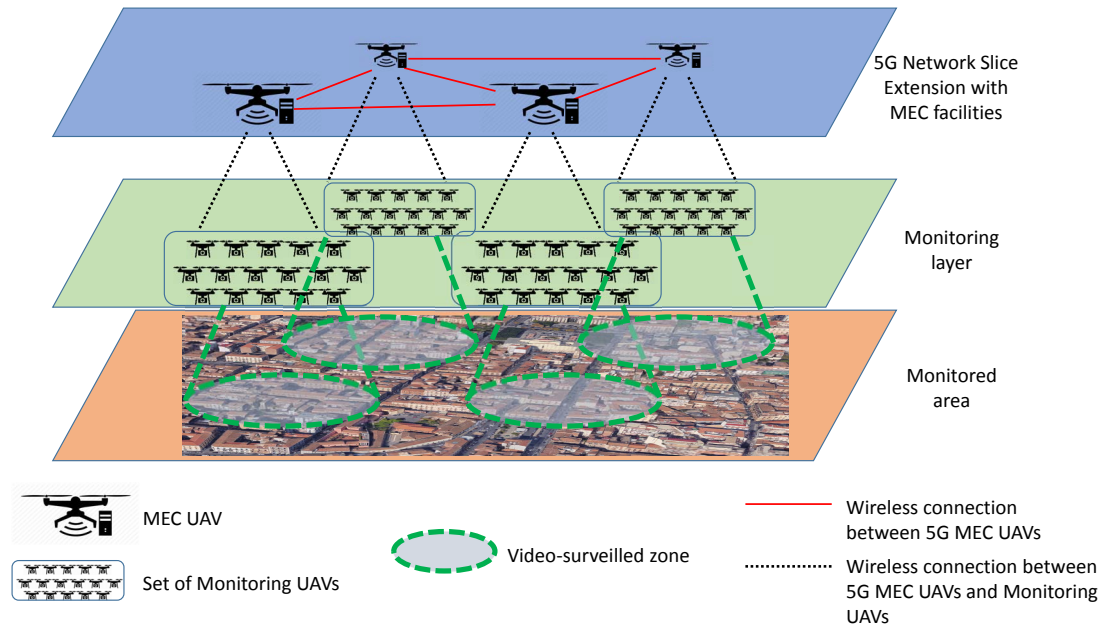


Figure 3.12: UAV for Video Surveillance. Proposed Architecture

referred in the sequel as job. The Monitoring UAVs provide very high quality video monitoring, thanks to a camera installed on board of each drone that captures thousands of images per second with 4K definition, like for example the Phantom’s Flex4K camera. It is clear that this type of data cannot be sent to the cloud, but not even to MEC servers installed on the ground, because this requires a lot of transmission time, aspect incompatible with the requirements of low-latency applications. In fact, assuming an average image size of 10 Mbytes, and an LTE connection of 50 Mbit/s, a transmission time of  $T_{D \rightarrow C_i} = 1.6s$  should be necessary to transmit each image from the drone to the cloud, on average.

For this reason, the 5G network slice is extended with a Flying Ad-hoc NETWORK (FANET) layer constituted by UAVs with MEC facilities (in the following referred to as MEC UAVs), flying very close to the layer of UAVs monitoring the area of interest, with the aim of extending the 5G network edge. The set of Monitoring UAVs covering a given zone is assigned to one MEC UAV, defined as the *primary drone* for that zone. The number of Monitoring UAVs assigned to one MEC UAV depends on the load generated by each Monitoring UAV and the load that each MEC UAV is able to manage with acceptable performance.

The considered scenario is the same shown in Fig. 4.1: the Controlled Domain is constituted by the monitored geographic area in which sensors and actuators are installed and by the cameras on board of Monitoring UAVs, while in the Master Domain there is the human operator aims at accessing the video-surveillance system through a graphic user interface (GUI).

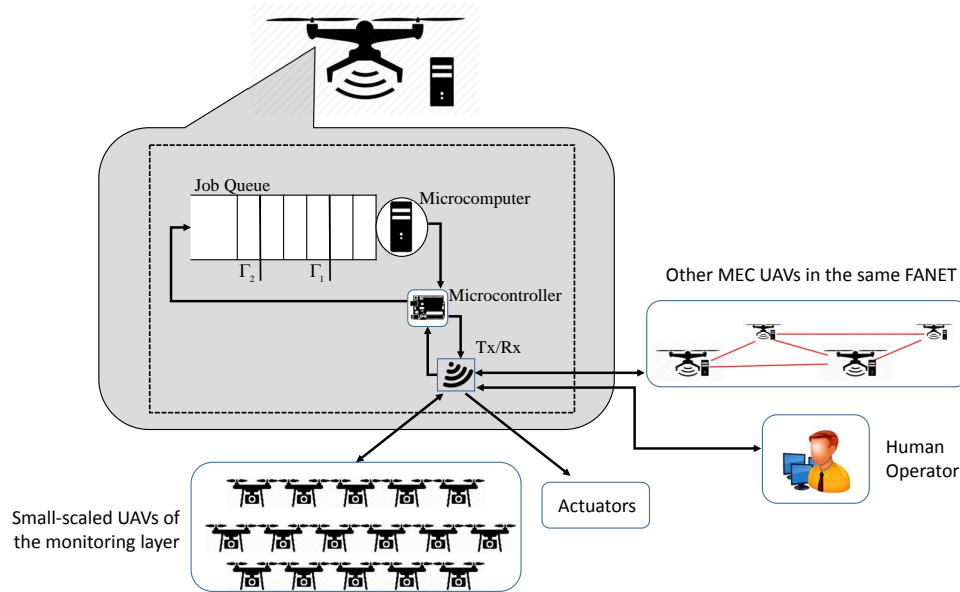


Figure 3.13: Drone Functional Architecture

The functional architecture of each drone is sketched in Fig. 3.13. It is constituted by:

- a Transmit/Receive (Tx/Rx) block equipped with multiple antennas for sending and receiving more than one data signal simultaneously over the same radio channel by exploiting Multiple-Input and Multiple-Output (MIMO) technologies;
- a Microcontroller board to control processing of the monitoring information coming from the Monitoring UAVs, and decide when sending trigger commands to the actuators in case of alarm event detection; a reduced flow of information is sent from the microcontroller to the remote operator in the Master Domain to inform him of the most relevant captured data and allow him to take decisions. In the case that decisions do not arrive from the operator in time, the Microcontroller is in charge of trigger actions on the actuators on the basis of the decisions suggested by the microcomputer;
- a Microcomputer to process jobs offloaded by the microcontroller.

The UAVs at the monitoring layer periodically capture images and send them to the MEC UAV assigned to them. The image capturing frequency changes according to feedback messages that Monitoring UAVs receive either from sensors installed on ground (e.g. motion detection) or the MEC UAV upon processing previous data. More specifically, different alarm levels can be set on the Microcontroller. Transitions between adjacent alarm levels are determined by the

information coming from the Monitoring UAVs and then processed: the higher the alarm level, the higher the image capturing frequency.

Through the Tx/Rx block, the MEC UAV receives information from Monitoring UAV and send them locally to the Microcontroller, that also could receive images captured by ground cameras.

The Microcontroller combines images and data received as jobs that are enqueued in a first-in-first-out (FIFO) job queue, waiting to be processed by the local Microcomputer which, as soon as possible, has to decide if changing the level of alarm in order to send an “action” message to the actuators installed on the ground. Given the small size of messages received by the sensors and transmitted to the actuators, and the small distance between sensors/actuators and the drone, data transmission time from sensors to the drone and from the drone to the actuators can be considered negligible.

The computing power of the Microcomputer has to be timely designed in such a way that the Tactile Internet requirement on the e2e maximum delay of 1 ms is not violated, considering that it is defined as:

$$T_{e2e} = T_{S \rightarrow D} + T_Q + T_P + T_{D \rightarrow A} \leq 1ms \quad (3.1)$$

where  $T_{S \rightarrow D}$  and  $T_{D \rightarrow A}$  are the transmission times from the sensors to the drone and from the drone to the actuators, which are considered negligible, while  $T_Q$  and  $T_P$  are the times spent in the job queue and in the queue processing unit (the microcontroller), respectively.

In the job queue two thresholds were introduced in order to reduce the queueing time,  $T_Q$ , and the job loss. Let  $\Gamma_1$  and  $\Gamma_2$  be these thresholds, with  $\Gamma_1 \leq \Gamma_2 \leq K - 1$ , where  $K - 1$  is the maximum size of the job queue. When the number of jobs in the queue exceeds  $\Gamma_2$ , local Microcomputer is not able to sustain the current job load, so the support of an additional drone, assuming the role of *secondary drone*, is required. The phase of the secondary drone selection starts with a broadcast help request issued by the primary drone to the drones in proximity. These last ones answer by sending their status to the primary drone, which will assign the role of secondary drone on the basis of the received information. The drone that is chosen as secondary drone is the one with the lowest queue among the ones that are in proximity (i.e. with a distance not greater than  $\delta$ ), which are not yet secondary drones, and which are not helped by any other drone to monitor the zone where they are the primary drone. The choice of the distance  $\delta$  is important in this step: too-low values of  $\delta$  reduce the number of candidates as secondary drones, while with a too-high value of  $\delta$  the secondary drone can be too far to

correctly video-monitor the zone monitored by the first drone. Obviously, if no drones are available to assume the role of secondary drone, then the primary drone will issue a new help requests at each new job arrival or service event, or at a timeout event. If the job queue of the primary drone saturated with or without the help of the secondary drone, some jobs are lost. After the selection of the secondary drone, the primary sends a direct communication to the chosen drone, and notifies it to the Monitoring UAVs in order that the data flow can be split to the two drones currently monitoring its area. A similar signaling information exchange occurs when the second drone has to finish its helping role. When the job queue length decreases to a value less than  $\Gamma_1$ , the secondary drone will not be more needed, and the zone is again monitored by only the primary drone.

Fig. 3.14 shows how the search algorithm for a secondary drone works, carried out by the generic  $i$ -th drone. Two possible events determine a queue length variation: a job arrives to the  $i$ -th drone or a job is served by the Microcomputer. If the queue state of this drone,  $q$ , is greater than  $\Gamma_2$  and the drone is not still helped, the *need\_for\_help* variable is set to 1 and a new *Search thread* is instantiated. As soon as the `<execute_s()>` method is invoked in the *Search thread*, it enters the `Running_Search` state and starts to execute the code of the Algorithm 1. At the same time, another thread, the *Timeout thread*, is also instantiated with the aim of repeating this procedure periodically until a secondary drone is not found. This last thread enters the `Suspended` state, waiting for receiving a `<resume_request_t()>` message.

If secondary drone search procedure, at its end, was able to find a secondary drone, the variable *need\_for\_help* is set to 0, the Id of the UAV chosen as secondary drone is sent to the Microcontroller, and the *Search thread* receives the `<stop_s()>` command to die. On the contrary, if no secondary drone has been found, the `<wait_s()>` method is invoked, the *Search thread* enters the `Suspended` state, and a `<resume_request_t()>` message is sent to the *Timeout thread*. This one leaves the `Suspended` state, invokes the `<resume_t()>` method and enters the `Running_Timeout` state where a countdown begins by decreasing the timeout variable. At a time when timeout reaches the value 0, if *need\_for\_help* is still equal to 1, the `<resume_request_s()>` method allows the *Search thread* to exit from the `Suspended` state and, after invoking `<resume_s()>`, returns to the `Running_Search()` state. Instead, if *need\_for\_help* is equal to 0, because in the meanwhile the queue of the drone has reached a value below the  $\Gamma_2$  threshold, the *Timeout thread* enters the `Suspended` state and waits for next actions.

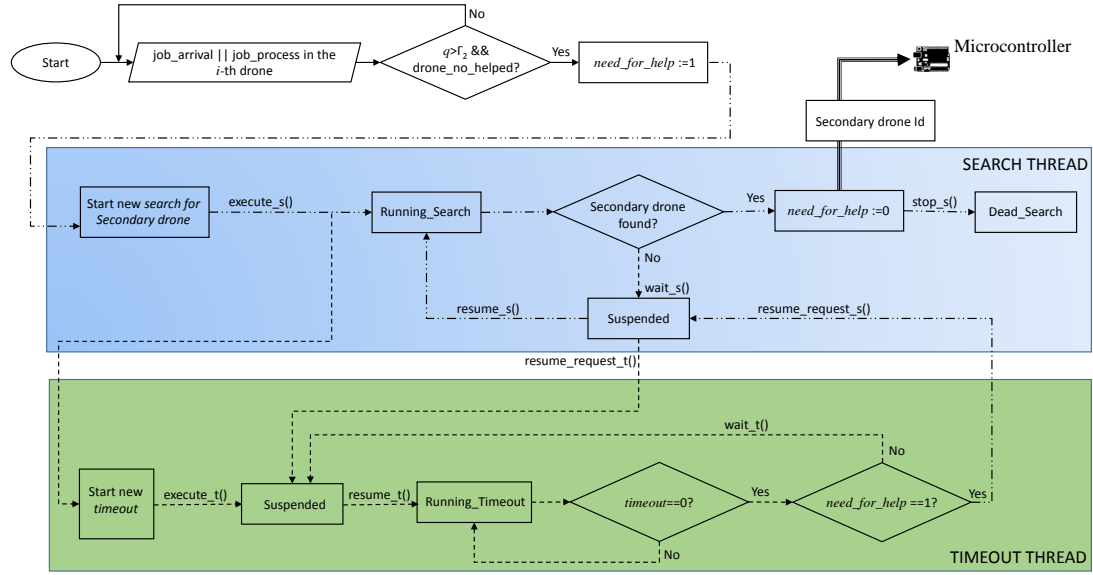


Figure 3.14: Mutual Help Policy implementation algorithm

---

**Algorithm 1** Running Search execution code

---

```

1: begin
2:   Send a help_request message in broadcast to UAVs within a distance  $\delta$ 
3:   Receive queue-state from all the UAVs in proximity that are neither help-
   ing other UAVs nor helped
4:   if UAVs available  $\geq 1$  then
5:     Secondary drone Id := drone Id of the drone with the smallest queue
6:   else
7:     Secondary drone Id := null
8:   end if
9:   return Secondary drone Id
10: end

```

---

### 3.2.2 Analytical System Model and results

A first analysis of the described system was conducted in [50], through the implementation of an analytical model, focusing the attention to one drone as a reference. The mean job arrival rate depends on the alarm level (in the following referred to as alarm state) of the monitored zone. It can be represented by an array  $\underline{\Lambda}$ , whose generic  $z$ -th element,  $\Lambda_{[z]}$ , contains the mean arrival rate when the zone covered by the considered drone is in the alarm state  $Z$ . Let  $Q^{(z)}$  be the state transition probability matrix of the alarm zone state process, which is independent of the behavior of the drone we are modeling.

As regards the job queue serving rate  $\mu P$ , it depends on whether the drone we are modeling here is working only for its zone, or it is helping, as a secondary drone, another drone in proximity. So it is possible to write:

$$\mu P = \begin{cases} \mu & \text{Drone works only for its zone} \\ \mu/2 & \text{Drone works as secondary drone} \end{cases} \quad (3.2)$$

where  $\mu$  indicates the mean job service rate provided by the microcomputer. Since the considered drone can be chosen to help another drone depending on the value  $q$  of its queue, and only if 1) it is not helped by another drone (i.e. the number of drones monitoring its area is  $d = 1$ ), and 2) it is not helping another drone, its transition behavior will be characterized with a transition rate matrix,  $Q^{(\mu P)}(q, d)$ , that depends on both  $q$  and  $d$ . This matrix is assumed as an input of the problem.

The system is modeled as a 4-dimension continuous-time Markov chain defined as follows:

$$S^{(\Sigma)}(t) = (S^{(Z)}(t), S^{(Q)}(t), S^{(D)}(t), S^{(\mu P)}(t)) \quad (3.3)$$

where:

- $S^{(Z)}(t)$  represents the state, i.e. the alarm level, of the area to be monitored. Its value determines the job arrival rate to the job queue. In the sequel, two states are assumed,  $L$  and  $H$ , representing a normal state and a pre-alarm state, respectively. Therefore, we have that  $S^{(Z)} \in \mathfrak{S}^{(Z)} \equiv \{L, H\}$ . Let  $\lambda^{(L)}$  and  $\lambda^{(H)}$  be the job arrival rates associated to these states, with  $\lambda^{(H)} \geq \lambda^{(L)}$ ;
- $S^{(Q)}(t)$  represents the state of the job queueing system, that is, the number of jobs that are present in the queue and in the server (i.e. the microcomputer) at the instant  $t$ . Its state space is  $\mathfrak{S}^{(Q)} \equiv [0, K]$ ;
- $S^{(D)}(t)$  represents the condition whether the drone its monitoring the area by itself, or there is a secondary drone helping it in monitoring the same zone. Therefore, this variable can be modeled with the number of drones that are monitoring the area of the considered drone, that is, its state space is  $\mathfrak{S}^{(D)} \equiv \{1, 2\}$ ;
- $S^{(\mu P)}(t)$  represents the actual service rate of the job queue, depending on whether the considered drone is helping another drone to monitor its area, or not. Therefore, it can be modeled with the amount of processor that the considered drone is dedicating to the own area. Thus, indicating the total job rate that can be served by the microprocessor as  $\mu$ , then the state space of  $S^{(\mu P)}(t)$  is  $\mathfrak{S}^{(\mu P)} \equiv \{\mu/2, \mu\}$ .



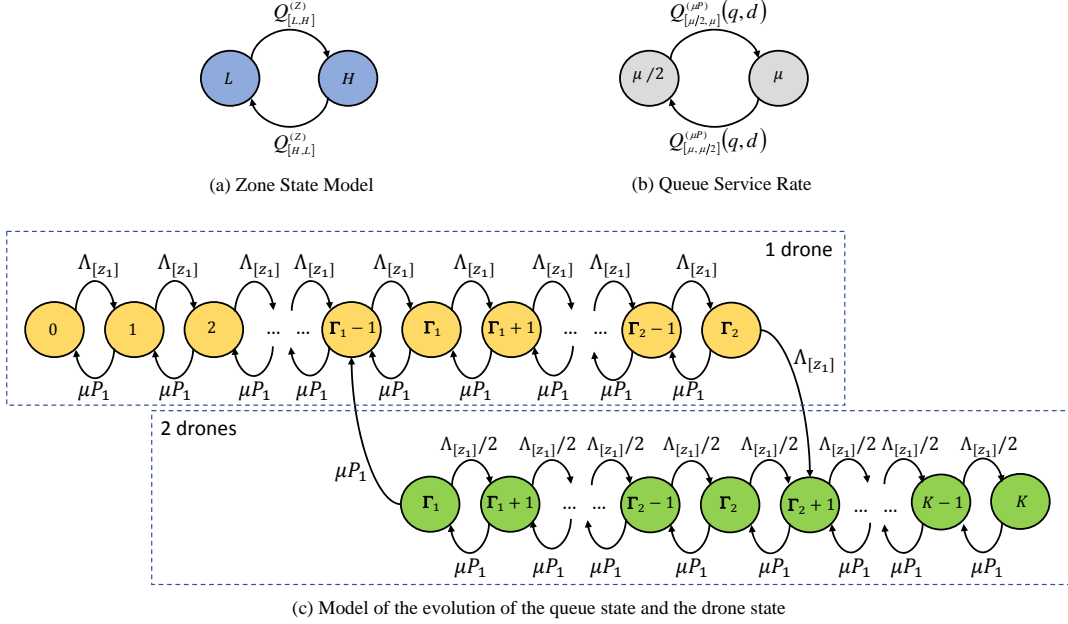


Figure 3.15: UAV for Video Surveillance. System Model diagrams

Indicating the state space of the Markov chain (defined in equation 3.3) as  $\mathfrak{S}^{(\Sigma)}$ , it is possible to consider two generic states  $s_{\Sigma 1} \in \mathfrak{S}^{(\Sigma)}$  and  $s_{\Sigma 2} \in \mathfrak{S}^{(\Sigma)}$ , where:

$$s_{\Sigma 1} = (z_1, q_1, d_1, \mu P_1) \quad (3.4)$$

$$s_{\Sigma 2} = (z_2, q_2, d_2, \mu P_2) \quad (3.5)$$

Fig. 3.15 shows the Markov chain transition diagrams.

In particular, in Fig. 3.15a there is the Markov chain of the zone alarm. its independence of the other chains is evident, it is governed by the matrix  $Q^{(Z)}$  only. On the contrary, the transition rates characterizing the state of the job queue server (Fig. 3.15b), which are the elements of the matrix  $Q^{(\mu P)}(q, d)$ , depend on the states of both the job queue,  $S^{(Q)}(t)$ , and the process  $S^{(D)}(t)$ .

The chain shown in Fig. 3.15c is constituted by two subchains: the upper one representing the job queue states requiring only one drone to monitor the zone, while the lower one containing the states where the primary drone is helped by a secondary drone. The transition between these two sub-chains is shown in the figure: when the cumulative number of jobs in the queue and in the server exceeds the threshold  $\Gamma_2$ , the state moves on the lowest sub-chain, where the job arrival rate is halved because another drone is involved as secondary drone to monitor the same zone and process the captured images. Only when the state decreases under the threshold  $\Gamma_1$ , monitoring is carried out by one drone only again. Assuming, as usual, that two state-change events occur at the same instant with

zero probability, we can write the generic element of the global state transition rate matrix as follows:

$$Q_{[s_{\Sigma 1}, s_{\Sigma 2}]}^{(\Sigma)} = \begin{cases} \Lambda_{[z_1]} & \text{if } C_1^{(Q,D)} \\ \Lambda_{[z_1]}/2 & \text{if } C_2^{(Q,D)} \\ \mu P_1 & \text{if } C_3^{(Q,D)} \\ Q_{[\mu P_1, \mu P_2]}^{(\mu P)}(q_1, d_1) & \text{if } \mu P_2 \neq \mu P_1 \\ Q_{[z_1, z_2]}^{(Z)} & \text{if } z_2 \neq z_1 \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

where:

- $C_1^{(Q,D)}$  is a Boolean variable defined as:

$$C_1^{(Q,D)} = \{0 \leq q_1 < \Gamma_2 \text{ and } q_2 = q_1 + 1 \text{ and } d_1 = 1\} \quad (3.7)$$

This means that the queue state increases by 1 job, and the transition starts from a state with only 1 active drone. In this case, the job arrival rate to the queue coincides with the whole rate of arrivals needed to monitor the considered zone, i.e.  $\Lambda_{[z_1]}$ ;

- $C_2^{(Q,D)}$  is a Boolean variable defined as:

$$C_2^{(Q,D)} = \{\Gamma_1 \leq q_1 < K \text{ and } q_2 = q_1 + 1 \text{ and } d_1 = 2\} \quad (3.8)$$

This means that the queue state increases by 1 job, but now the transition starts from a state with 2 active drones. In this case, the job arrival rate to the queue of the first drone is half of the previous case;

- $C_3^{(Q,D)}$  is a Boolean variable defined as:

$$C_3^{(Q,D)} = \{q_2 = q_1 - 1\} \quad (3.9)$$

and represents the case in which the queue state decreases by 1 job.

At this point, it is possible to calculate the steady-state probability array, whose generic element is defined as:

$$\pi_{[z, q, d, \mu P]}^{(\Sigma)} = \lim_{t \rightarrow \infty} Pr\{S^{(Z)}(t) = z, S^{(Q)}(t) = q, S^{(D)}(t) = d, S^{(\mu P)}(t) = \mu P\} \quad (3.10)$$

and can be calculated by solving the following linear equation system, in which

$\underline{\pi}^{(\Sigma)}$  is a row array containing the steady-state probabilities of all the state in  $\mathfrak{S}^{(\Sigma)}$ :

$$\begin{cases} \underline{\pi}^{(\Sigma)} \cdot Q^{(\Sigma)} = \underline{0}^T \\ \underline{\pi}^{(\Sigma)} \cdot \underline{1}^T = 1 \end{cases} \quad (3.11)$$

The second equation of (3.11) imposes that the sum of all the elements of array  $\underline{\pi}^{(\Sigma)}$  is equal to 1. Solving the above system, the marginal steady-state probability array of the job queue can be derived:

$$\pi_{[q]}^{(Q)} \equiv \lim_{t \rightarrow \infty} Pr\{S^{(Q)}(t) = q\} = \sum_z \sum_d \sum_{\mu P} \pi_{[z,q,d,\mu P]^{(\Sigma)}} \quad (3.12)$$

The mean number of jobs in the job queueing system (either in the queue or in service) can be calculated as follow:

$$\bar{N} = \sum_q q \cdot \pi_{[q]}^{(Q)} \quad (3.13)$$

Next, the mean arrival rate of jobs to the queue,  $\bar{\lambda}$ , is calculated as:

$$\bar{\lambda} = \sum_z \sum_q \sum_{\mu P} \left\{ \Lambda_{[z]} \cdot \pi_{[z,q,1,\mu P]^{(\Sigma)}} + \frac{1}{2} \Lambda_{[z]} \cdot \pi_{[z,q,2,\mu P]^{(\Sigma)}} \right\} \quad (3.14)$$

so the mean service time, defined as the mean time spent in the job queueing system, can be calculated by the Little theorem as follows:

$$\bar{T}^{(\Sigma)} = \frac{\bar{N}}{\bar{\lambda}} \quad (3.15)$$

About the job loss probability due to queue overflow, it is easy to demonstrate that it coincides with the probability that the queueing system is full, that is:

$$P_{Loss}^{(Job)} = \pi_{[K]}^{(Q)} \quad (3.16)$$

Finally, the probability that the system delay is greater than 1 ms (due to the Tactile Internet requirements) is calculated as follows:

$$P_{TI} = Pr\{\text{System Delay} \geq 1\text{ms}\} = \sum_{q=q_{TI}}^K \pi_{[q]}^{(Q)} \quad (3.17)$$

with  $q_{TI}$  the minimum queue value giving a queueing system delay greater than

1 ms. It is calculates as:

$$q_{TI} = \lceil 1\text{ms} \cdot \overline{\mu P} \rceil \quad (3.18)$$

where  $\overline{\mu P}$  is the mean service rate, given by:

$$\overline{\mu P} = \sum_z \sum_q \sum_d \sum_{\mu P} \mu P \cdot \pi_{[z,q,d,\mu P]}^{(\Sigma)} \quad (3.19)$$

To evaluate the performance of this model, some experiments have been executed. To this purpose, a drone with a job queue capacity of  $K = 300$  jobs and the two thresholds set as  $\Gamma_1 = K/3$  and  $\Gamma_2 = 2\Gamma_1$  were considered. The job arrival rate during the two zone alarm levels (i.e. “L” and “H”) is represented by the array  $\underline{\Lambda} = [10, 17h]$ , where  $h$  is a parameter that will be varied in the range  $[1.0, 2.0]$  in the numerical analysis. This allows to consider different frame rates during the “H” alarm state. The zone state will be characterized by the following transition rate matrix:

$$Q^{(Z)} = \begin{bmatrix} 0.997 & 0.003 \\ 0.090 & 0.910 \end{bmatrix} \quad (3.20)$$

Results in Fig. 3.16 have been calculated by varying the job queue service rate,  $\mu$ , in the range  $[6, 40]$  job/ms, in order to evaluate the impact of the processor capacity on the system performance, and considering five cases have been considered, one for each value of image capture rate during the pre-alarm phase “H”.

Fig. 3.16a and 3.16b show the mean delay and the mean number of jobs in the job queue. These two parameters decrease with the processor service rate and are almost independent of  $\lambda_H$ . The job loss probability is depicted in Fig. 3.16c and, as expected, for the lowest values of job queue service rate, there are important losses. However, the main problem of Tactile Internet applications is the delays. For this reason, in Fig. 3.16d the probability that the delay exceeds 1 ms, calculated as in equation (3.17), is plotted. This graph helps to choose the microcomputer that gives acceptable performance, keeping this probability less than a given target value. For example, if the application requires a probability of violating the 1 ms requirement not greater than  $1 \cdot 10^{-7}$ , with a  $\lambda_H = 29.75$  job/ms, a processor that is able to serve jobs with a rate of at least 33.2 job/ms is necessary.

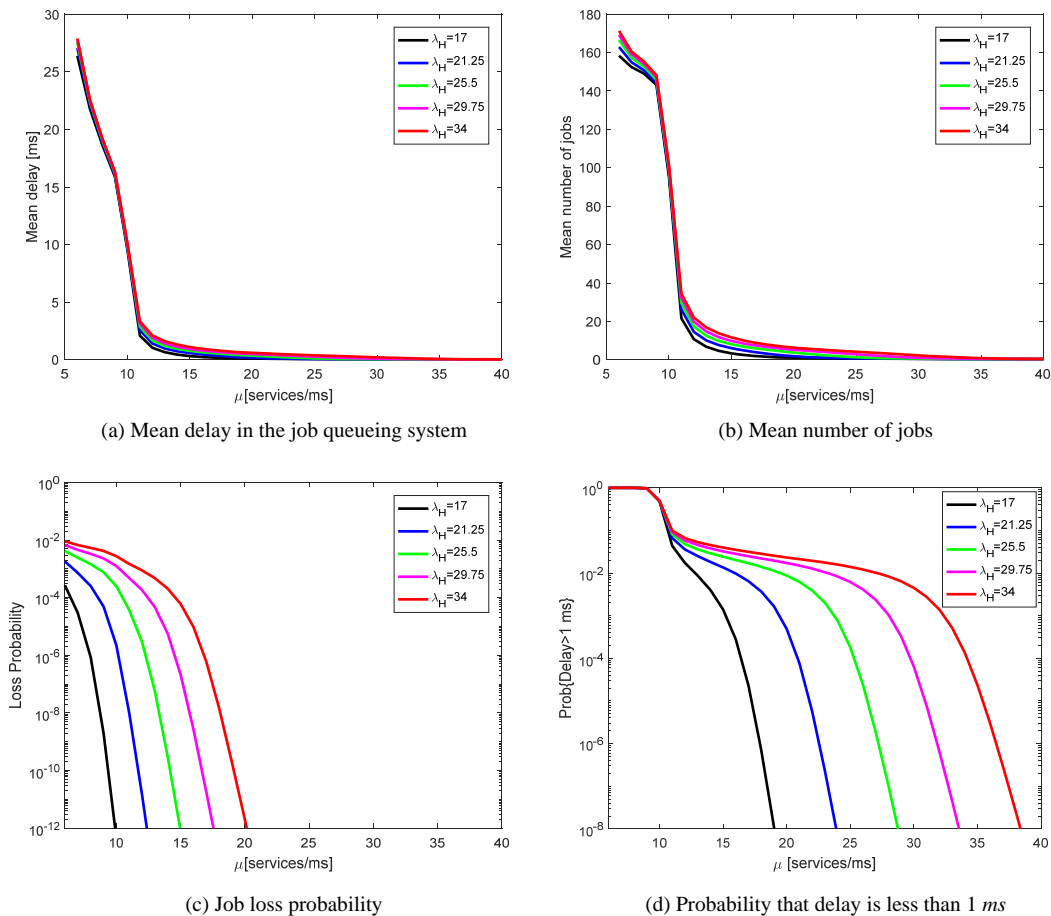


Figure 3.16: UAV for Video Surveillance. Numerical Results

### 3.2.3 Event-based Matlab simulation and results

The second step of this work was to implement an event-based Matlab simulator. This allows to analyze the completed system with a fleet of simulated UAVs and evaluate the performance of the proposed solution.

A 5G slice extension with  $N = 64$  MEC UAVs was considered, organized in a FANET with a regular grid topology, as shown in Fig. 3.17. In order to evaluate the gain achieved by using the proposed Mutual Help Policy among MEC UAVs, two different configurations were analyzed:

- “w/ help”: each UAV has a horizontal coverage range of  $\delta = 1$  km, and therefore it is able to cooperate with all the other UAVs in the platform;
- “w/o help”: each UAV has a horizontal coverage range of  $\delta = 70$  m, and therefore each zone is covered by the primary drone only. This case is equivalent to the state-of-art case, in which the strategy of “mutual help among MEC UAVs” is not applied.

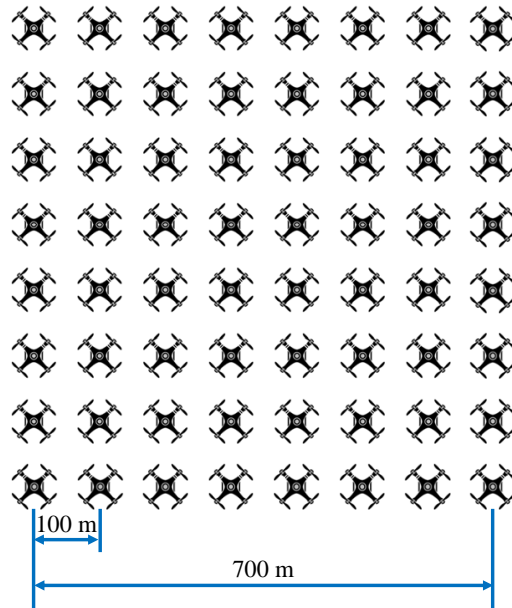


Figure 3.17: FANET topology used in the use case

Two levels of alarm for each monitored zone were assumed, and an exponentially distributed job arrival rate with a mean value depending on the current level. More specifically, there is a mean arrival rate during no-alarm state of  $\lambda_L = 1$  arrival/ms, while the arrival rate during an alarm state is increased with a factor of 15, that is,  $\lambda_H = 15$  arrivals/ms. As concerns the Microprocessor unit on board of each MEC UAV, in this simulation it is able to process one job in  $T_J = 769\mu s$ , so it has a job processing rate of  $\mu = 13$  jobs/ms. Moreover, the alarm-level transitions is based on a 2-state Markov chain characterized by the following transition rate matrix:

$$Q^{(A)} = \begin{bmatrix} -1.67 \cdot 10^{-2} & 1.67 \cdot 10^{-2} \\ 0.5 & -0.5 \end{bmatrix} \quad (3.21)$$

In the numerical analysis the job queue size,  $K$ , varies in the range  $[100, 1000]$ . The two thresholds  $\Gamma_1$  and  $\Gamma_2$ , after a huge number of simulations, have been decided as follows:

$$\Gamma_2 = 2K/3 \quad \text{and} \quad \Gamma_1 = \Gamma_2 - 10 \quad (3.22)$$

Every set of experiments was repeated 30 times with different random seeds, and the duration of simulation has been automatically chosen in such a way that the 95% confidence interval is less than 1% of the average results. The seeds,

which affected all the random parameters in the experiments, were taken from the default pseudo-random generator of Matlab.

The first two considered metrics, regarding application level QoS, are the *mean response time* and the *job loss rate*. The mean response time is defined as the mean value of the duration of the permanence for a job in the system and, due to the fact that the transmission time between sensors and Monitoring UAV, Monitoring UAV and MEC UAV and MEC UAV and actuators are negligible, as already said it can be defined as:

$$\bar{T}_{e2e} = \bar{T}_Q + \bar{T}_{\mu P} \quad (3.23)$$

where  $\bar{T}_Q$  is the mean time of permanence in the queue and can be derived through the Little law as an average of the ratio between the expected value of the number of jobs in the queue of each drone  $u$ ,  $Q_u$ , and the expected value of the arrival rate to that drone,  $\Lambda_u$ , that is:

$$\bar{T}_Q = \frac{1}{N} \sum_{u=1}^N \frac{E\{Q_u\}}{E\{\Lambda_u\}} \quad (3.24)$$

$\bar{T}_{\mu P}$  is the mean value of the time needed by a job to be processed in the Microcomputer, and is an input of the problem.

If we indicate how much time the job queue of the UAV  $u$  stays with a job queue length of  $q$  jobs in a period of duration  $t$  as  $\tau_{u,q}(t)$ , and the cumulative number of arrivals in the same interval as  $A_u(t)$ , from Equation (3.24) it is possible to write:

$$\bar{T}_Q = \frac{1}{N} \sum_{u=1}^N \lim_{t \rightarrow \infty} \sum_{q=0}^K q \cdot \frac{\tau_{u,q}(t)}{A_u(t)} \quad (3.25)$$

The job loss rate, as the previous metric, is averaged over all the UAVs and, for each UAV  $u$ , it is calculated as the ratio between the number of jobs that are lost for queue overflow in a period with a duration of  $t$ ,  $J_u^{(L)}(t)$ , and the total number of jobs created by the monitoring UAVs assigned to  $u$  in the same period,  $J_u^{(A)}(t)$ . This means that:

$$P_{JL} = \lim_{t \rightarrow \infty} \frac{\sum_{u=1}^N J_u^{(L)}(t)}{\sum_{u=1}^N J_u^{(A)}(t)} \quad (3.26)$$

Fig. 3.18 plot these two application-level QoS parameters against the job queue size  $K$ . In both Figs. 3.18a and 3.18b, it is shown the gain achieved by applying the Mutual Help Policy among MEC UAVs. For example, it is easy to notice that, for job buffer size of 300, the job loss probability is around  $1 \cdot 10^{-5}$  if the Mutual

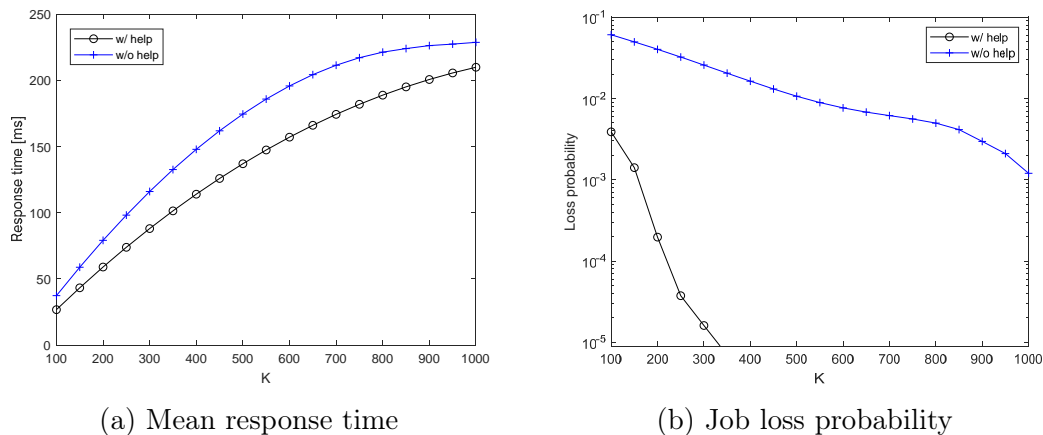


Figure 3.18: UAV for Video Surveillance. Application-level QoS parameters

Help Policy is applied, with a mean response time of 88 ms. If a higher response time is accepted, job losses can be further reduced by increasing the buffer size.

Another important analysis regarding the mean response time and the job loss probability is the impact of the arrival rate parameter to the overall system performance. Considering the arrival rate, during an alarm state, in the interval [6, 24] arrival/ms and the arrival rate during a no-alarm state constant and equal to 1 arrival/ms, Fig. 3.19 shows the behavior of two parameters.

It can be noticed that, although a gap between the “w/ help” and the “w/o help” remains in the job loss probability (Fig. 3.19b), the mean response time shown in Fig. 3.19a is improved by the Mutual Help Policy for intermediate values of the arrival rate since. In fact, for low values of  $\lambda_H$ , the platform is underloaded, and therefore MEC UAVs do not need any help, while, for high values of  $\lambda_H$ , the platform is so overloaded that it is unlikely to find MEC UAVs available for helping.

The third parameter is the *no-helped probability*, it is averaged over all the UAVs and, for each of them, it is defined as the ratio between the cumulative time duration  $T_u^{(NH|\geq\Gamma_2)}$  calculated until the time instance  $t$  of the period in which the UAV  $u$ , although with a job queue length higher than  $\Gamma_2$ , is not helped by a secondary drone because not available, and the overall time duration,  $T_u^{(\geq\Gamma_2)}(t)$ , of the period, in the same interval, that the considered UAV has had a job queue length higher than  $\Gamma_2$ :

$$P_{\geq\Gamma_2}^{(NH)} = \frac{1}{N} \sum_{u=1}^N \lim_{t \rightarrow \infty} \frac{T_u^{(NH|\geq\Gamma_2)}(t)}{T_u^{(\geq\Gamma_2)}(t)} \quad (3.27)$$

Another parameter is the *mean duration of no-helped periods*, defined as:



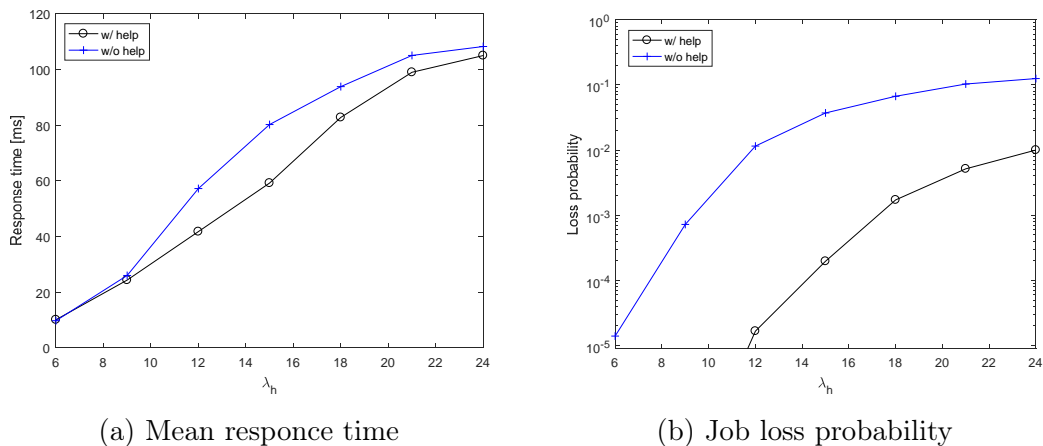


Figure 3.19: UAV for Video Surveillance. Dependence of application-level QoS parameters on the job arrival rate during alarm states

$$\bar{T}^{(NH|\geq\Gamma_2)} = \frac{1}{N} \sum_{u=1}^N \left[ \lim_{t \rightarrow \infty} \frac{T_u^{(NH|\geq\Gamma_2)}(t)}{C_{NH,u}(t)} \right] \quad (3.28)$$

where  $C_{NH,u}(t)$  is the number of no-helped occurrences when the UAV  $u$  needs to be helped in  $t$  seconds. Finally, the last parameter represents the *percentage of time devoted to help another UAV*. It can be calculated as the cumulative time spent by each UAV in helping some other UAV in a period of duration  $t$ ,  $T_u^{(Helping)}$ , normalized over the time duration  $t$ , and averaged for all the UAVs in the system:

$$R^{(H)} = \frac{1}{N} \sum_{u=1}^N \left[ \lim_{t \rightarrow \infty} \frac{T_u^{(Helping)}(t)}{t} \right] \times 100\% \quad (3.29)$$

Fig. 5.36 presents the above three performance parameters. Of course, the curves relating to the “w/o help case” in Figs. 3.20a and 3.20c are trivial, but have been reported here for the sake of completeness.

For higher values of  $K$ , and consequently of the two thresholds, the queue length rarely exceeds  $\Gamma_2$ , and therefore the gain of the Mutual Help Policy is low again. This is the reason why in Figs. 3.20a and 3.20b the curves for the “w/ help” case are not present for high values of  $K$ . Instead, for low values of  $K$ , the curves are present and decrease because situations characterized by queue lengths higher than  $\Gamma_2$  are less frequent for  $K$  increasing, and so the secondary drone search is less invoked, and therefore more UAVs are available to help the few ones that request some help. The curve of the “w/o help” case in Fig. 3.20b shown two different behaviors of the system: it increases up to  $K = 650$ , because, as said, for low values of  $K$  and  $K$  increasing, the loss probability decreases and the whole

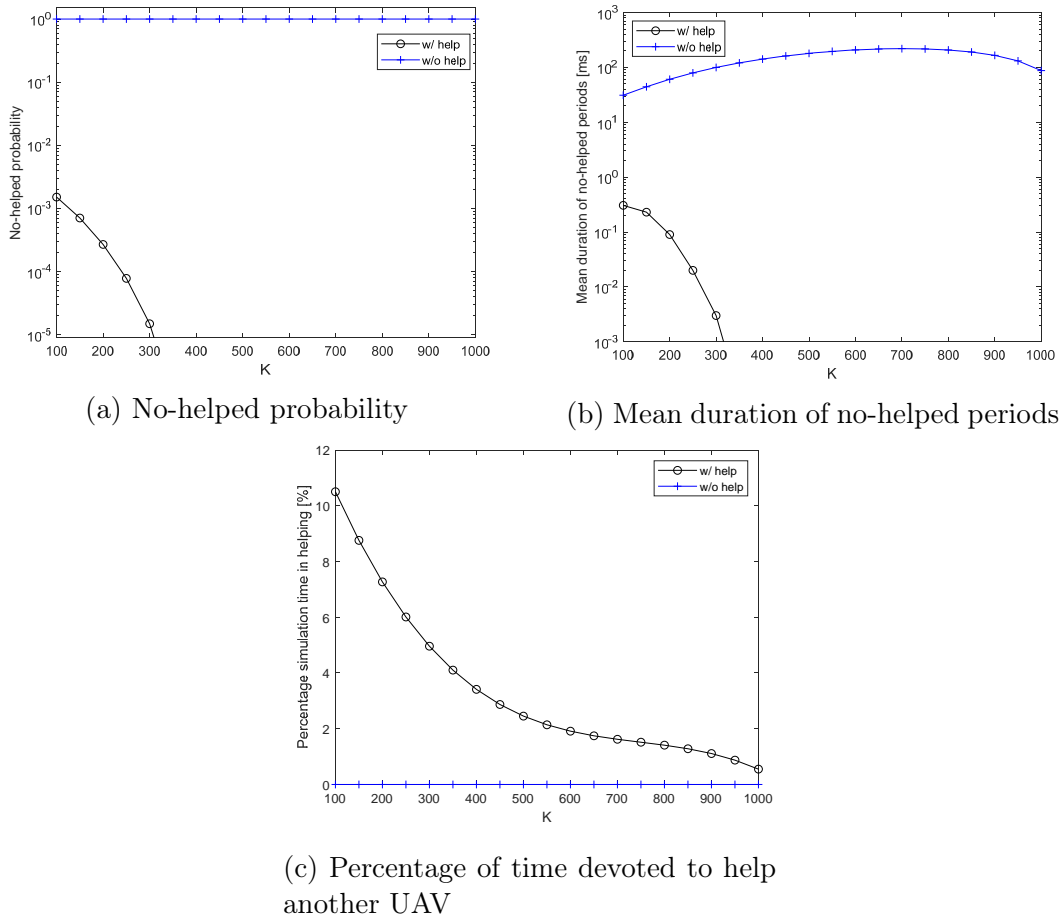


Figure 3.20: UAV for Video Surveillance. Performance parameters for a behaviour analysis of the proposed framework

number of packets managed by the MEC platform at the same time increases, so increasing the time to find a secondary UAV. Instead, the right part of the curve is decreasing because the probability that the queue length is greater than  $\Gamma_2$  decreases, and therefore the system appears less loaded. This explains why the curve in Fig. 3.20c for the “w/ help” case is decreasing in the whole range of  $K$ .

To better analyze the impact of the Mutual Help Policy, in Fig. 3.21 the normalized histograms of the job queue length for  $K=200$  and  $K=900$ , measured for one of the drones in the FANET, and for both the configurations “w/o help” and “w/ help”, are shown. It is clear that if the Mutual Help Policy is not applied, the queue length spans in the whole range between 0 and  $K$ , also causing some losses for  $K = 200$ . On the contrary, application of the Mutual Help Policy allows to maintain the queue rarely over 160 jobs for  $K = 200$  and 600 jobs for  $K = 900$ , so strongly reducing losses.

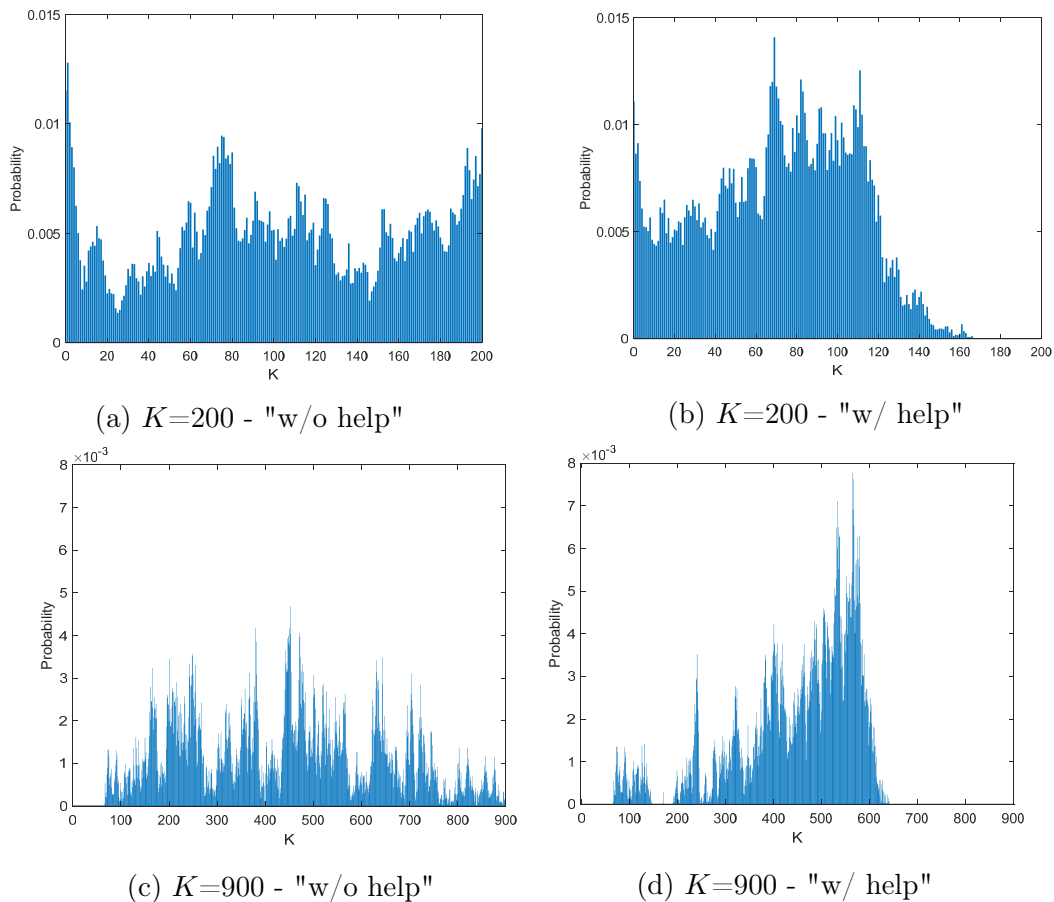


Figure 3.21: UAV for Video Surveillance. Normalized histograms of the job queue length

### 3.2.4 MEC UAV managed with Reinforcement Learning

Another step in the application of UAVs for network slice extension was presented in [52], published on ACM Transactions on Internet Technologies (TOIT), and in [53], published in IEEE Journal on Selected Areas in Communication (JSAC), and consists in the introduction of a System Controller (SC) with the target of deciding the number of active CPUs at runtime (in the first paper) together with the maximum number of offloaded job received by the ground devices (in the second paper) by maximizing an objective function weighing power consumption, job loss probability, and processing latency. Moreover, Reinforcement Learning (RL) technique was implemented to support SC in its decisions, with the target of maximizing a long-term reward function. Furthermore, in the considered scenario, the devices positioned on the ground can decide whether to process the data locally, or apply offloading policies. Offload can be either coarse-grained (also referred to total offloading), in which full tasks are migrated to the UAV, or fine-grained (also referred to as partial offloading), aimed at transmitting as little

code as possible, the one containing only the computation-hungry parts of the application.

The idea of using RL is due to the fact that, in recent years, machine learning techniques are registering an increasing use in networks, and more and more will have them in the 5G network and, even more so, in the future 6G network [54]. Their ability to work in complex contexts allows them to implement decision making techniques to provide high performance in contexts such as resource allocation, optimal routing policies associated with the prediction of network traffic (due to congestion or possible failures) especially in contexts such as IoT in which, given the limited availability of resources that sensors and actuators have. To have an idea of the possible applications, in [55] and [56], the use of RL to manage computational resources and to take decisions in cloud contexts with environmental changes was presented by the authors.

### **Scenario without cooperation between UAVs**

The idea proposed in [52] presents some differences as compared with other works, differing both in the approach and in the target. In fact, most of the previous works applied RL by simulation, trying an action and evaluating the consequent reward, while RL approach used in this article is model-based: the system is modeled by a Markov chain, and optimization is realized a priori. In this way, there is the possibility to present numerical results analytically derived. Another difference is that RL is used to optimize an unique objective function that capture the main performance indices regarding the application requirements (job loss probability, processing latency) but also the duration of the flight (in this case, the amount of battery consumed by the computing elements on board of UAVs). This last element constitutes a specific peculiarity of this work, since, when using a UAV to provide computing services, power consumption due to transmissions UAV-2-UAV, and UAV-2-device on ground, becomes negligible, while consumption due to the computing element must be taken into consideration, because it becomes comparable with one of the UAV engines, thus strongly impacting the flight duration. It has been preferred to use RL to supervised algorithms, since RL does not rely strictly on set of “supervised” data (the training set), but relies on being able to monitor the response of the actions taken and measures against a definition of a “reward.” However, unsupervised learning was not use because the expected reward is known upfront.

As shown in Fig. 3.22a, each UAV is equipped with computing elements that provide MEC facilities. Also in this case, data generated by IoT devices are orga-

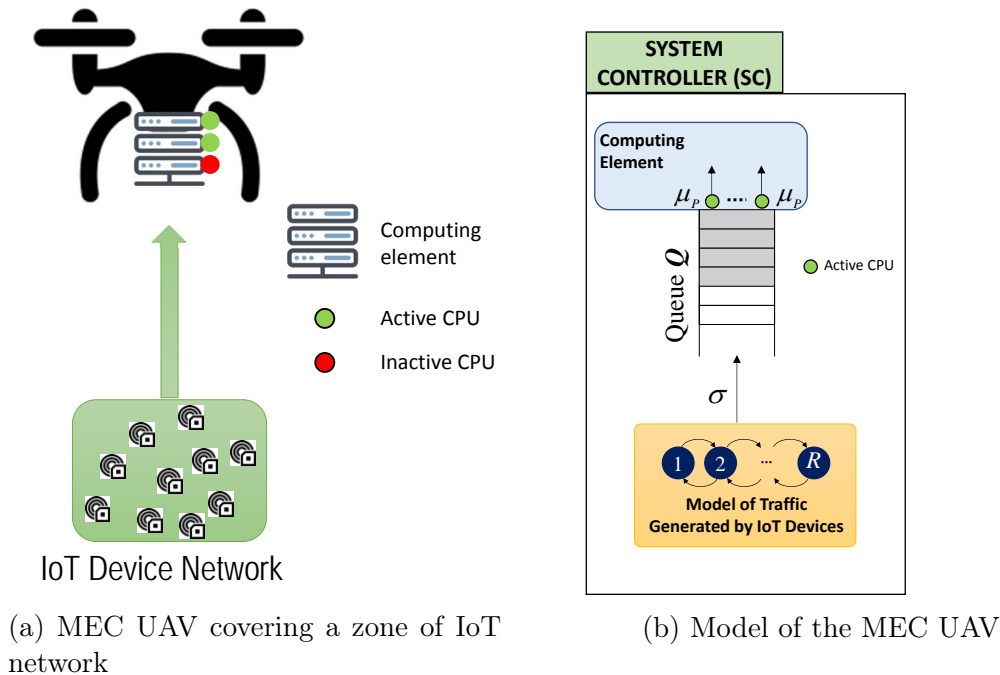


Figure 3.22: An IoT network zone covered by MEC UAV

nized in jobs that are offloaded to the UAV covering their area to be processed. Securing data processing is an important aspect that must be considered in any computing platform, but it is out of the scope of the considered work, where the behavior of a single UAV is considered, without cooperation with other UAVs.

As described in the previous section, the behavior of each zone is modeled with different states (in particular with a low activity state (LA) and a high activity state (HA)). Each state is characterized by a specific emission rate process of jobs and each job is assumed requiring the same computation complexity, independently of the state of the zone. The basic idea is that, indicating the number of available CPUs as  $L$ , the number of active CPUs is changed at runtime to be able to manage time-variant situations while considering the above target issues. So, when the assigned area is in high-activity state, the number of active CPUs is increased, and this can be changed at runtime also when the priority of the above target issues changes according to the current mission.

In any RL problem, the Agent, here represented by the SC, takes actions and receives observations from the environment. An action  $a$  is one of all the possible decisions the agent can take. In our case, an action consists of the choice of the number of active CPUs. A policy  $\rho$  defines the action  $a$  to be taken for each system state  $s_\Sigma$ , i.e.,  $a = \rho(s_\Sigma)$ . Received information consists of a reward for the performed action and the new state of the environment. That reward informs the

Agent of how good or bad was the taken action. RL tries to figure out what to do to maximize received rewards, and it does this by itself.

SC component takes this decision by applying a model-based RL to maximize a reward function weighing the three above elements and representing the immediate reward. The objective reward function is defined as follows:

$$F_{RW} = -k_1 \frac{\bar{\xi} - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} - k_2 \frac{\bar{\lambda} - \bar{\lambda}_{MIN}}{\bar{\lambda}_{MAX} - \bar{\lambda}_{MIN}} - k_3 \frac{\bar{\delta} - \bar{\delta}_{MIN}}{\bar{\delta}_{MAX} - \bar{\delta}_{MIN}} \quad (3.30)$$

where  $\bar{\xi}$  represents the power consumption,  $\bar{\lambda}$  the log10 of the job loss probability, and  $\bar{\delta}$  the mean job processing latency. These parameters are weighed by the constants  $k_1$ ,  $k_2$ , and  $k_3$ , representing the importance that the three above parameters have for the considered application scenario and normalized in the interval  $[0, 1]$  to be comparable. Since the three parts of the objective reward function weighed by the three constants are correlated to each other (for example, privileging performance in terms of job loss probability will also privilege performance in terms of processing latency), the impact on each performance parameter is not linear with the value of the relevant constant. Therefore, it is not possible to decide a priori a set of constants  $k_1$ ,  $k_2$ , and  $k_3$  to directly achieve the desired performance set. For this reason it is necessary the implementation of an analytical model, in such a way as to run it a few times offline to achieve a fine-tuning of the parameters.

Actions are taken by the SC periodically at the beginning of each time slot, whose duration is indicated as  $\Delta$ . As shown in Fig. 3.22b, after the offloading phase, jobs are enqueued in the queue  $Q$  of the UAV, where they wait for some available CPU according to a FIFO policy. Let  $K$  be the maximum number of jobs that can be accommodated in the queue  $Q$ , and  $\mu P$  is the job service rate of each active CPU. A key element is the *Immediate Reward*,  $R(n)$ , a scalar feedback, positive or negative, that the agent receives from the environment, which allows to quantify the success or failure of its actions according to its specific goal. A *Cumulative Reward* indicated as  $G(n)$ , is the long-term reward, used by the agent if it should not be greedy by taking actions associated with maximum reward at the current time only, but it has to plan ahead. The cumulative reward is defined as:

$$G(n) = \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1), \gamma \in [0, 1] \quad (3.31)$$

where  $\gamma$  is the *discount factor*, an input parameter, that informs the agent of how much it should care about rewards now as compared to rewards in the future. If  $\gamma = 0$  the agent only cares about the immediate reward. On the contrary, a value of  $\gamma = 1$  means that the agent cares about all future rewards.

For each state of the environment, a *State-value Function* and an *Action-value Function* are defined to tell the agent how good it is to be in a particular state and how good it is to take a particular action. In other words, they inform the agent of how much reward to expect if it takes a particular action in a particular state, being a prediction of expected future rewards used to evaluate goodness/badness of states. For a given policy  $\rho$ , the State-value function for a state  $s_\Sigma$  is defined as:

$$v_\rho(s_\Sigma) = E_\rho\{G(n)|S(n) = s_\Sigma\} \quad (3.32)$$

where  $E_\rho\{\}$  is the expected value given that the agent follows the policy  $\rho$ . It represents the expected return when the system starts from the state  $s_\Sigma$  and follows the policy  $\rho$ . The Action-value function represents the value of taking the action  $a$  in the state  $s_\Sigma$  under a policy  $\rho$ , and is defined as follows:

$$q_\rho(s_\Sigma, a) = E_\rho\{G(n)|S(n) = s_\Sigma, A(n) = a\} \quad (3.33)$$

An MDP  $\Sigma$  that describes the environment, for a given policy  $\rho$  specifying an action  $a$  for each state  $s_\Sigma$ , is completely defined by the tuple

$$(\mathfrak{S}^{(\Sigma)}, \mathfrak{S}^{(A)}, P^{(\Sigma|\rho)}, \Psi^{(\Sigma|\rho)}, \gamma) \quad (3.34)$$

where:

- $\mathfrak{S}^{(\Sigma)}$  is the system state space;
- $\mathfrak{S}^{(A)}$  is a finite set of action;
- $P^{(\Sigma|\rho)}$  is the state transition probability matrix. It depends on the policy  $\rho$  that specifies the action for each starting state. Its generic element represents the transition probability from the state  $s'_\Sigma$  to the state  $s''_\Sigma$  when, according to the policy  $\rho$ , the action  $a$  is performed at the beginning of the slot  $n$ . It is given by:

$$P_{[s'_\Sigma, s''_\Sigma]}^{(\Sigma|a)} = Pr\{S^{(\Sigma)}(n) = s''_\Sigma | S^{(\Sigma)}(n-1) = s'_\Sigma, A(n) = a\} \quad (3.35)$$

- $\Psi^{(\Sigma|\rho)}$  is the immediate reward matrix. Its generic element, representing the immediate reward received performing the action  $a$  at the slot  $n$  when the system transits from  $s'_{\Sigma}$  to  $s''_{\Sigma}$ , is given by:

$$\Psi_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} = E\{R(n)|S^{(\Sigma)}(n-1) = s'_{\Sigma}, S^{(\Sigma)}(n) = s''_{\Sigma}, A(n) = a\} \quad (3.36)$$

- $\gamma$  is the discount factor.

Substituting (3.31) in (3.32) and applying the theorem of total probability to all the possible arrival states  $S^{(\Sigma)}(n) = s''_{\Sigma}$  after the transition from the slot  $(n-1)$  to the slot  $n$ :

$$\begin{aligned} v_{\rho}(s'_{\Sigma}) &= E_{\rho}\{G(n-1)|S^{(\Sigma)}(n-1) = s'_{\Sigma}\} \\ &= E_{\rho}\left\{\sum_{k=0}^{\infty} \gamma^k \cdot R(n+k)|S^{(\Sigma)}(n-1) = s'_{\Sigma}\right\} \\ &= E_{\rho}\left\{R(n) + \gamma \cdot \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1)|S^{(\Sigma)}(n-1) = s'_{\Sigma}\right\} \end{aligned} \quad (3.37)$$

and applying the total probability theorem on the state  $S(n)$ :

$$\begin{aligned} v_{\rho}(s'_{\Sigma}) &= \sum_{\forall s''_{\Sigma} \in \mathfrak{S}^{(\Sigma)}} P_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} \cdot [\Psi_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} + \gamma \cdot E_{\rho}\{\sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1)|S^{(\Sigma)}(n) = s''_{\Sigma}\}] \\ &= \sum_{\forall s''_{\Sigma} \in \mathfrak{S}^{(\Sigma)}} P_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} \cdot [\Psi_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} + \gamma \cdot v_{\rho}(s''_{\Sigma})] \end{aligned} \quad (3.38)$$

The expression in Equation (3.38) is the Bellman equation for the state  $s'_{\Sigma}$  and gives a relationship between the value of a state  $s'_{\Sigma}$  and the values of its successive states. The value of the start state must equal the value of the expected next state, plus the reward expected along the way. The state-value function  $v_{\rho}(s_{\Sigma})$ , for each  $s_{\Sigma} \in \mathfrak{S}^{(\Sigma)}$ , is the unique solution to the Bellman equation and its existence and uniqueness are guaranteed as long as the discount factor is  $\gamma < 1$ .

Considering that  $\rho^* \in \mathfrak{S}^{(\rho)}$  represents an optimal policy if its state-value function is better than or equal to the state-value function of all the other policies [57], for all the states of the system, it is possible to write:



$$v_{\rho^*}(s_{\Sigma}) = \max_{\forall \rho \in \mathfrak{S}(\rho)} \{v_{\rho}(s_{\Sigma})\}, \forall s_{\Sigma} \in \mathfrak{S}^{(\Sigma)} \quad (3.39)$$

where  $v_{\rho^*}(s_{\Sigma})$  is the state-value function for a policy and must satisfy the self-consistency condition given by the Bellman equation. For this reason (applying the definition of  $G(n)$  and the total probability theorem):

$$\begin{aligned} v_{\rho^*}(s'_{\Sigma}) &= \max_{\forall a \in \mathfrak{S}^{(a)}} E_{\rho^*} \{G(n-1) | S(n-1) = s'_{\Sigma}, A(n) = a\} \\ &= \max_{\forall a \in \mathfrak{S}^{(a)}} E_{\rho^*} \left\{ \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k) | S(n-1) = s'_{\Sigma}, A(n) = a \right\} \\ &= \max_{\forall a \in \mathfrak{S}^{(a)}} E_{\rho^*} \left\{ R(n) + \gamma \cdot \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1) | S(n-1) = s'_{\Sigma}, A(n) = a \right\} \\ &= \max_{\forall a \in \mathfrak{S}^{(a)}} E_{\rho^*} \{R(n) + \gamma v_{*}(S(n)) | S(n-1) = s'_{\Sigma}, A(n) = a\} \\ &= \max_{\forall a \in \mathfrak{S}^{(a)}} \left( \sum_{\forall s''_{\Sigma} \in \mathfrak{S}^{(\Sigma)}} P_{[s'_{\Sigma}, s''_{\Sigma}]^{(\Sigma|a)}} \cdot [R_{[s'_{\Sigma}, s''_{\Sigma}]^{(\Sigma|a)}} + \gamma \cdot v_{*}(s''_{\Sigma})] \right) \end{aligned} \quad (3.40)$$

The Bellman optimality equation for the state  $s'_{\Sigma}$  under the optimal policy was derived. It is a system of equation, one for each state. So, if there are  $N$  states, then there are  $N$  equations in  $N$  unknowns. The solution of this system gives the optimal state-value function  $v_{\rho^*}(s_{\Sigma})$ , for each state  $s_{\Sigma} \in \mathfrak{S}^{(\Sigma)}$ . Once the optimal state-value function  $v_{\rho^*}(s_{\Sigma})$  is known for all the states, the optimal policy can be determined. For each state  $s_{\Sigma}$  there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns non-zero probability only to these actions is an optimal policy. This can be easily argued, because any policy that is greedy with respect to the optimal evaluation function  $v_{\rho^*}(s_{\Sigma})$  is an optimal policy. Let note that, since  $v_{\rho^*}(s_{\Sigma})$  already takes into account the reward consequences of all possible future behavior, using  $v_{\rho^*}(s_{\Sigma})$ , the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the (discounted) long-term optimal actions.

The proposed system was modeled with a discrete-time Markov chain. To this purpose, the job arrival process from the device in the considered zone is modeled as a Switched Batch Bernoulli Process (SBBP), the most general arrival process in the discrete-time domain, able to capture both first- and second-order statistics

of any arrival process [58]. Let indicate this job arrival process with  $Z(n)$ . Let  $\Omega^{(Z)}$  be the set of all the possible numbers of job arrivals from a zone. The number of arrivals follows a probability density function (pdf) modulated by the state  $S^{(Z)}(n)$  of the underlying Markov chain of the SBBP  $Z(n)$ . Therefore,  $Z(n)$  is characterized by the transition probability matrix of the underlying Markov chain,  $P^{(Z)}$  and the job emission probability matrix,  $B^{(Z)}$ , whose rows contain the arrival pdf for each state of the underlying Markov chain. These parameters are assumed as input of the problem.

The whole system is represented by a two-dimensional discrete-time Markov chain, based on Fig. 3.22b, whose state is defined as:

$$\underline{S}^{(\Sigma)}(n) = (S^{(Z)}(n), S^{(Q)}(n)) \quad (3.41)$$

where:

- $S^{(Z)}(n)$  is the state of the underlying Markov chain of the SBBP modeling the job arrival process. If  $\mathfrak{S}^{(Z)} = \{1, 2, \dots, R\}$  is the set of states characterizing the behavior of the zone,  $S^{(Z)}(n) \in \mathfrak{S}^{(Z)}$ ;
- $S^{(Q)}(n)$  is the state of the UAV queue  $Q$ , with  $S^{(Q)}(n) \in [0, \dots, K]$  the number of jobs in the queue at the slot  $n$ ; the term  $K$  is the maximum number of jobs that can be buffered in the queue.

To derive the transition probability matrix of the MDP described so far, let consider the following two generic states:

- $\underline{s}'_{\Sigma} = (s'_Z, s'_Q) = \underline{S}^{(\Sigma)}(n-1)$  at the slot  $n-1$
- $\underline{s}''_{\Sigma} = (s''_Z, s''_Q) = \underline{S}^{(\Sigma)}(n)$  at the slot  $n$

Considering the Fig. 3.23, the following event sequence are performed to transit from the slot  $n-1$  to the slot  $n$ :

- Decision of the new value  $a$  according to the best policy achieved by RL, based on the value of the current system state  $\underline{S}^{(\Sigma)}(n-1)$ ;
- Update of the zone state, with the transition from  $S^{(Z)}(n-1) = s'_Z$  to  $S^{(Z)}(n) = s''_Z$ ;
- Dequeue of some jobs from the queue  $Q$ , according to the number  $a$  of CPUs that have worked in the slot  $n$ ;

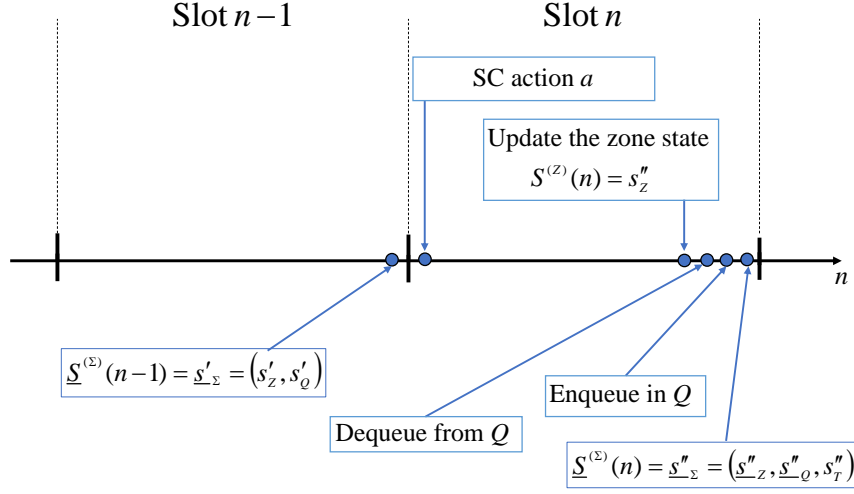


Figure 3.23: UAV and RL. Time diagram of the sequence of the events in a state transition

- Enqueue of a number of jobs,  $\sigma = Z(n)$ , according to the SBBP process modeling job arrivals, whose underlying Markov chain state,  $S^{(Z)}(n)$ , has been updated in the previous step;
- Update of the system state from  $\underline{S}^{(\Sigma)}(n-1)$  to  $\underline{S}^{(\Sigma)}(n)$ , according to the evolution of each of its components, as described in all the previous steps.

The generic element of the transition probability matrix can be defined as follows:

$$P_{[\underline{s}'_{\Sigma}, \underline{s}''_{\Sigma}]}^{(\Sigma|a)} = P_{[\underline{s}'_Z, \underline{s}''_Z]}^{(Z)} \cdot P_{[\underline{s}'_Q, \underline{s}''_Q]}^{(Q|a)}(s''_Z) \quad (3.42)$$

where the matrix  $P^{(Z)}$  was already defined, while  $P^{(Q|a)}(s''_Z)$  represents the behavior of the queue  $Q$ . It depends on the arrival state of the underlying Markov chain of the zone, and this dependence determines the number of job arrivals in the queue, and the applied policy, and consequently the action  $a$  for each starting state. Its generic element can be derived as follows:

$$P_{[\underline{s}'_Q, \underline{s}''_Q]}^{(Q|a)}(s''_Z) = Pr\{S^{(Q)}(n) = s''_Q | S^{(Q)}(n-1) = s'_Q, A(n) = a\} \quad (3.43)$$

To evaluate this probability, let us indicate the number of jobs that can be served in one slot by an active CPU as  $\chi_p = \mu_p \cdot \Delta$ , where  $\mu_p$  is the job serving rate expressed in job/s. If  $\chi_p$  is not an integer, let model the service rate of an active CPU by assuming that it serves:

- $\bar{\chi}_p^{(+)} = \lceil \chi_p \rceil$  jobs with a probability  $p_{\chi^+} = \chi_p - \lfloor \chi_p \rfloor$ ;

–  $\bar{\chi}_p^{(-)} = \lfloor \chi_p \rfloor$  jobs with a probability  $p_{\chi^-} = 1 - p_{\chi^+}$

Applying the total probability theorem to the number  $\sigma \in \Omega^{(Z)}$  of possible arrivals and to the number of departure  $c_i \in \{\bar{\chi}_p^{(-)}, \bar{\chi}_p^{(+)}\}$  from the  $i$ -th active CPU, with  $i \in \{1, \dots, a\}$ , we have:

$$\begin{aligned}
 P_{[s'_Q, s''_Q]}^{(Q|a)}(s''_Z) &= \sum_{\forall \sigma \in \Omega^{(Z)}} \sum_{i=1}^a \sum_{\forall [c_1, \dots, c_a]} B_{[s''_Z, \sigma]}^{(Z)} \cdot Pr\{[c_1, \dots, c_a] \text{departures}\} \\
 &\cdot Pr\left\{S^{(Q)}(n) = s''_Q | S^{(Q)}(n-1) = s'_Q, A(n) = a, Z(n) = \sigma, [c_1, \dots, c_a]\right\}
 \end{aligned} \tag{3.44}$$

The first probability term can be easily calculated assuming that the behaviors of the queue servers are independent of each other. Therefore, the joint service probability for all the CPUs is the product of marginal probability for each CPU among the  $a$  that are active.

The second probability term in (3.44) can be calculated by remembering the event sequence shown in Fig. 3.23 and considering that the queue state  $S^{(Q)}(n)$  can never become negative and never exceed  $K$ . Therefore, it is possible to consider that the starting state of the queue,  $s'_Q$ , is decreased by the departures,  $\sum_{i=1}^a c_i$ . Then, after imposing that this is not negative, the number of arrivals,  $\sigma$ , is added and finally impose that the last result is not greater than the maximum queue size  $K$ . This means that:

$$Pr\left\{S^{(Q)}(n) = s''_Q | S^{(Q)}(n-1) = s'_Q, A(n) = a, Z(n) = \sigma, [c_1, \dots, c_a] \text{departures}\right\} = \tag{3.45}$$

$$= \begin{cases} 1 & \text{if } s''_Q = \min\{\max\{s'_Q - \sum_{i=1}^a c_i, 0\} + \sigma, K\} \\ 0 & \text{otherwise} \end{cases}$$

Now, the expected value of the immediate reward for a given transition from the generic state  $\underline{S}^{(\Sigma)}(n-1)$ , to the generic state  $\underline{S}^{(\Sigma)}(n)$ , and for a given action  $a$  taken according to the state  $\underline{S}^{(\Sigma)}(n-1)$ , can be calculated by weighing the three key parameters characterizing the system behavior. More in detail, from (3.30), the reward is defined as:

$$\Psi_{[s'_\Sigma, s''_\Sigma]}^{(\Sigma|a)} = -k_1 \frac{\bar{\xi}(a) - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} - k_2 \frac{\bar{\lambda}(s'_\Sigma, a) - \bar{\lambda}_{MIN}}{\bar{\lambda}_{MAX} - \bar{\lambda}_{MIN}} - k_3 \frac{\bar{\delta}(s''_\Sigma, a) - \bar{\delta}_{MIN}}{\bar{\delta}_{MAX} - \bar{\delta}_{MIN}} \quad (3.46)$$

where:

- the first term, regarding the penalty (it becomes a reward, thanks to the minus sign) due to the power consumption depending on the number of CPUs switched on by the SC when the action  $a$  is performed according to the starting state  $s'_\Sigma$ , is defined as:

$$\bar{\xi}(a) = a \cdot \xi_{\mu P} \quad (3.47)$$

with  $\xi_{\mu P}$  equal to the power consumption of each CPU;

- the second term,  $\bar{\lambda}(s'_\Sigma, a)$ , is the penalty related to the job loss for queue overflows. Starting from the knowledge of the starting and arrival states, it is calculated as the  $\log_{10}$  of the expected number of jobs lost in the considered transition:

$$\begin{aligned} \bar{\lambda}(s'_\Sigma, a) = \log_{10} & \sum_{\forall \sigma \in \Omega^{(Z)}} \frac{1}{\sigma} \sum_{i=1}^a \sum_{\forall [c_1, \dots, c_a]} B_{[s''_Z, \sigma]}^{(Z)} \cdot Pr\{[c_1, \dots, c_a] \text{departures}\} \cdot \\ & \cdot \max \left\{ \max \left\{ s'_Q - \sum_{i=1}^a c_i, 0 \right\} + \sigma - K, 0 \right\} \end{aligned} \quad (3.48)$$

- the third term regards the processing latency, defined as the delay suffered in the queueing system. To this purpose, assuming that conditions of this queue remain constant in the future, the SC calculates its expected value as follows:

$$\bar{\delta}(s'_\Sigma, s''_\Sigma, a) = \left[ \frac{s''_{Q_1}}{a \cdot \chi_p} \right] \quad (3.49)$$

where  $\chi_p$  is the number of jobs that can be served in one slot by an active CPU.

To evaluate the performance of this proposed framework, the next step is to calculate the steady-state probability array, whose generic element is:

$$\pi_{[s_\Sigma]}^{(\Sigma)} = \lim_{n \rightarrow \infty} Pr \{ S^{(\Sigma)}(n) = s_\Sigma \} \quad (3.50)$$

and can be calculated by solving the following linear equation system, in which  $\underline{\pi}^{(\Sigma)}$  is a row array containing the steady-state probabilities of all the state in  $\mathfrak{S}^{(\Sigma)}$ :

$$\begin{cases} \underline{\pi}^{(\Sigma)} \cdot P^{(\Sigma)} = \underline{\pi}^{(\Sigma)} \\ \underline{\pi}^{(\Sigma)} \cdot \underline{\mathbf{1}}^T = 1 \end{cases} \quad (3.51)$$

Finally, the three main performance parameters characterizing the reward function are formulated.

The mean power consumption, indicating  $a(s'_{\Sigma})$  as the number of CPUs activated by the SC in the system state  $s'_{\Sigma}$ , is defined as:

$$\bar{\xi} = \sum_{\forall s'_{\Sigma}} a(s'_{\Sigma}) \pi_{[s'_{\Sigma}]}^{(\Sigma)} \cdot \xi_{\mu P} \quad (3.52)$$

The mean processing latency is calculated through the Little law applied to the queue:

$$\bar{\delta} = \frac{\bar{N}_Q}{\bar{\Lambda}_Q} \quad (3.53)$$

with

$$\bar{N}_Q = \sum_{\forall s_{\Sigma}} s_Q \cdot \pi_{[s_{\Sigma}]}^{(\Sigma)} \quad (3.54)$$

$$\bar{\Lambda}_Q = \sum_{\forall s'_{\Sigma} \in \mathfrak{S}^{(\Sigma)}} \sum_{\forall s''_Z \in \mathfrak{S}^{(Z)}} \sum_{\forall \sigma \in \Omega^{(Z)}} \min\{\sigma, K - s'_{Q_1}\} \cdot B_{[s''_Z, \sigma]}^{(Z)} \cdot P_{[s'_Z, s''_Z]}^{(Z)} \cdot \pi_{[s'_{\Sigma}]}^{(\Sigma)} \quad (3.55)$$

About the per-slot loss probability, it can be calculated by averaging the number of lost jobs when the system moves from the state  $s'_{\Sigma}$  to the state  $s''_{\Sigma}$  and the SC takes the action a:

$$\bar{\lambda} = \sum_{\forall s'_{\Sigma} \in \mathfrak{S}^{(\Sigma)}} \sum_{\forall s''_Z \in \mathfrak{S}^{(Z)}} \bar{\lambda}(s'_{\Sigma}, a) \cdot P_{[s'_{\Sigma}, s''_Z]}^{(\Sigma|a)} \cdot \pi_{[s'_{\Sigma}]}^{(\Sigma)} \quad (3.56)$$

Now, it is possible to consider a case study to apply the proposed framework and evaluate some numerical results, aimed at both showing how the proposed framework behaves and the gain achieved in respect to the state-of-the-art.

An IoT network installed in a geographic area covered by a FANET of rotary-wing UAVs moving at an average speed of  $5m/s$  is considered. It is assumed a power consumption of the engine of the UAV equal to  $P_E^{(UAV)} = 1 - 3kW$  and that the UAV is equipped with a i7-2600K computer element with 32 GB DDR4 SO-DIMM ram, six CPUs available to be used for computing services, and Ubuntu Server 16.04 64 bit installed as operating system. Power consumption of

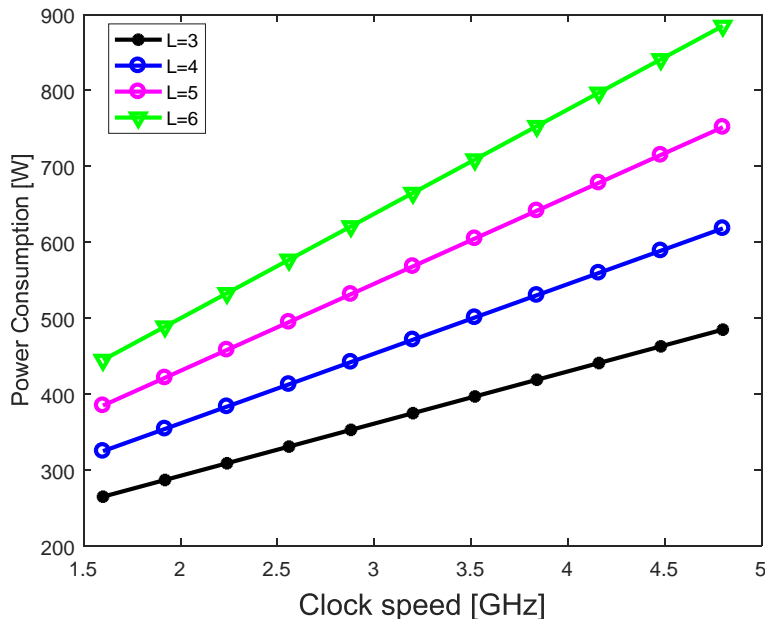


Figure 3.24: i7-2600K power consumption vs. clock speed for different numbers of active CPUs

the computing element depends on the clock speed and is time variant according to the current number of CPUs activated by the SC.

In Fig. 3.24, there is shown the variation of the power consumption when the clock speed ranges in the interval  $[1.6, 4.8]GHz$  and a different number of CPUs is active.

Let consider a slot duration of  $\Delta = 300ms$  and assume that each CPU is able to serve one job when its clock speed is equal to  $3.2GHz$ . Therefore, assuming a linear job service rate as a function of the clock speed, when the CPU clock speed ranges in the interval  $[1.6, 4.8]GHz$ , the job service rate of each CPU ranges in the interval  $[0.5, 1.5]$  job/slot. Moreover, it is assumed that at most  $K = 15$  jobs can be accommodated in the queue of the UAV.

As far as the job arrival process is concerned, two cases, measured by a smart-agriculture monitoring platform available on a big farmland near Catania (Italy), are considered: the first case, LA, is characterized by a job arrival rate with an average value of 4 jobs/slot while the second one, labeled as HA, is characterized by a job arrival rate with an average value of 5 jobs/slot. Starting from the real traces, and applying the inverse eigenvalue technique in the discrete-time domain [58], the arrival processes in the two cases was modeled with the SBBPs  $Z_{LA}(n)$  and  $Z_{HA}(n)$  characterized by the following matrices:

$$Q^{(Z_{LA})} = \begin{bmatrix} 0.864 & 0.136 \\ 0.143 & 0.857 \end{bmatrix} \quad Q^{(Z_{HA})} = \begin{bmatrix} 0.950 & 0.050 \\ 0.333 & 0.667 \end{bmatrix} \quad (3.57)$$

$$B^{(Z_{LA})} = \begin{bmatrix} & \langle \sigma = 2 \rangle & \langle \sigma = 3 \rangle & \langle \sigma = 4 \rangle & \langle \sigma = 5 \rangle \\ \langle s_z = 1 \rangle & 0.07 & 0.19 & 0.74 & 0 \\ \langle s_z = 2 \rangle & 0 & 0.21 & 0.25 & 0.54 \end{bmatrix}$$

$$B^{(Z_{HA})} = \begin{bmatrix} & \langle \sigma = 4 \rangle & \langle \sigma = 5 \rangle & \langle \sigma = 6 \rangle & \langle \sigma = 7 \rangle \\ \langle s_z = 1 \rangle & 0.27 & 0.66 & 0.07 & 0 \\ \langle s_z = 2 \rangle & 0 & 0.12 & 0.46 & 0.42 \end{bmatrix} \quad (3.58)$$

During the experiments, three different scenarios were considered, each characterized by different values of the configuration setup  $K = (k1, k2, k3)$  for the equation (3.70). More specifically:

- $K1 = (1, 1, 1)$ , where all the performance parameters are weighed the same way;
- $K2 = (1, 3, 1)$ , where higher weight to the loss probability was set;
- $K3 = (1, 3, 3)$ , where was also increased the weight of the processing latency.

A first analysis regards the performance, in terms of loss probability, processing latency, and mean number of active CPUs, achieved in the three cases of available CPUs  $L = 3$ ,  $L = 4$ , and  $L = 6$ . Results are shown in Fig. 3.25, 3.26 and 3.27 for the LA scenario and Fig. 3.28, 3.29, and 3.30 for the HA scenario. Curves compare the three configuration setups with the case in which RL is not applied representing the state-of-the-art (realized by considering all the available CPUs active).

As expected, the loss probability decreases with the job service rate. Its worst case is for  $K1$ , since in this case the SC gives the least priority to the job loss probability parameter, while the case  $K3$  is the best one, since increasing  $k3$ , although aimed at improving the processing latency, determines a queue length decrease, and therefore loss probability decreases as well. Job loss probability achieved for the no-RL case is, of course, better than the other cases, since no-RL technique uses all the available CPUs not taking care of the current and future states. Moreover, comparing the three cases of Fig. 3.25, let notice that curves are lower for greater numbers of available CPUs, and comparing these figures with the three cases of Fig. 3.28, it is possible to observe greater values of loss



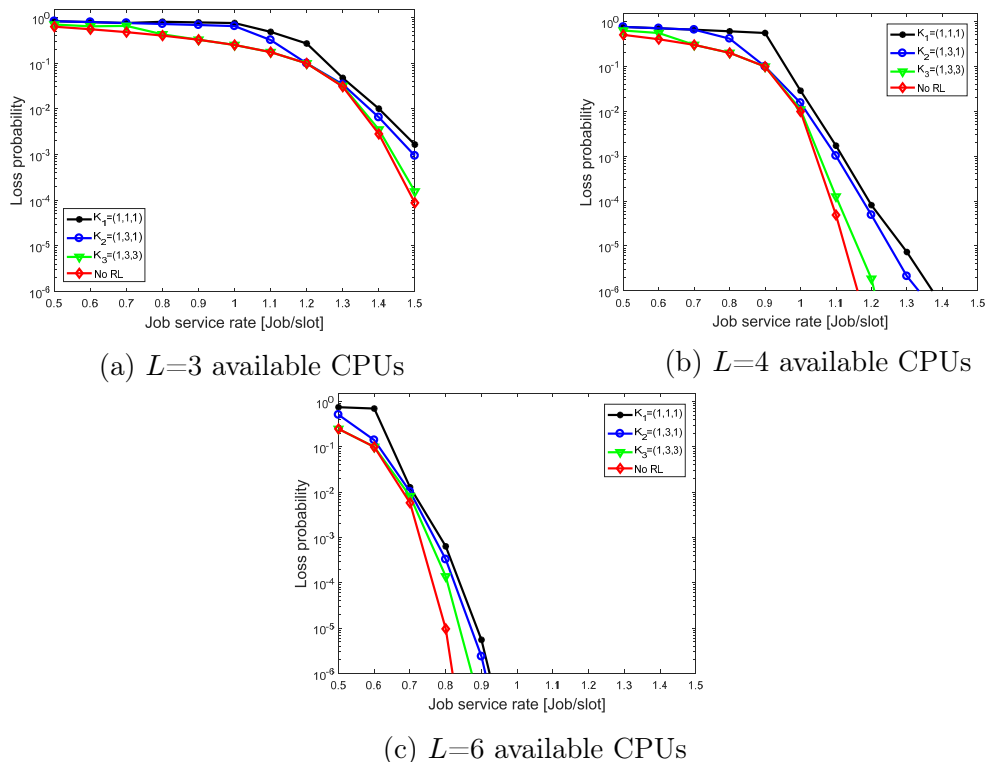


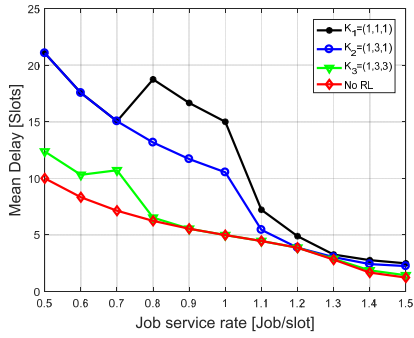
Figure 3.25: Job loss probability for the IoT devices in LA state, against the job service rate for each CPU

probability due to the higher load in HA state. In addition, Fig. 3.28 shows how  $L = 3$  active CPUs are not sufficient to support HA load, while  $L = 4$  and  $L = 6$  are able to provide acceptable loss probabilities but only for the considered setup  $K3$  and for high clock speed values.

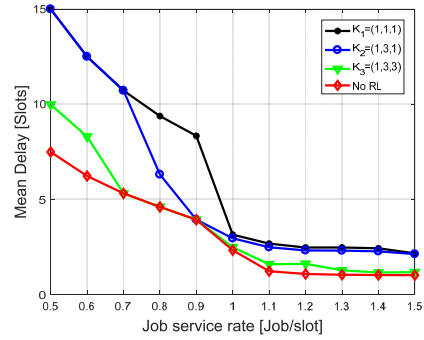
Job processing latency is shown in Fig. 3.26 and 3.29 for the two considered zone activity states, respectively. The non-monotonic behavior is due to the optimization process applied by the SC, which aims at maximizing the reward function by acting on the number of active CPUs. So, as expected, latency achieved for  $K1$  is not less than delay achieved for  $K2$ , which is not less than the one achieved for  $K3$ , while latency achieved when no-RL is applied is the best one. However, the non-monotonic behavior and the fact that some values are shared by two or more curves can be explained by the curves in Fig. 3.27 and 3.30, representing the number of active CPUs chosen by the SC. More specifically, the SC tends to activate all the CPUs in the central subrange of clock speed, while on the left part of the curves this is not useful (the clock speed is too small to try to optimize reward activating all the CPUs), and on the right part this is not necessary (the clock speed is so high that it is not necessary to use all the CPUs).

The central range depends on the number  $L$  of available CPUs and on the

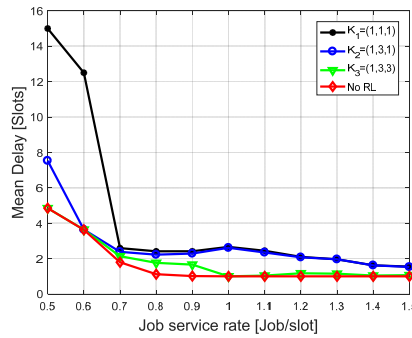
CHAPTER 3. MANAGEMENT AND ORCHESTRATION OF NETWORK SLICES



(a)  $L=3$  available CPUs

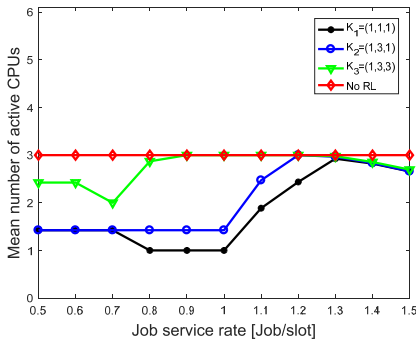


(b)  $L=4$  available CPUs

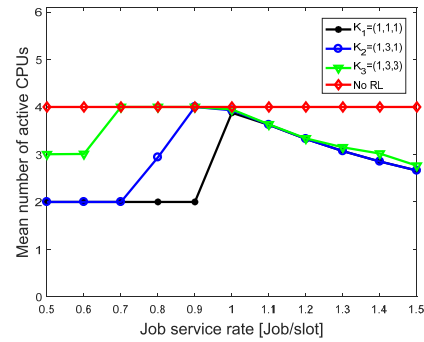


(c)  $L=6$  available CPUs

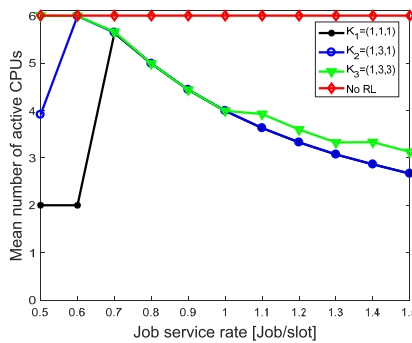
Figure 3.26: Job processing latency for the IoT devices in LA state



(a)  $L=3$  available CPUs



(b)  $L=4$  available CPUs



(c)  $L=6$  available CPUs

Figure 3.27: Mean number of active CPUs for the IoT devices in LA state

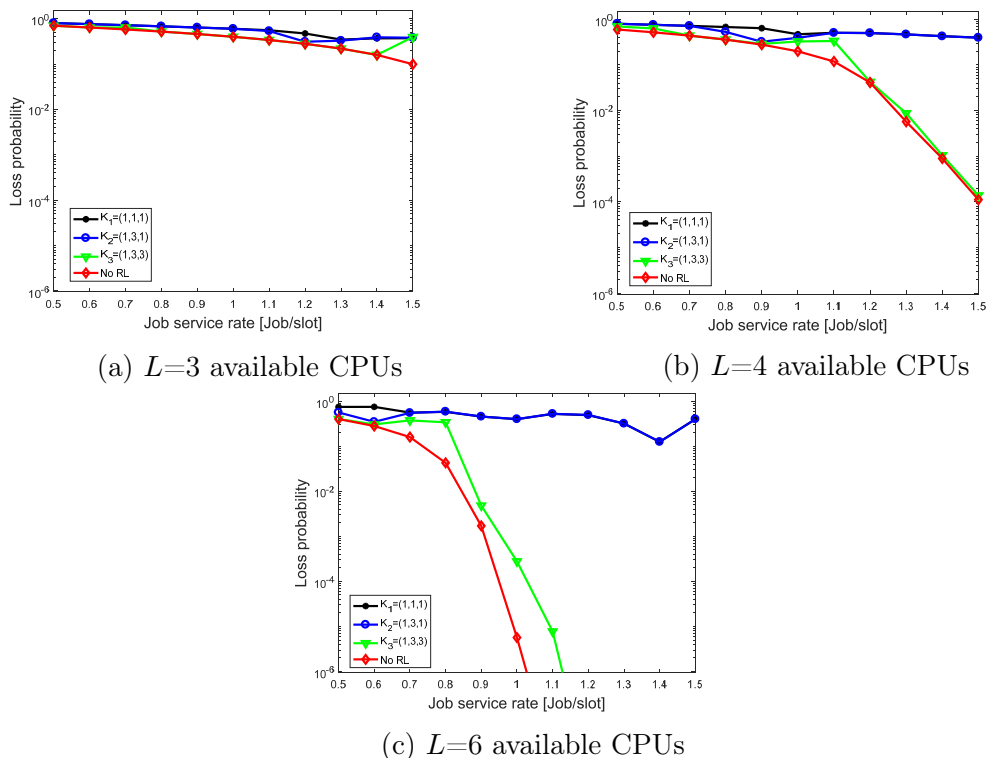


Figure 3.28: Job loss probability for the IoT devices in HA state, against the job service rate for each CPU

configuration setup. Of course, the mean number of active CPUs in the HA state is greater than the same number in the LA state. Moreover, looking at Figs. 3.27c and 3.30c, it is possible to see the power saving achieved with our mechanism. In fact, observing Fig. 3.25c and 3.26c, for higher clock speed and  $L = 6$ , loss probability is negligible and mean delay is very low. Therefore, in that case the SC can reduce the number of CPUs to be maintained active. This fact is more evident for the configuration setups  $K1$  and  $K2$ , where the SC takes better care of power consumption.

### Scenario with cooperation between UAV

In [53] was introduced the possibility of a cooperation between two nearby UAVs: if a UAV is overloaded of jobs, and typically this occurs when the area assigned to it is in the HA state, it asks for help to the closest UAV that is not stressed at that time.

In Fig. 3.31, the MEC UAV 1 is monitoring a HA area and has obtained availability for help from MEC UAV 2. In this way, UAV 1 and UAV 2 cooperate in order to optimize a common objective function weighing the power-saving related to CE power consumption, delay and loss probability experienced by both

## CHAPTER 3. MANAGEMENT AND ORCHESTRATION OF NETWORK SLICES

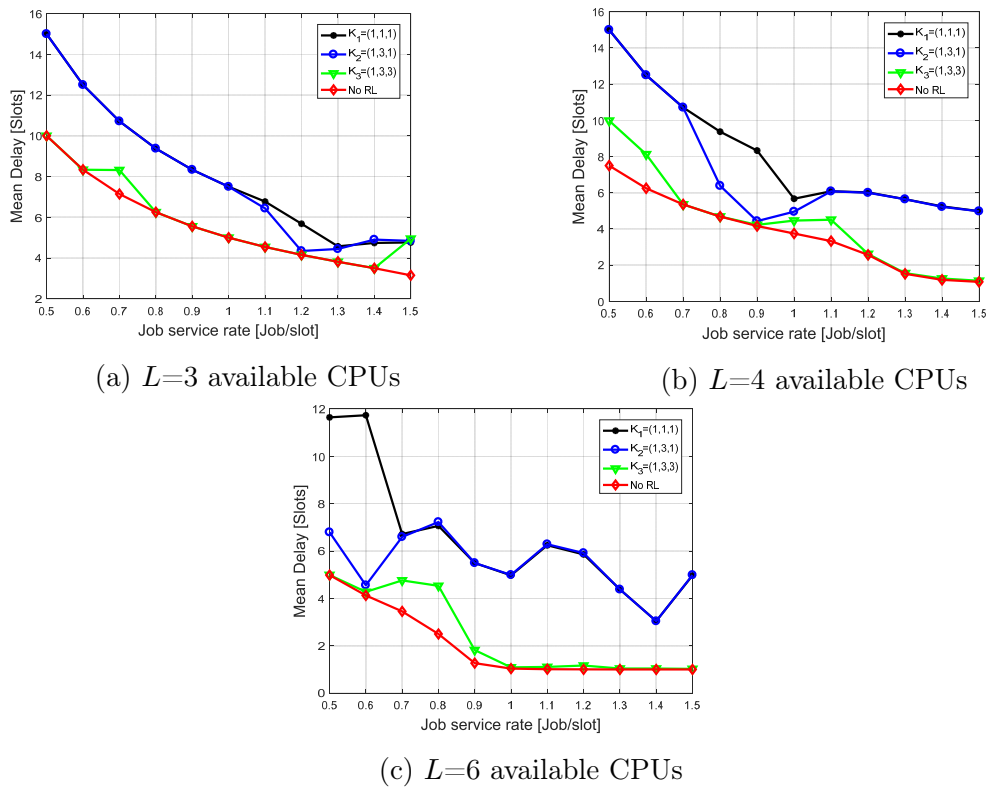


Figure 3.29: Job processing latency for the IoT devices in HA state

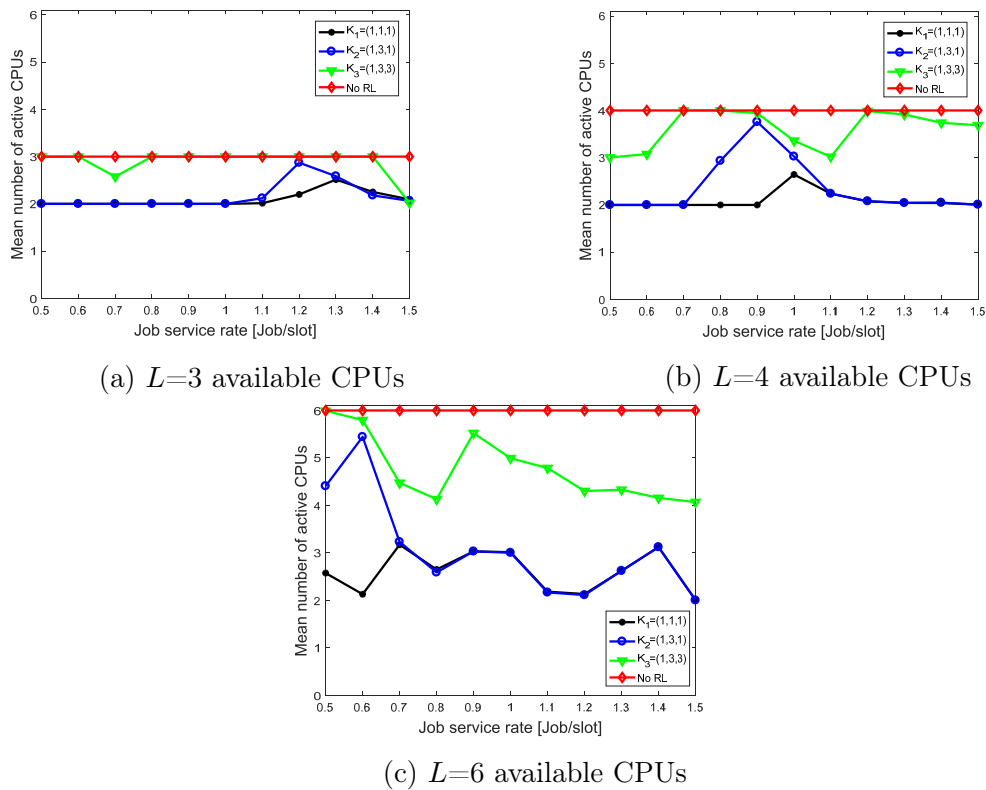


Figure 3.30: Mean number of active CPUs for the IoT devices in HA state

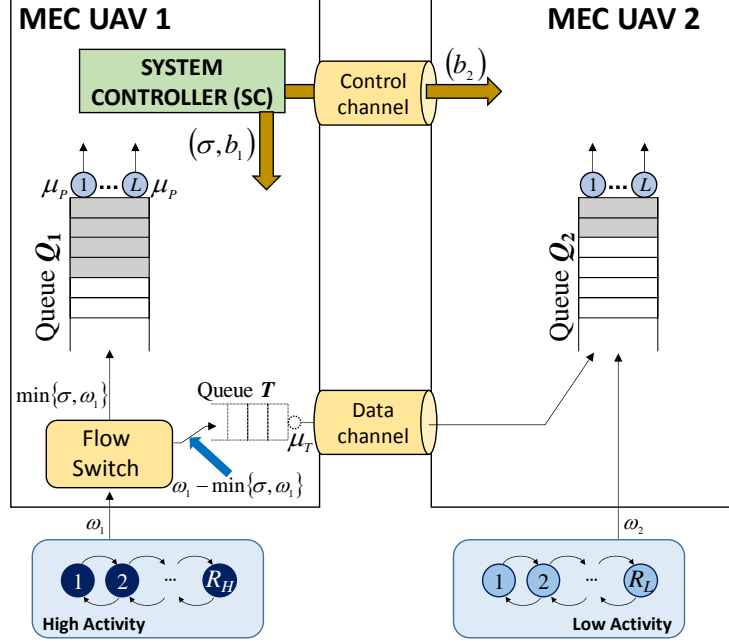


Figure 3.31: UAV cooperation scenario. Functional architecture of the helped and helping MEC UAVs

offloaded and non-offloaded jobs. The SC, in this case, has to decide not only how many CEs activating on UAV 1,  $b_1$ , but also the number of CPUs that has to be active in UAV 2,  $b_2$ , and the maximum number  $\sigma$  of jobs to be locally managed by UAV1 among the  $\omega_1$  jobs arrived by the HA area. The other  $(\omega_1 - \sigma)$  jobs are offloaded to UAV2. As in the previous case, the SC takes decisions using RL to maximize the reward function:

$$F_{RW} = -c_1 \frac{\bar{\xi} - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} - c_2 \frac{\bar{\lambda} - \bar{\lambda}_{MIN}}{\bar{\lambda}_{MAX} - \bar{\lambda}_{MIN}} - c_3 \frac{\bar{\delta} - \bar{\delta}_{MIN}}{\bar{\delta}_{MAX} - \bar{\delta}_{MIN}} \quad (3.59)$$

where  $\bar{\xi}$ ,  $\bar{\lambda}$  and  $\bar{\delta}$  represents the three parameters already described.

As shown in Fig. 3.31, two communications channels are activated between the UAVs: a data channel for transmission of offloaded jobs; a signaling channel to exchange control information between the SC and UAV 2. Due to the very low amount of information transmitted on the second channel, the delay on this channel is negligible.

Jobs to be locally processed by UAV 1 are enqueued in the queue  $Q_1$ , where they will wait to be processed with a FIFO policy. The offloaded jobs are enqueued in the transmission queue  $T$  to be transmitted to the other UAV, where will be enqueued together with the jobs arriving from the area monitored by that UAV.

Queue  $T$  is served with a bitrate depending on the quality of the wireless channel between UAV 1 and UAV 2. In the following, the job service rate of each CE is indicated as  $\mu_p$ , and the job transmission rate on the wireless channel between UAVs as  $\mu_T$ .

As before, it is possible to define a Cumulative Reward  $G(n)$  as in (3.31), a State-value Function (3.32) and an Action-value Function (3.33). A MDP  $\Sigma$  model is defined as already described in (3.34), with the same considerations about the transition probability matrix  $P^{(\Sigma|\rho)}$  (3.35) and the immediate reward matrix  $\Psi^{(\Sigma|\rho)}$  (3.36).

The Bellman equation (3.38) and the Bellman optimality equation (3.40) are derived in the same way described in the previous case.

The proposed system was modeled with a three-dimensional discrete-time Markov chain, whose state is defined as:

$$\underline{S}^{(\Sigma)}(n) = \left( \underline{S}^{(Z)}(n), \underline{S}^{(Q)}(n), S^{(T)}(n) \right) \quad (3.60)$$

It is different respect the case of UAVs working without cooperation because now the model has to consider not only the state of the zone  $Z_1$  and of the queue  $Q_1$ , but also the state of  $Z_2$ , the queue of UAV2  $Q_2$ , and the state of the data channel  $T$ . More in detail:

- $\underline{S}^{(Z)}(n) = (S^{(Z_1)}(n), S^{(Z_2)}(n))$ . Let  $\mathfrak{S}^{(Z_1)} = \{1, 2, \dots, R_H\}$  be the set of states characterizing the behavior of area 1, being it in the HA state, and  $\mathfrak{S}^{(Z_2)} = \{1, 2, \dots, R_L\}$  be the set of states characterizing the behavior of area 2, being it in LA state. As in the previous case,  $S^{(Z_i)}(n) \in \mathfrak{S}^{(Z_i)}, i \in \{1, 2\}$  is the state of area  $i$  at the slot  $n$  and, defining  $\Omega^{(Z)}$  the set of all the possible numbers of job arrivals from an area, it is characterized by the transition probability matrix  $P^{(Z_H)}$  and  $P^{(Z_L)}$  and by the job emission probability matrix  $B^{(Z_1)}$  and  $B^{(Z_2)}$  for area 1 and area 2 respectively;
- $\underline{S}^{(Q)}(n) = (S^{(Q_1)}(n), S^{(Q_2)}(n))$  represents the state of UAV queues  $Q_1$  and  $Q_2$ , being  $S^{(Q_i)}(n) \in [0, \dots, K], i \in \{1, 2\}$ .  $K$  is the maximum number of jobs that can be buffered in each of these queue;
- $S^{(T)}(n) \in [0, \dots, M]$  is the state of the transmission queue for offloading from UAV 1 to UAV 2, where  $M$  is the maximum number of jobs that can be accommodated in queue  $T$ .

Assuming that the time needed to process one job in one of the CEs is less than the time  $t_T = 1/\mu_T$  needed to transmit one job from UAV 1 to UAV 2, in the

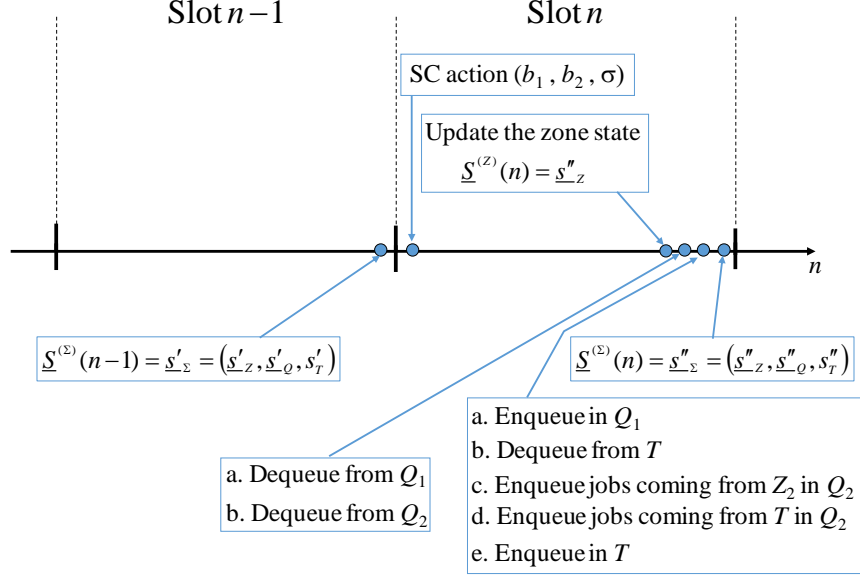


Figure 3.32: UAVs collaboration and RL. Time diagram of the sequence of the events in a state transition

following the average time needed to process one job in a CE is considered equal to the action decision period,  $\Delta$ , also used as the slot duration. The probability to process one job in one slot is equal to 1, while the probability of transmitting one job in one slot from UAV 1 to UAV 2 is  $p_{TX} = \Delta/t_T$ .

To derive the transition probability matrix of the whole system constituted by UAV 1 and UAV 2 and the transmission queue  $T$ , let consider again two generic states  $\underline{s}'_{\Sigma}$  and  $\underline{s}''_{\Sigma}$  defined as:

- $\underline{s}'_{\Sigma} = (s'_{Z}, s'_{Q}, s'_{T}) = \underline{s}^{(\Sigma)}(n-1)$  at the slot  $n-1$
- $\underline{s}''_{\Sigma} = (s''_{Z}, s''_{Q}, s''_{T}) = \underline{s}^{(\Sigma)}(n)$  at the slot  $n$

Considering the Fig. 3.32, the following event sequence are performed to transit from the slot  $n-1$  to the slot  $n$ :

- Decision of the new value  $a = (b_1, b_2, \sigma)$ , with  $b_i \in [1, L]$ , according to the best policy achieved by RL, based on the value of the current system state  $\underline{s}'_{\Sigma} = \underline{s}^{(\Sigma)}(n-1)$ ;
- Update of the zone state, with the transition from  $\underline{s}^{(Z)}(n-1)$  to  $\underline{s}^{(Z)}(n)$ ;
- Dequeue of some jobs from the queues  $Q_1$  and  $Q_2$ , according to the number  $b_i$  of CPUs that have worked in the slot  $n$  in the  $i$ -th UAV;

- The arrival of some jobs,  $\underline{Z}(n) = (\omega_1, \omega_2)$ , according to the SBBP processes, whose underlying Markov chain states,  $S^{(Z_1)}(n)$  and  $S^{(Z_2)}(n)$ , have been updated in the step 2;
- Enqueue of some jobs in queue  $Q_1$  equal to  $\min\{\sigma, \omega_1\}$ , where  $\sigma$  has been decided at step 1. These jobs will suffer a delay due to queue  $Q_1$ ;
- Dequeue of one job from the Transmission queue  $T$  (this occurs with the job departure probability), and enqueue it in queue  $Q_2$  of UAV 2 after the jobs arrived from area 2;
- Enqueue in queue  $T$  of the jobs arrived from area 1 and to be offloaded. These jobs will suffer a delay that is the sum of the delay in  $T$  and the delay in  $Q_2$ ;
- Update of the system state from  $\underline{S}^{(\Sigma)}(n-1)$  to  $\underline{S}^{(\Sigma)}(n)$ , according to the evolution of each of its components, as described in all the previous steps.

The generic element of the transition probability matrix can be defined as follows:

$$P_{[\underline{s}'_{\Sigma}, \underline{s}''_{\Sigma}]}^{(\Sigma|a)} = P_{[\underline{s}'_Z, \underline{s}''_Z]}^{(Z)} \cdot P_{[(\underline{s}'_Q, \underline{s}'_T), (\underline{s}''_Q, \underline{s}''_T)]}^{(Q,T|a)}(\underline{s}''_Z) \quad (3.61)$$

Assuming that the behavior of each area is statistically independent of the behavior of the other one, the generic element of the  $P^{(Z)}$  matrix is given by:

$$P_{[\underline{s}'_Z, \underline{s}''_Z]}^{(Z)} = P_{[\underline{s}'_{Z_1}, \underline{s}''_{Z_1}]}^{(Z_1)} \cdot P_{[\underline{s}'_{Z_2}, \underline{s}''_{Z_2}]}^{(Z_2)} \quad (3.62)$$

The matrix  $P^{(Q,T|a)}(\underline{s}''_Z)$  models the behavior of the three queues  $Q_1$ ,  $Q_2$ , and  $T$ . Also in this scenario, it depends on the arrival process, characterized by the states of the underlying Markov chain of the areas, which determine the number of job arrivals in the queues and the action  $a$  for each starting state. Its generic element is:

$$\begin{aligned} & P_{[(\underline{s}'_Q, \underline{s}'_T), (\underline{s}''_Q, \underline{s}''_T)]}^{(Q,T|a)}(\underline{s}''_Z) = \\ & = Pr \left\{ \underline{S}^{(Q)}(n) = \underline{s}''_Q, S^{(T)}(n) = \underline{s}''_T \mid \underline{S}^{(Q)}(n-1) = \underline{s}'_Q, S^{(T)}(n-1) = \underline{s}'_T, A(n) = a \right\} \end{aligned} \quad (3.63)$$

To evaluate this probability, let apply the total probability theorem to the number of possible arrivals from the monitored areas,  $\omega_1$  and  $\omega_2$ :



$$\begin{aligned}
 P_{[(s'_Q, s'_T), (s''_Q, s''_T)]}^{(Q, T|a)}(\underline{s}''_Z) &= \sum_{\forall \omega_1 \in \Omega^{(Z)} \forall \omega_2 \in \Omega^{(Z)}} \sum_{[s''_{Z_1}, \omega_1]} B^{(Z_1)} \cdot B^{(Z_2)}_{[s''_{Z_2}, \omega_2]}. \\
 Pr\{\underline{S}^{(Q)}(n) = \underline{s}''_Q, S^{(T)}(n) = s''_T | \underline{S}^{(Q)}(n-1) = \underline{s}'_Q, S^{(T)}(n-1) = s'_T, \\
 A(n) = a, Z_1(n) = \omega_1, Z_2(n) = \omega_2\} & \quad (3.64)
 \end{aligned}$$

The probability term of the previous equation can be evaluated by considering that, choosing the slot duration equal to the mean job service time on a UAV CE,  $b_1$  jobs will be served in queue  $Q_1$  and in queue  $Q_2$ . Instead, the number of jobs that leave the transmission queue  $T$  depends on the job size and the throughput of the connection link from UAV 1 to UAV 2. Now, applying again the theorem of total probability to the number of jobs that are transmitted from the transmission queue, indicated as  $d_T$ , with  $d_T \in \{0, 1\}$ , the probability term in the previous equation can be written as:

$$\begin{aligned}
 Pr\{\underline{S}^{(Q)}(n) = \underline{s}''_Q, S^{(T)}(n) = s''_T | \underline{S}^{(Q)}(n-1) = \underline{s}'_Q, S^{(T)}(n-1) = s'_T, \\
 A(n) = a, Z_1(n) = \omega_1, Z_2(n) = \omega_2\} &= \sum_{d_T=0}^1 Pr\{d_T\} \cdot f^{(Q_1)}(s'_{Q_1}, s''_{Q_1}, b_1, \omega_1, \sigma) \cdot \\
 \cdot f^{(Q_2)}(s'_{Q_2}, s''_{Q_2}, s'_T, b_2, \omega_2, d_T) \cdot f^{(T)}(s'_T, s''_T, \omega_1, d_T, \sigma) & \quad (3.65)
 \end{aligned}$$

The term  $Pr\{d_T\}$  represents the probability that jobs leave queue  $T$  because transmitted and is defined as:

$$Pr\{d_T\} = \begin{cases} p_{TX} & \text{if } d_T=1 \\ 1 - p_{TX} & \text{if } d_T=0 \end{cases} \quad (3.66)$$

$f^{(Q_1)}(\cdot)$ ,  $f^{(Q_2)}(\cdot)$  and  $f^{(T)}(\cdot)$  are functions providing the one-slot evolution probabilities of the two UAV queues and the transmission queue, respectively. About the first function, it is defined as:

$$f^{(Q_1)}(\cdot) = \begin{cases} 1 & \text{if } s''_{Q_1} = \min\{\max\{s'_{Q_1} - b_1, 0\} + \min\{\sigma, \omega_1\}, K\} \\ 0 & \text{otherwise} \end{cases} \quad (3.67)$$

To calculate the second function, let consider that the number of jobs in  $Q_2$ , after arrivals and departures, is the sum of the  $\omega_2$  jobs arriving from area 2 and

the jobs arriving from queue  $T$ , which is  $d_T$  if at least jobs are  $d_T$  present in  $T$ . Then, the derivation of the final number of jobs in  $Q_2$  at the end of the slot has to account for the comparison with the maximum queue size  $K$ , and the number of departures,  $b_2$ . So  $f^{(Q_2)}(\cdot)$  is equal to:

$$f^{(Q_2)}(\cdot) = \begin{cases} 1 & \text{if } s''_{Q_2} = \min\{\max\{s'_{Q_2} - b_2, 0\} + \omega_2 + \min\{s'_T, d_T\}, K\} \\ 0 & \text{otherwise} \end{cases} \quad (3.68)$$

Finally, the function  $f^{(T)}(\cdot)$  can be easily derived considering that a number of jobs equal to  $(\omega_1 - \min\{\sigma, \omega_1\})$  enter queue  $T$ , while  $d_T$  jobs are dequeued:

$$f^{(T)}(\cdot) = \begin{cases} 1 & \text{if } s''_T = \min\{\max\{s'_T - d_T, 0\} + (\omega_1 - \min\{\sigma, \omega_1\}), M\} \\ 0 & \text{otherwise} \end{cases} \quad (3.69)$$

As in the previous case, let define the expected value of the immediate reward by weighing the three key parameters characterizing the system behavior, that is, power consumption, job loss probability, and delay. More in details:

$$\Psi_{[s'_\Sigma, s''_\Sigma]}^{(\Sigma|a)} = -c_1 \frac{\bar{\xi}(a) - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} - c_2 \frac{\bar{\lambda}(\underline{s}'_\Sigma, s''_{Z_2}, a) - \bar{\lambda}_{MIN}}{\bar{\lambda}_{MAX} - \bar{\lambda}_{MIN}} - c_3 \frac{\bar{\delta}(\underline{s}''_\Sigma, a) - \bar{\delta}_{MIN}}{\bar{\delta}_{MAX} - \bar{\delta}_{MIN}} \quad (3.70)$$

the meaning of the three terms is the same as in the previous case, but with some changes due to the presence of area 2 and queue  $T$ - In particular:

- in the first term, regarding the penalty for power consumption, due to the definition of  $a = (b_1, b_2, \sigma)$ ,  $\bar{\xi}(a)$  is defined like:

$$\bar{\xi}(a) = (b_1 + b_2) \cdot \xi_{\mu P} \quad (3.71)$$

- the second term is the penalty related to the job loss. It is calculated as the  $\log_{10}$  of the expected number of the fraction of jobs lost in the considered transition,  $X^{(\Sigma)}(n)$ , over the mean number of arrivals, in the same slot,  $Z^{(\Sigma)}(n)$ :

$$\bar{\lambda}(\underline{s}'_\Sigma, s''_{Z_2}, a) = \log_{10} \frac{X^{(\Sigma)}(n)}{Z^{(\Sigma)}(n)} \quad (3.72)$$

where

$$\begin{aligned}
 Z^{(\Sigma)}(n) &= E\{Z(n)|\underline{S}^{(Z)} = \underline{s}'_Z\} = \\
 &= E\{Z_1|S^{(Z_1)} = s'_{Z_1}\} + E\{Z_2|S^{(Z_2)} = s'_{Z_2}\} = \\
 &= \sum_{\forall \omega_1 \in \Omega^{(Z_1)}} \omega_1 \cdot B_{[s''_{Z_1}, \omega_1]}^{(Z_1)} + \sum_{\forall \omega_2 \in \Omega^{(Z_2)}} \omega_2 \cdot B_{[s''_{Z_2}, \omega_2]}^{(Z_2)}
 \end{aligned} \tag{3.73}$$

while  $X^{(\Sigma)}(n)$  is the sum of jobs lost in  $Q_1, Q_2$  and  $T$ :

$$\begin{aligned}
 X^{(\Sigma)}(n) &= \sum_{\forall \omega_1 \in \Omega^{(Z)}} B_{[s''_{Z_1}, \omega_1]}^{(Z_1)} \sum_{\forall \omega_2 \in \Omega^{(Z)}} B_{[s''_{Z_2}, \omega_2]}^{(Z_2)} \cdot \\
 &\cdot [\max\{\max\{s'_{Q_1} - b_1, 0\} + \sigma - K, 0\} + \\
 &+ \sum_{d_T=0}^1 \max\{\max\{s'_T - d_T, 0\} + (\omega_1 - \sigma) - M, 0\} + \\
 &+ \max\{\max\{s'_{Q_2} - b_2, 0\} + \omega_2 + \min\{d_T, s'_T\} - K, 0\}]
 \end{aligned} \tag{3.74}$$

- the third element regards the delay suffered in the system queues. As already said, the jobs arrived at UAV 1 can suffer either the delay in queue  $Q_1$ , if not offloaded, or the sum of the delays suffered in queue  $T$  and in queue  $Q_2$ , where also jobs coming from area 2 are buffered, which delay has to be considered too. These three delays are weighed by the percentage of traffic steered through these paths. To be more conservative, and assuming that conditions of those queues remain constant in the future, the SC, for its decision, considers the delay of the last job of the burst that is enqueued at each slot. So, applying the total probability theorem to the number of arrived jobs from both the areas, this term can be defined as:

$$\begin{aligned}
 \bar{\delta}(\underline{s}'_{\Sigma}, \underline{s}''_{\Sigma}, a) &= \sum_{\forall \omega_1 \in \Omega^{(Z)}} B_{[s''_{Z_1}, \omega_1]}^{(Z_1)} \sum_{\forall \omega_2 \in \Omega^{(Z)}} B_{[s''_{Z_2}, \omega_2]}^{(Z_2)} \cdot \frac{1}{\omega_1 + \omega_2} \cdot \\
 &\cdot [\min\{\sigma, \omega_1\} \cdot \bar{\delta}_{1,nOL} + (\omega_1 - \min\{\sigma, \omega_1\}) \cdot \bar{\delta}_{1,OL} + \omega_2 \cdot \bar{\delta}_2]
 \end{aligned} \tag{3.75}$$

where  $\bar{\delta}_{1,nOL}$  is the expected delay suffered by the last job enqueued in the  $Q_1$ ,  $\bar{\delta}_{1,OL}$  is the expected delay suffered by the last job offloaded along the path  $T$  and  $Q_2$ , while  $\bar{\delta}_2$  is the expected delay suffered by the last job enqueued  $Q_2$  coming from area 2.

The generic element of the steady-state probability array can be defined as in (3.50) and (3.51).

Let derive the three main performance parameters characterizing the reward function. The mean power consumption is:

$$\bar{\xi} = \xi_{\mu P} \cdot \sum_{\forall \underline{s}_{\Sigma} \in \mathfrak{S}(\Sigma)} [b_1(\underline{s}_{\Sigma}) + b_2(\underline{s}_{\Sigma})] \cdot \pi_{[\underline{s}_{\Sigma}]}^{(\Sigma)} \quad (3.76)$$

where  $b_1(\underline{s}_{\Sigma})$  and  $+b_2(\underline{s}_{\Sigma})$  are the number of CEs activated by Sc in the UAV1 and UAV2.

The mean delays suffered in the three queues can be calculated using Little law. It was already defined in the previous case, considering only a queue in the UAV1. In this case, it has to be calculated for  $Q_1$ ,  $Q_2$  and  $T$ . More specifically, for  $Q_1$  it is:

$$\bar{\delta}_{Q_1} = \frac{\bar{N}_{Q_1}}{\bar{\Lambda}_{Q_1}} \quad (3.77)$$

with:

$$\bar{N}_{Q_1} = \sum_{\forall \underline{s}_{\Sigma} \in \mathfrak{S}(\Sigma)} s_{Q_1} \cdot \pi_{[\underline{s}_{\Sigma}]}^{(\Sigma)} \quad (3.78)$$

$$\bar{\Lambda}_{Q_1} = \sum_{\forall \underline{s}'_{\Sigma} \in \mathfrak{S}(\Sigma)} \sum_{\forall \underline{s}''_{Z_1} \in \mathfrak{S}(Z)} \sum_{\forall \omega_1 \in \mathfrak{S}(V)} \min\{\sigma, \omega_1, K - s'_{Q_1}\} \cdot B_{[s''_{Z_1}, \omega_1]}^{(Z_1)} \cdot P_{[s'_{Z_1}, s''_{Z_1}]}^{(Z_1)} \cdot \pi_{[\underline{s}'_{\Sigma}]}^{(\Sigma)} \quad (3.79)$$

For the queue  $T$ :

$$\bar{\delta}_T = \frac{\bar{N}_T}{\bar{\Lambda}_T} \quad (3.80)$$

with:

$$\bar{N}_T = \sum_{\forall \underline{s}_{\Sigma} \in \mathfrak{S}(\Sigma)} s_T \cdot \pi_{[\underline{s}_{\Sigma}]}^{(\Sigma)} \quad (3.81)$$

$$\bar{\Lambda}_T = \sum_{\forall \underline{s}'_{\Sigma} \in \mathfrak{S}(\Sigma)} \sum_{\forall \underline{s}''_{Z_1} \in \mathfrak{S}(Z)} \sum_{\forall \omega_1 \in \mathfrak{S}(V)} \min\{\max(\omega_1 - \sigma, 0), M - s'_T\} \cdot B_{[s''_{Z_1}, \omega_1]}^{(Z_1)} \cdot P_{[s'_{Z_1}, s''_{Z_1}]}^{(Z_1)} \cdot \pi_{[\underline{s}'_{\Sigma}]}^{(\Sigma)} \quad (3.82)$$

For the queue  $Q_2$ , as already said, the number of job arrivals in one slot is  $\omega_2$  from the area 2 plus one if one job arrives from queue  $T$ , with probability  $Pr\{d_T\}$ :

$$\bar{\delta}_{Q_2} = \frac{\bar{N}_{Q_2}}{\bar{\Lambda}_{Q_2}} \quad (3.83)$$

with:

$$\bar{N}_{Q_2} = \sum_{\forall \underline{s}_{\Sigma} \in \mathfrak{S}(\Sigma)} s_{Q_2} \cdot \pi_{[\underline{s}_{\Sigma}]}^{(\Sigma)} \quad (3.84)$$

$$\begin{aligned} \bar{\Lambda}_{Q_2} = & \sum_{\forall \underline{s}'_{\Sigma} \in \mathfrak{S}(\Sigma)} \sum_{\forall \underline{s}''_{Z_2} \in \mathfrak{S}(Z)} \sum_{\forall \omega_2 \in \mathfrak{S}(V)} \sum_{d_t=0}^1 Pr\{d_T\} \min\{\omega_2 + \min(d_T, s'_T), K - s'_{Q_2}\} \cdot \\ & \cdot B_{[s''_{Z_2}, \omega_2]}^{(Z_2)} \cdot P_{[s'_{Z_2}, s''_{Z_2}]}^{(Z_2)} \cdot \pi_{[s'_{\Sigma}]}^{(\Sigma)} \end{aligned} \quad (3.85)$$

Finally, the per-slot loss probability can be calculated by averaging the number of lost jobs when the system moves from the state  $\underline{s}'_{\Sigma}$  to the state  $\underline{s}''_{\Sigma}$  and the SC takes the action  $a$ :

$$\bar{\lambda} = \sum_{\forall \underline{s}'_{\Sigma} \in \mathfrak{S}(\Sigma)} \sum_{\forall \underline{s}''_{Z} \in \mathfrak{S}(Z)} \bar{\lambda}(\underline{s}'_{\Sigma}, \underline{s}''_{Z}, a) \cdot P_{[\underline{s}'_{\Sigma}, \underline{s}''_{Z}]}^{(\Sigma|a)} \cdot \pi_{[\underline{s}'_{\Sigma}]}^{(\Sigma)} \quad (3.86)$$

Now, it is possible to consider a case study to apply the proposed framework and evaluate some numerical results, aimed at both showing how the proposed framework behaves and the gain achieved in respect to the state-of-the-art. A FANET of rotary-wing UAVs is considered, with a power consumption of 150 W on average. Let assume that CEs installed onboard each UAV are Computer Processor Units (CPUs) Intel® Core™ i7-10510U Processors, 8MB Cache, 4.80 GHz, and 32GB DDR4 SO-DIMM ram. The cases in which  $L = 3$  and  $L = 4$  CPUs are available onboard of UAV to provide the slice extension service is consider. The system power consumption for computation with no active CPUs is  $\xi_S = 10W$ , while each active CPU absorbs  $\xi_{\mu P} = 25W$ . So, the instantaneous power consumption when  $b(n)$  CPUs are active is:

$$\xi_{Proc}(n) = \xi_S + b(n) \cdot \xi_{\mu P} \quad (3.87)$$

Each UAV has a job queue that can contain, at most,  $K = 15$  jobs, and a transmission queue where at most  $M = 5$  jobs can be enqueued to wait for transmission. We consider jobs having, on average, a size of 7 MB, and requiring a mean processing time of  $\Delta = 300$  ms, also chosen as the slot duration.

The same  $Q$  and  $B$  matrix of the previous case has been used as input of the problem. The presented analysis is done against the transmission bitrate, in the following also referred to as job offloading bitrate, on the radio channel from UAV 1 to UAV 2, ranging in the interval [28, 180] Mbit/s. For example, let assume that four UAVs are used to monitor an area of 1,000 square meters, and that the area covered by each UAV is in the HA state for the 6% of the time ( $p_H = 0.06$ ), while it is in LA state in the remaining time ( $p_L = 0.94$ ). Moreover, let assume that each UAV transmits with a power of 0.5 W using a carrier frequency of 2

GHz on a channel characterized by noise power spectral  $N_0 = -170$  dBm/Hz with an additional path loss to free space in loss of 3 dB. In this case, the job offloading bitrate ranges in the considered interval [28, 180] Mbit/s by using a channel bandwidth ranging between 975 kHz and 6.2 MHz.

To evaluate the gain introduced by the proposed strategy, in the following referred to as “RL ALL”, it is compared with the following three strategies:

- “No RL”: all the available CPUs are maintained on, and offload is not used, so UAV 1 is not helped by UAV 2. This represents the state-of-art strategy;
- “RL CPU”: RL is applied to decide the number of CPUs, while offload is not applied;
- “RL OL”: RL is applied to decide how many jobs have to be offloaded in each slot, whilst all the available CPUs are maintained on.

Moreover, two different scenarios are considered, each characterized by different importance assigned to power saving, loss probability, and delay. This is achieved by considering two different configurations  $C = (c_1, c_2, c_3)$ . More specifically, the cases  $C1 = (1, 1, 1)$ , where all the parameters are weighed in the same way, and  $C2 = (1, 3, 3)$ , where loss and delay take a higher priority than power consumption, were considered. Finally, a discount factor  $\gamma = 0.8$  was used by the SC to apply RL.

First, results in the case of  $L = 3$  available CPUs, that is when UAV 1 is strongly stressed, are presented. Loss probability, mean delay and mean number of used CPUs are shown in Figs. 3.33, 3.34 and 3.35 for all the three queueing systems, as a function of the job offloading bitrate.

Referring to Fig. 3.33b, some curves listed in the legend are not present since they have negligible values, less than the lower limit of the y-axis. Moreover, curves related to the cases not applying RL on the number of active CPUs are not shown in Fig. 3.35 since trivial, given that they use all the 3 available CPUs.

In Fig. 3.33, the reference cases “RL CPU” and “No RL”, are constant against the job offloading bitrate since they do not use offload. The loss probability achieved with the other two policies, that is, “RL OL” and “RL ALL”, rapidly decreases in  $Q_1$  w.r.t. the offloading bitrate, and is lower for the C2 configuration, since this configuration privileges loss and delay. Instead, in queue  $Q_2$ , loss probability increases for these two last policies, except for the “RL ALL” policy with C2 configuration. Increasing is motivated by the fact that the higher the offloading bitrate, the higher the number of offloaded jobs, and therefore the higher the  $Q_2$

# CHAPTER 3. MANAGEMENT AND ORCHESTRATION OF NETWORK SLICES

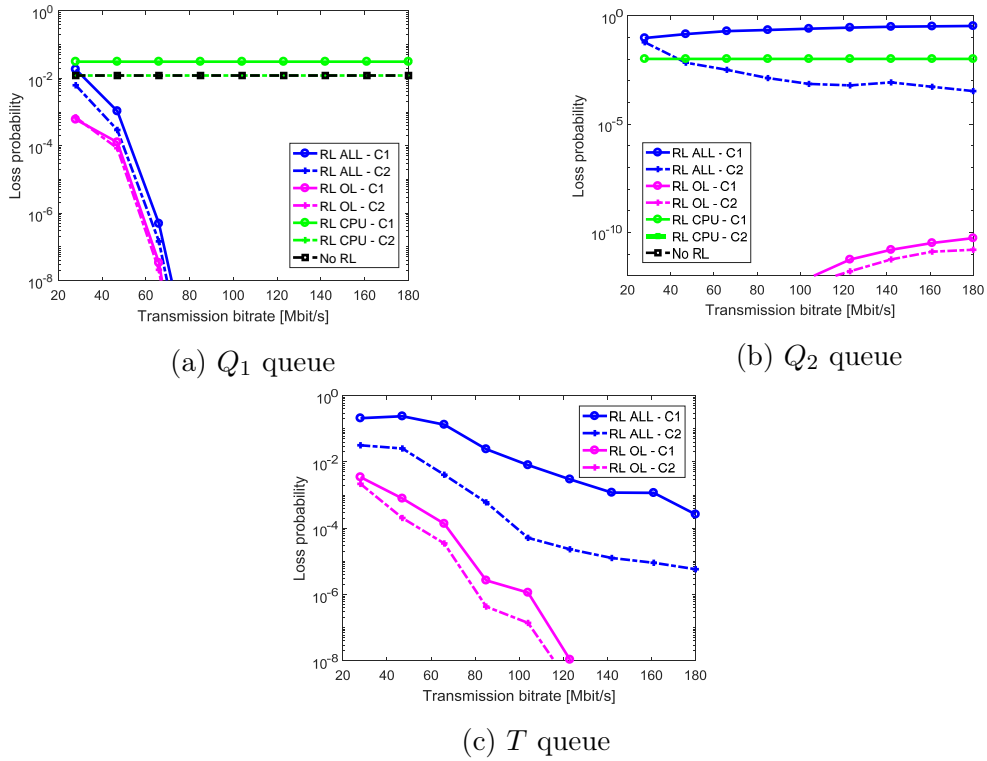


Figure 3.33: UAVs cooperation and RL. Loss probability for  $L = 3$  available CPUs

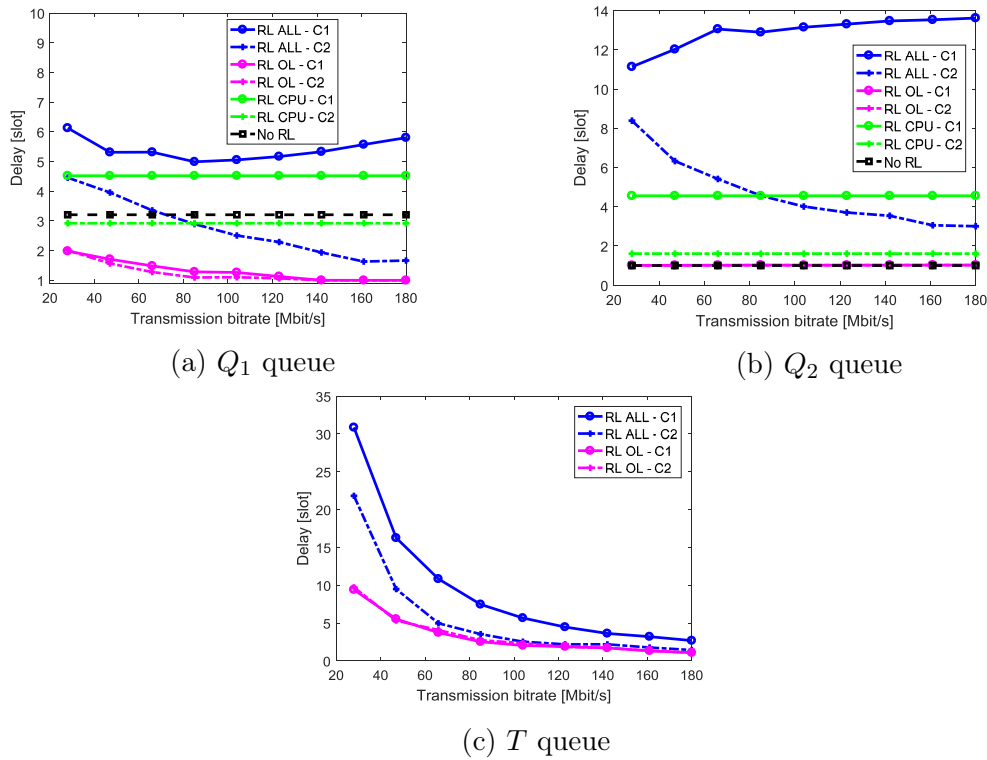


Figure 3.34: UAVs cooperation and RL. Mean delay for  $L = 3$  available CPUs

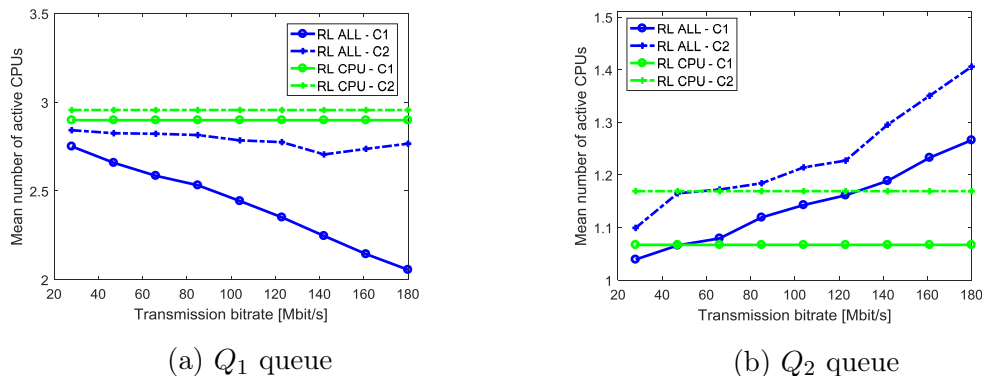


Figure 3.35: UAVs cooperation and RL. Mean number of active CPUs for  $L = 3$  available CPUs

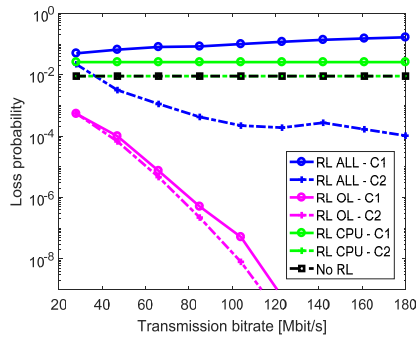
input rate; on the other hand, the decreasing behavior for the C2 configuration is because, to privilege loss probability even when UAV 2 receives more offload from UAV 1, the SC decides to use a higher number of CPUs, as shown in Fig. 3.35b. The strategy “RL CPU” suffers from a high amount of losses with the C1 configuration in both  $Q_1$  and  $Q_2$  since the SC aims at also saving energy, as shown in Fig. 3.35b. again, characterized by the lowest mean number of active CPUs. This is not true only in  $Q_2$  for job offloading bitrates less than 45 Mbit/s because, in this case, offload is not so much useful for the “RL ALL” policy with the C1 configuration, behaving the offloading link as a system bottleneck. This last statement is also supported by the curves in Figs. 3.33.c and 3.34.c, where high values of loss probability and delay in queue  $T$  are obtained for low values of job offloading bitrate, especially in cases of C1 configuration, i.e. when the privileged target is energy saving.

As far as the mean delay is concerned, the best performance in  $Q_1$  is achieved with the “RL OL” policy, since it uses job offload while maintaining all the CPUs active. Instead, the “RL CPU” policy with the C1 configuration performs worse than “No RL” since it aims at saving some energy by switching off some CPU.

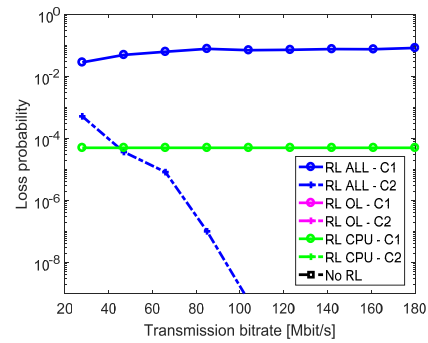
Now, to also evaluate the impact of the number of available CPUs on system performance and flight duration, the case of  $L = 4$  available CPUs is included. Figs. 3.36 and 3.37 present the overall loss probability and mean delay for all the considered strategies. Instead, Fig. 3.38 shows the power consumption gain of the “RL ALL” strategy as compared with the other ones for both the configurations C1 and C2. In these figures, let appreciate the higher flexibility of the proposed “RL ALL” policy as compared to the other ones. Indeed, the “RL ALL” policy introduces more losses and higher delays when these parameters are not of primary importance (C1 configuration), but in this case, it is able to obtain a positive



## CHAPTER 3. MANAGEMENT AND ORCHESTRATION OF NETWORK SLICES

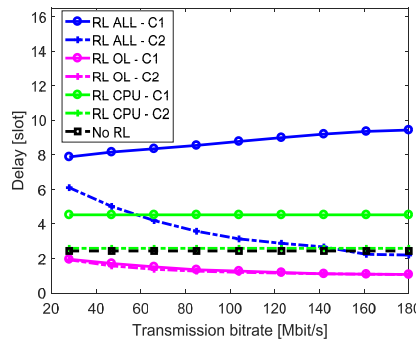


(a)  $L = 3$  available CPUs

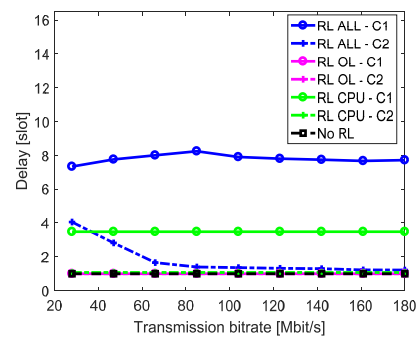


(b)  $L = 4$  available CPUs

Figure 3.36: UAVs cooperation and RL. Loss probability (on both non-offloaded and offloaded jobs)

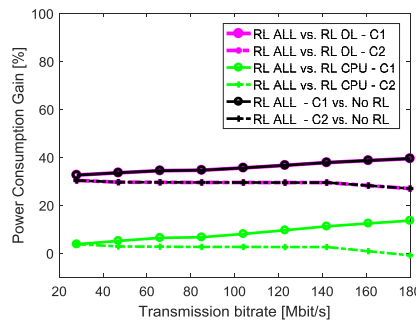


(a)  $L = 3$  available CPUs

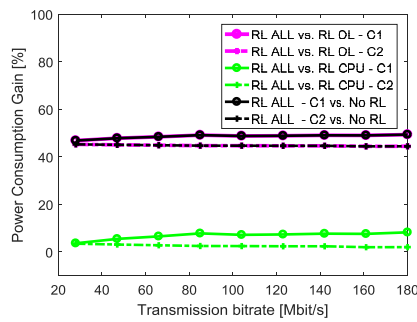


(b)  $L = 4$  available CPUs

Figure 3.37: UAVs cooperation and RL. Mean delay



(a)  $L = 3$  available CPUs



(b)  $L = 4$  available CPUs

Figure 3.38: UAVs cooperation and RL. Power consumption gain

power consumption gain with high values. On the contrary, if loss probability and delay are more important, “RL ALL” is able to achieve very low values for them, although with lower power consumption gain than all the other policies, even if this gain remains positive. Of course, the highest power consumption gain for “RL ALL” is obtained against the “RL OL” and the “No RL” policies, since these maintain all the available CPUs active. Moreover, this gain is higher when

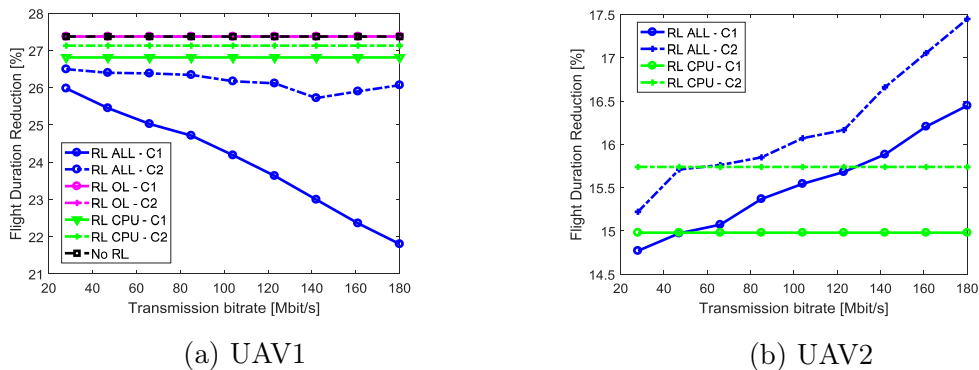


Figure 3.39: UAVs cooperation and RL. Reduction percentage of flight duration for  $L = 3$

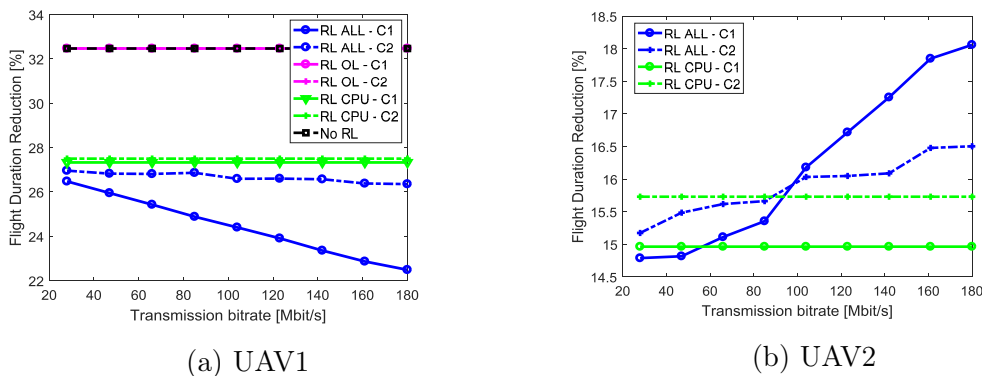


Figure 3.40: UAVs cooperation and RL. Reduction percentage of flight duration for  $L = 4$

a higher number of CPUs is available onboard UAV ( $L = 4$  rather than  $L = 3$ ).

Finally, Figs. 3.39 and 3.40 present the impact of the considered four strategies on the flight duration, for both the UAVs and in both the cases of  $L = 3$  and  $L = 4$ . To this purpose, the percentage of flight duration reduction caused by supplying CPUs is compared to the power consumption needed to supply UAV engines. Both the “RL CPU” and “No RL” policies are considered as a reference because they maintain all available CPUs active, and this is the reason why their curves are constant w.r.t. the job offloading bitrate. They have not been reported for UAV 2 since obtained values are the same as UAV 1. From these figures let observe how the strategies using RL to decide the number of active CPUs outperform the previous ones. Moreover, when the “RL ALL” strategy is applied, the flight duration of UAV 1 is maximized, and this is more evident for high values of job offloading bitrate and, of course, when the C1 configuration is used. On the other side, the reduction of UAV 2 flight duration due to CPUs is very small, even if increasing, as expected, with the job offloading bitrate. The same

considerations can be done with  $L = 4$  but, of course, with a higher impact on the flight duration reduction.

In conclusion, the experimental results show the flexibility of “RL ALL” and its ability to satisfy network slice requirements by varying configuration parameters. Thus, the SC using “RL ALL” can adapt its behavior to the application scenario at runtime with great flexibility.

# Chapter 4

## Network slicing for vertical applications

This chapter introduces the use of network slicing for vertical applications. In particular, two different scenarios are presented: the first concerns the Tactile Internet network slice and an implementation of its main component, the TSE (Section 4.1), while the second refers to the Vehicular network slice, in which a multi-offloading strategy between vehicles and MEC server is proposed (Section 4.2).

### 4.1 The TSE for Tactile Internet

The network slice providing guarantees of e2e delay not greater than 1 ms is called *Tactile Internet*. Due to this stringent requirement, it is necessary to insert a component, at the edge of the network, that can compensate for any malfunctions of the core network, in order to guarantee the requested requirements to the applications that use this slice. The *Tactile Support Engine* (TSE) has precisely this goal, thanks also to the Artificial Intelligence (AI) techniques that are implemented within it.

Despite the importance of the TSE, the literature lacks a definition of the architecture of this component and how it integrates with the rest of the actors of the Tactile Internet network slice. For this reason, in the paper [59] submitted to the Special Issue on Network Intelligence (Computer Communication), a TSE architecture was proposed and the integration with the other components presented.

In Subsection 4.1.1, the general architecture of the Tactile Internet network slice is described, while Subsection 4.1.2 proposes a structure of the TSE. It is then

applied to a real scenario, to obtain measures to evaluate its performance.

### 4.1.1 Tactile Internet Architecture

The term Tactile Internet was first coined by G. P. Fettweis in early 2014 [60] to provide services with interaction latency typically required for tactile steering and control of real and virtual objects without creating cyber-sickness. More specifically, its definition agreed within the IEEE 1918.1 WG [61], is: “A network (or network of networks) for remotely accessing, perceiving, manipulating, or controlling real or virtual objects or processes in perceived real-time by humans or machines.”

The idea is to provide applications with real-time interaction of human beings, with response time to the environment in the order of milliseconds. Therefore, the Tactile Internet aims at defining a new human-machine interaction where the network provides a physiological latency of human beings to build real-time interactive systems. The primary goal is to provide the necessary infrastructure, given that not only content does need to be transported in the future Internet, but also control information with a maximum round-trip time in which a sensor reads information and a connected system reacts with actuators within few milliseconds. Another key challenge of the Tactile Internet is to be able to provide carrier-grade access reliability and robustness, e.g., an uptime of the system of so-called seven nines (99.99999 %), i.e., a failure rate of  $10^{-7}$  [62].

In [63] it has been observed that, considering the propagation delay due to the speed of light, the maximum distance between a steering and control server and the point of tactile interaction by the users is of 150-750 km (considering an e2e delay between 1-5 milliseconds). However, let us note that, if sensors produce big data, their transmission introduces an additional delay that depends on the link capacity where these data have to be transmitted. For this reason, and due to the numerous applications ranging from tele-operation, automotive, and immersive virtual reality to the Internet of drones, interpersonal communications, live haptic-enabled broadcast and cooperative automated driving [64], a lot of research activities and investments from industrial and academic entities have been devoted to the study of Tactile Internet.

The general architecture for a Tactile Internet network slice is shown in Fig. 4.1. It is constituted by three different domains:

- the *Master Domain*, consisting of an operator, either human or machine, and a human-system interface (HSI). This interface is a master robot/controlling

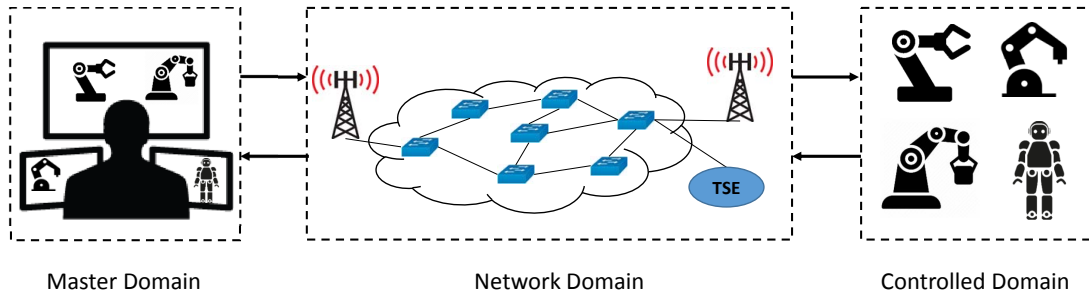


Figure 4.1: Tactile Internet scenario

device (e.g. a haptic device) that converts the operator’s input to a “Tactile Internet input” through various coding techniques;

- the *Controlled Domain*, consisting of several devices (e.g. robots or objects) located in a real environment, constituted by several components, each characterized by specific parameters. These components are controlled directly by the Master Domain through various command signals for interaction in a remote environment;
- the *Network Domain*, which is the telecommunications infrastructure connecting the aforementioned two domains. It provides the Tactile Internet network slice.

The role of the Network Domain is crucial in a scenario with very low latency requirements. A delay in the transmission of the state of controlled devices from the Controlled Domain to the Master Domain would cause an incorrect synchronization between the state of device components in the real environment and what the operator perceives through the display. In the same way, a delay in the transmission of control values (e.g. the current to be applied to a motor of a robot) in the opposite direction would cause a control of the remote device not synchronized with the operator’s decisions.

For this reason, the presence of the TSE is fundamental. Its goal is to enhance the capacity of the network by providing the required level of QoS to the end-users. In other words, when the network presents some time-limited problems, the TSE intervenes to substitute/integrate information generated by the Master Domain that are not arrived in time, so that the Controlled Domain remains unaware of these problems. The behavior of the TSE is based on techniques of AI [65]. The task of the TSE is enabled by the application of the MEC paradigm, allowing cloud computing proximity to host the tools of AI in the service of mobile devices.

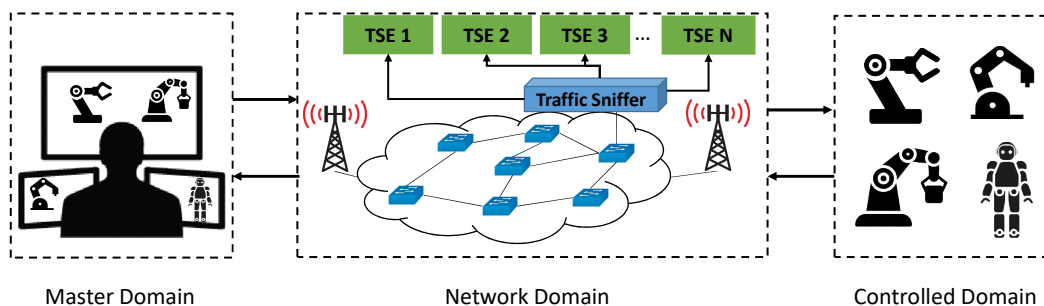


Figure 4.2: TSE Reference System

### 4.1.2 TSE Implementation

Starting from Fig. 4.1, the proposed system is shown in Fig. 4.2. Two VNFs running at the edge node connected to the Controlled Domain site were considered:

- the *Traffic Sniffer* that collects data from the network. It uses the *pcap* library [45] already described in Chapter 3 to capture packets from the network and analyze them;
- the *TSE* that implements AI at the network edge to close the loop of the Controlled domain when the Network domain is not respecting the declared maximum tolerated end-to-end delay; this is achieved by applying AI on the data received by the Traffic Sniffer.

As seen in Fig. 4.2, there is one Traffic Sniffer for each edge node of the Network Domain, and one or more TSEs, depending on the number of controlled devices placed in the Controlled Domain that access through that edge node. The traffic flow transmitted by each controlled device is characterized by an ID that allows the Traffic Sniffer to know the TSE which has to receive that information.

The TSE, thanks to its position very close to the Controlled Domain, always receives, through the Traffic Sniffer VNF, data generated by the devices in the Controlled Domain. When the response from the Master Domain is not received from the TSE by a given  $\Delta t$  time, which is an application-based time constraint (e.g. it is equal to 5 *ms* for remote gaming), it forecasts the missing values and sends them directly to the Controlled Domain.

About the Network domain, its behavior is classified into two states: 1) *Good state*, when the round-trip time, also including the processing time at the Master Domain, is less than the required threshold  $\Delta t$ ; 2) *Bad state*, when this latency requirement is not satisfied, or some data are lost. The behavior of the Tactile Internet components in the two states is sketched in Fig. 4.3 and Fig. 4.4.

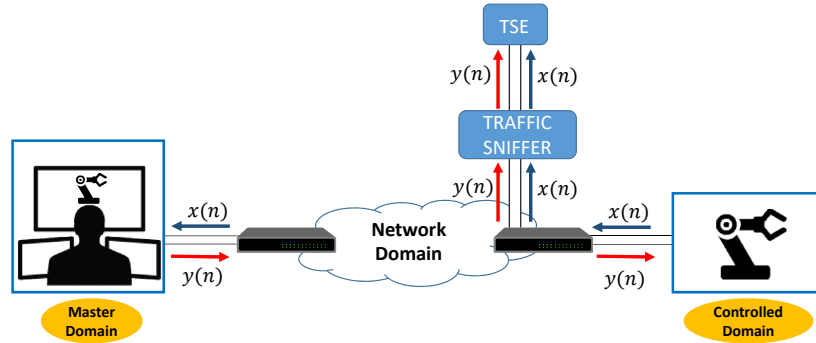


Figure 4.3: TSE implementation. Network in Good state

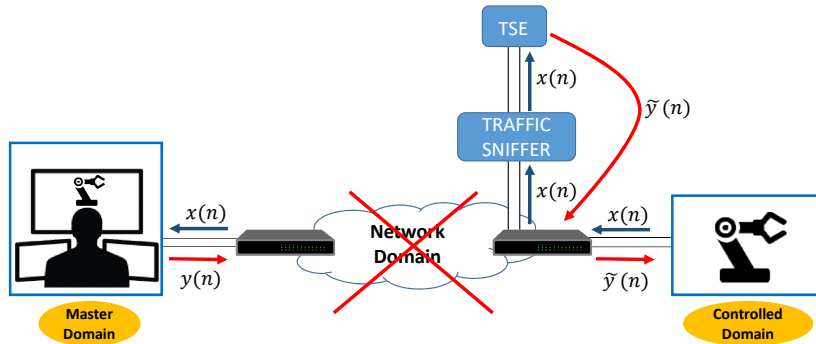


Figure 4.4: TSE implementation. Network in Bad state

At every slot  $n$ , the Controlled Domain sends an information element with a value  $x(n)$  to the Master Domain, regarding some parameters of the controlled device. When the edge access node (for instance, an SDN switch) receives this information, it forwards this element to both the Master Domain through the network and to the Traffic Sniffer. The latter reads the ID of the device that has generated the information element, and forwards the  $x(n)$  value to the TSE controlling that device. The TSE receives this information and stores it. On the other side of the network, the user application in the Master Domain receives  $x(n)$ , and a human operator can see the controlled device and its state on a display. Based on the application, the human operator can remotely control a particular aspect of the device using a haptic device, by sending a reply value  $y(n)$  to the remote Controlled Domain. If the network is working in good condition (Good state), i.e. there are no delay or loss problems, this value arrives at the edge access node on the Controlled Domain side in time. That access edge node forwards that information element to both the controlled device and the Traffic Sniffer. This latter reads the ID of the device associated with  $y(n)$  (it is the same of the ID carried out by  $x(n)$ ) and sends the value  $y(n)$  to the respective TSE, which also



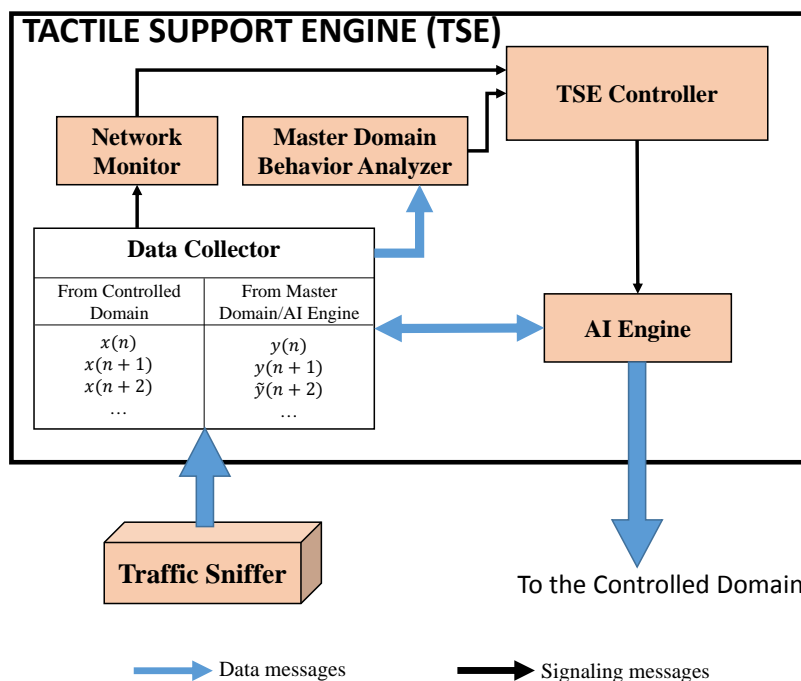


Figure 4.5: Architecture of the TSE

stores it. Instead, if the TSE does not receive  $y(n)$  before the expiration of the timeout for the corresponding  $x(n)$  received so far, it deduces that the Tactile Internet requirements in the network are not satisfied. This triggers the TSE to enter the forecasting phase and calculate the  $\tilde{y}(n)$  value to be sent to the Controlled Domain.

Referring to Fig. 4.5, it is possible to analyze the components inside the proposed TSE: a Data Collector, a Network Monitor, a Master Domain Behavior Analyzer, a TSE Controller and an AI Engine.

The *Data Collector* is a database where the  $x(n)$ , received from the Controlled Domain,  $y(n)$ , received from the Master Domain, and  $\tilde{y}(n)$ , forecasted by the AI Engine, are stored. The *Network Monitor* has the role of monitoring the state of the Network Domain and communicating to the *TSE Controller* if any state change is experienced. To achieve this monitoring task, the Network Monitor starts a timeout lasting  $\Delta t$ , based on the requirements of the application layer, whenever a new sample of  $x(n)$  is received and stored in the Data Collector. If the relevant sample  $y(n)$  is received before the timeout expires, the Network Monitor deduces that the network is in the *Good state*, otherwise the network is considered in the *Bad state*. In this last case, the Network Monitor sends a trigger message to the TSE Controller to alert it of this event. When the TSE Controller receives this message, it starts the forecasting procedure by sending a message to the *AI*

*Engine*, a block implementing a Machine Learning (ML) technique for forecasting samples when needed. The above message contains the number  $N$  of samples to be predicted, and a reference to the model to be used according to the current behavior detected by the Master Domain Behavior Analyzer.

Another component inside the TSE is the *Master Domain Behavior Analyzer*. Its task is to discover changes in the behavior of the human operator by analyzing the  $y(n)$  trace generated by the Master Domain. In this implementation, it is used a deep learning architecture inspired by recent advances in image and video processing, which exploits correlations typical of gaming behaviors in the short-term timescale. The Master Domain Behavior Analyzer acts as a typical deep learning system: it properly formats input data sequences received from the Data Collector when the network is in Good state; such input is fed to a deep neural network that extrapolates and processes input features to provide information regarding whether the game behavior is changed as compared to the behavior detected during the last analysis. This is an important aspect of the TSE because, if a variation in the Master Domain behavior is registered as a new never modelled behavior, the TSE Controller has to trigger the AI Engine for a new training phase. This is required as the previous model has become outdated to forecast the future values, and no models are stored in the *Model Database* (DB) with the revealed features. In this case, as said so far, the TSE Controller sends a message to the AI Engine to start to work.

The AI Engine block is implemented by adopting an efficient, gradient-based prediction and forecasting technique called Long Short Term Memory (LSTM) [66] to forecast missed samples. This technique outperforms the other models in terms of learning from long term dependencies. Further, it operates independently of the gap length which makes it an ideal choice over Recurrent Neural Network (RNN) for the considered system. The LSTM comes under supervised learning, which makes use of an algorithm to learn the mapping function from the input variables to the output variables by determining the relationship between a dependent and an independent variable. The objective is to approximate the mapping function so well that, when the model encounters a new input variable, it can predict the output variables for the respective inputs. LSTM networks are used for classifying, processing, and making predictions based on time series data. A standard LSTM unit (Fig. 4.6) is composed of a cell that remembers values over random time intervals and input, output and forget gates that regulate the flow of information into and out of the cell as described in [66]. This unit contrasts with standard RNN in having only a single *tanh* layer. LSTMs also have a

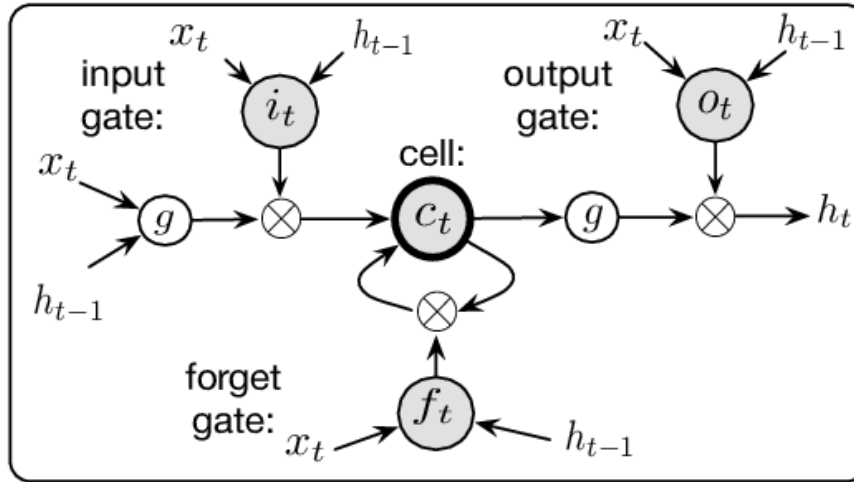


Figure 4.6: A standard LSTM unit

chain-like structure similar to the RNN, but the repeating module has a different structure compared to standard RNN. To be specific, instead of having a single neural network layer, there are four layers that interact in a very special way.

In the proposed implementation, at the beginning, the TSE starts with a basic version of an LSTM model that has been trained using data representing a typical Master Domain behavior scenario and stored inside the AI Engine. New models and updating of the models previously stored are included at runtime when the AI Engine receives trigger messages from the TSE Controller and the network is in Good state.

The behavior of the AI Engine is explained in Fig. 4.7. The AI Engine remains in *Idle* state until it does not receive any trigger message from the TSE Controller. As already said, it can receive two types of messages that correspond to: M1) a generation of  $N$  forecasted samples, using the model that better represents the current Master Domain behavior; M2) a request for calculating a new forecasting model to be included in the Model Database DB.

The first type of message denotes a change of the state of the network. In this case, the TSE Controller requests the forecasting of a certain number of future values, so that the AI Engine enters the *Forecasting* state. It contacts the Data Collector to obtain the last *look\_back* ( $L_B$ ) values received by the Controlled Domain, loads the most appropriate behavior model from the database, and uses it to forecast a number  $N$  of values, as indicated by the TSE Controller. The  $L_B$  parameter is used during the training and forecasting phases to organize the input data, i.e., it indicates how many samples are to be read to predict/forecast the next one. The  $N$  forecasted values are sent to both the Data Collector and the Controlled Domain: the former saves the needed forecasted values in its table

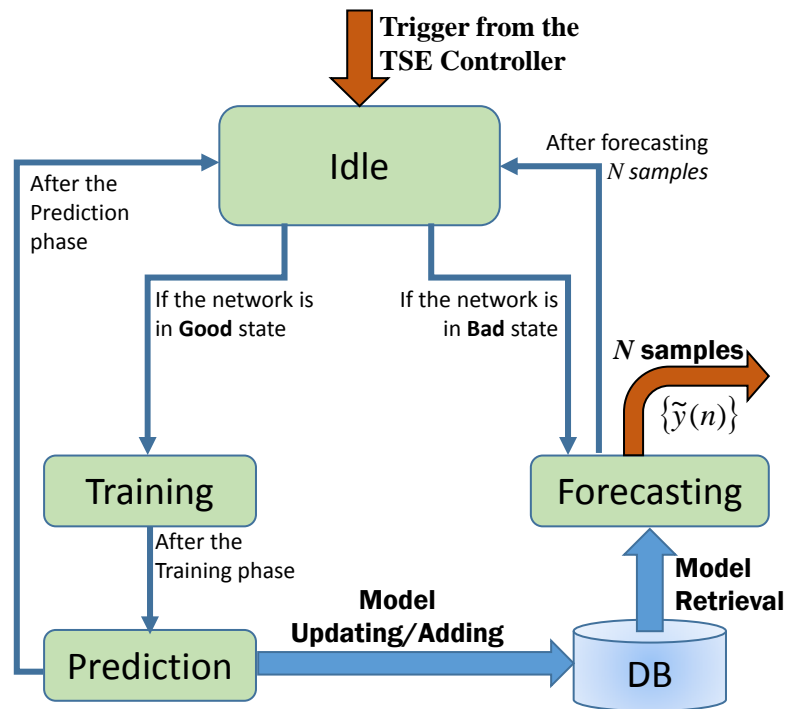


Figure 4.7: TSE. Diagram of AI Engine life cycle

in such a way to maintain a perfect match between  $x(n)$  and its corresponding  $y(n)$ ; the latter uses them to manage the controlled device. After that, the AI Engine returns in the Idle state.

The second type of message represents a change of the Master Domain behavior, triggered by the Master Domain Behavior Analyzer. In this case, the AI Engine changes its state by entering into the *Training* state in order to create a new model that better forecasts possible future values. For this reason, in this message, the TSE Controller includes the size of the dataset that has to be used. Consequently, the AI engine contacts the Data Collector to receive the dataset sample and starts the training phase and then the prediction phase to generate the new model. Models are also compared by considering different values of  $L_B$ , in order to find the best model and the associated value of  $L_B$ . At the end of the prediction phase, AI engine saves the obtained model in the Model Database DB, together with the associated  $L_B$ , or overwrites a previously-calculated less-accurate model, and comes back to the Idle state. From this instant, it will use this new model as long as it remains in the Forecasting state, until another change of the Master Domain behavior is detected. In the Model Database, a mapping between the behavior of the Master Domain and a particular model is set. As a result, in the future, it is not necessary to train a model again if the Master Domain behaves like in some

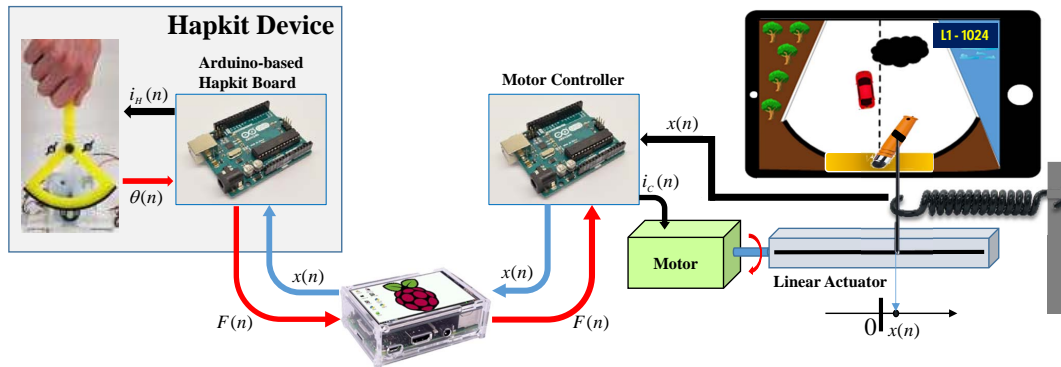


Figure 4.8: TSE Use Case: Local Setup

period of the past. In this case, a stored model corresponding to that behavior can be invoked and used in the forecasting phase.

### Experiment scenario and results

An experiment characterized by a maximum tolerated latency of  $5\text{ ms}$  was realized in order to test the proposed framework. It does not have the same time and failure rate constraints as that of other Tactile Internet applications, i.e.  $1\text{ ms}$ , but it is useful to deduce general guidelines. The experiment consists of two steps: the first one is an implementation of a car racing game played locally; the second step involves simulating a system, containing Master, Network and Control domains to realize the Tactile-Internet like scenario. This is done with the help of a Matlab-based tool that allows simulating the network behavior, by introducing delay and losses, and thereby calculating the performance.

The setup for the first step is depicted in Fig. 4.8. It consists of a game application for Android devices, where the player has to ride a car along a concave road by avoiding obstacles. The presence of obstacles (holes, walls or pedestrians) on the road becomes more frequent as the difficulty of the game level increases, forcing the player to change the trajectory repeatedly. Given the concavity of the road, if the user does not give any input, the car comes to a steady-state position in the middle of the road.

In the basic game, user actions are performed by moving the finger of the user across the screen of an Android tablet or a phone, where the game is running, to apply the desired trajectory changes. In this case, to realize an experiment where the time constraints could be highlighted, the player's finger was substituted by a mini-stylus touch pen moved with the help of a *Linear Actuator*, attached with a motor controlled by a *Motor Controller*. In this scenario an Arduino Microcon-

troller was used to control the motor: it modulates the current  $i_C(n)$  at the input of the motor according to the force  $F(n)$  imposed by the player. The measure of the road concavity is obtained with the help of a spring. The spring is in a state of rest when the car is at the center of the road i.e. when the stylus pen is at the center of the touch bar. Farther the car moves away from the center, higher is the force applied by the spring to bring it back to its original state. The position of the touch pen is represented by  $x(n)$ .

To avoid obstacles, the player can modify the trajectory of the car with the aid of a Hapkit device. Hapkit is an open-hardware device designed by the Stanford University [67] to impose forces on a remote object with only one degree of freedom. It has a joystick-like handle that interacts with the Hapkit Board, an Arduino-based circuit board with functionalities allowing to read data from a magneto-resistive sensor, and a power amplifier that drives the motor connected to it.

With the assistance of the Hapkit device, the player can feel the force relating to the position of the car on the road (i.e. null when the car is in the middle, maximum when the car is at a farther distance from the middle of the lane). The user then imposes an additional force (apart from the force needed to maintain the current position considering the concave shape of the road) to change the trajectory and thereby avoid obstacles. The Hapkit Board gives a torsion moment to the joystick-like handle with a time-variant current  $i_H(n)$ , and the player changes the angle  $\theta(n)$  to modify the car trajectory. Then the Hapkit Board calculates the applied force  $F(n)$  from the angle  $\theta(n)$ . This information is then sent to the RaspberryPi interfaced through a USB cable.

The second step of the experiment is achieved by simulating the system shown in Fig. 4.9, to realize a Tactile-Internet scenario as the one shown in Fig. 4.2. The simulation is realized in a Matlab-based tool, with the help of real traces obtained from the local setup of the first step as inputs.

As shown in Fig. 4.9, the game environment constitutes the Controlled Domain, with the Motor Controller, Motor, Linear Actuator, spring, mini-stylus touch pen and the tablet where the game is running. Two RaspberryPis are connected in both edge sites of the network and are used to transmit data to the Network Domain, and receive data in the opposite direction. The Master Domain is realized with the Hapkit device, a RaspberryPi that is used to communicate with the remote RaspberryPi to exchange messages, and a User display (monitor or a TV) connected to the HDMI port of the RaspberryPi to view the game remotely. Lastly, the Network Domain is a typical network infrastructure implementing the

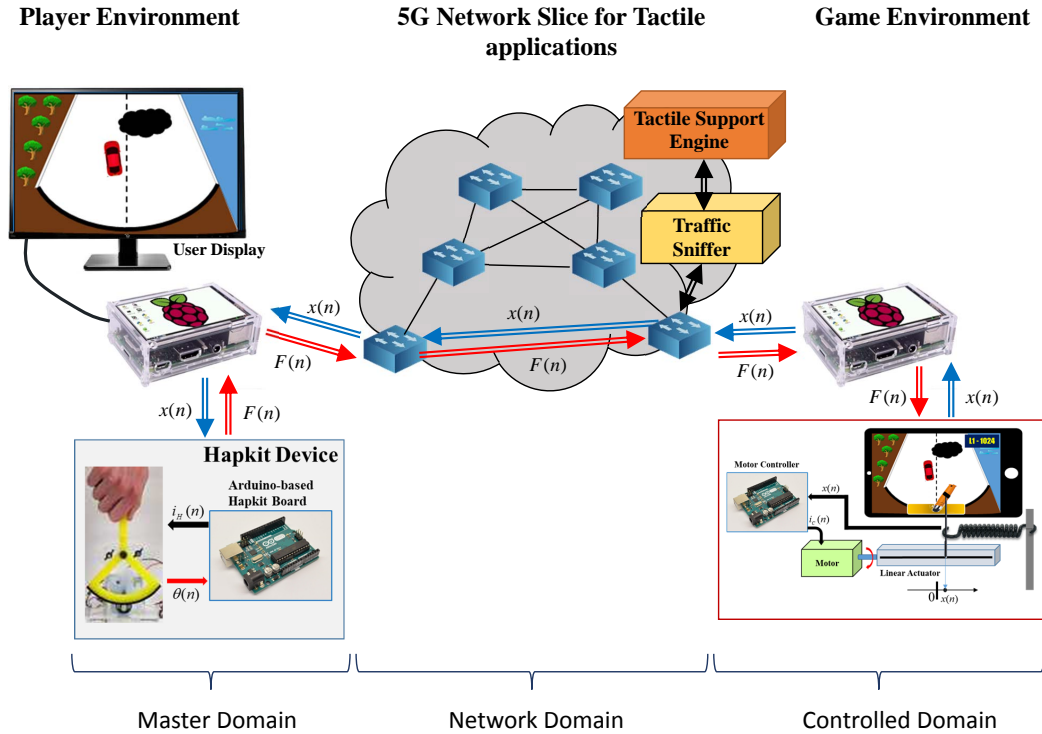


Figure 4.9: TSE Use Case: Distributed Setup

SDN paradigm with two edge access nodes in the considered scenario, one towards the Master Domain and the other towards the Controlled Domain. It is simulated using a two-state Markov model that alternates the state of the network between Good and Bad. An implementation of the TSE is also included according to the architecture presented in Fig. 4.2. Since there is only one controlled device, it is sufficient to have only one TSE deployed inside the Network Domain.

When the Master Domain receives  $x(n)$ , the car position is updated on the User display. Accordingly, the player moves the joystick-like handle to change the car trajectory. The Hapkit Board, in turn, generates  $F(n)$ , which is the force to be applied to the remote spring in the Controlled Domain. This force is then sent to the remote Motor Controller through the RaspberryPi in the Master Domain, the Network Domain, and the RaspberryPi in the Controlled Domain.

If the network is in Good state, the information arrives at the edge node and subsequently to the Traffic Sniffer and TSE on one side and to the Motor Controller on the other side. TSE stores and uses the received information from Controlled and Master Domains as described before. The  $F(n)$  arrived at the Motor Controller is converted into electric current  $i_C(n)$  that controls the Linear Actuator in such a way that it can move the stylus on the tablet screen. On the flip side, if the network is in the Bad state, the  $F(n)$  value does not arrive in time to the

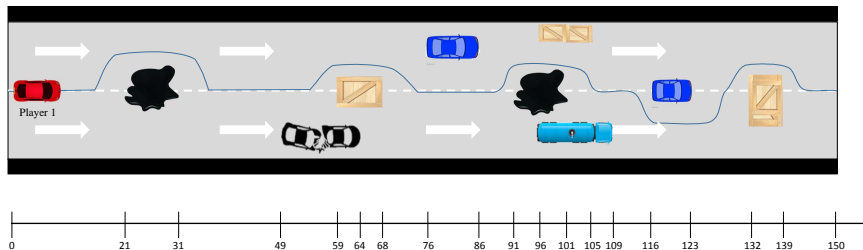


Figure 4.10: TSE. Obstacle positions drawn for the Level 1, Lap 1

edge node at the Controlled Domain side, causing the TSE to forecast and send it to the Motor Controller.

About the game, each level is characterized by a different car speed:  $v_1$  for the level 1,  $v_2 = 1.5 \cdot v_1$  for the level 2, and  $v_3 = 2 \cdot v_1$  for the level 3. Consequently, the game duration is of 150 s for level 1, 100 s for level 2, and 75 s for level 3.

First of all, the game route was generated, constituted by three laps, each with four obstacles in the middle of the road, two on the left and two on the right. The exact placement of the obstacles is randomly chosen accordingly. As an example, Fig. 4.10 shows the obstacle placement drawn for the first lap of level 1.

According to step 1, the three levels of the game were played locally, by using the setup presented in Fig. 4.8, that is, acting on the joystick-like handle, but watching the tablet directly, with no delays and losses between the Master Domain and the Controlled Domain. During this phase, the traces of the position,  $x(n)$ , of the touch pen on the tablet screen, and the subsequent force  $F(n)$  applied by the player and sent by the Hapkit Board to the Motor Controller were saved. The measured traces  $F(n)$  and  $x(n)$  are shown in Fig. 4.11. Since the force applied to a spring follows the Hook law, i.e.  $F(n) = -K_S \cdot x(n)$ , and considering that in this case all messages containing the force arrive at the Motor Controller, each value  $x(n)$  is always proportional to the force  $F(n)$  applied by the player, and hence this is represented by a single curve in Fig. 4.11. The proportionality constant is  $-1/K_S$ ,  $K_S = 4 \text{ N/cm}$  being the spring constant used in our experiment.

Moreover, during this phase, the TSE is fed with the obtained traces for training. The Master Domain Behavior Analyzer, as expected, has detected two behavior changes at the time instances  $t_2 = 150 \text{ s}$  and  $t_3 = 250 \text{ s}$ , i.e. at the beginning of the levels 2 and 3, respectively, represented by an evident change in car speed. For each revealed player behavior, the AI Engine has calculated some trajectory models applied by the player, for different values of  $L_B$ . The achieved models have been compared in terms of Peak Signal-to-Noise Ratio (PSNR), defined as the ratio between the peak power of the real signal  $x(n)$  and the mean power of the



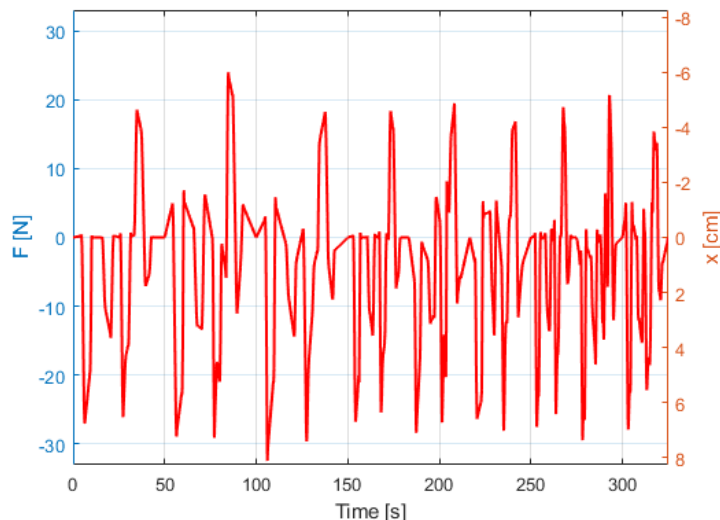


Figure 4.11: TSE. Force and position traces during a game in the local setup

error made on its estimation, i.e.  $E(n) = x(n) - \hat{x}(n)$ , to find the best model for each game level (i.e. for each player behavior). Based on their comparison, shown in Fig. 4.12, at the end of each model computation, the TSE updates the Model Database DB if the found model exhibits a higher PSNR than its predecessor. The best models found during the training phase were the models calculated with  $L_B = 120$  samples for level 1,  $L_B = 60$  samples for level 2, and  $L_B = 100$  samples for level 3.

After that, step 2 sketched in Fig. 4.9 starts. As anticipated so far, a discrete-time Matlab simulator to generate the network behavior, and traces and models found and saved during step 1 were used. The slot duration has been assumed equal to the maximum tolerated end-to-end latency, i.e.  $5 \text{ ms}$ .

The network behavior is modelled with a two-state Markov model, described by the transition probability matrix given below:

$$P^{(N)} = \begin{bmatrix} 1 - 1/\hat{T}_{Good} & 1/\hat{T}_{Good} \\ 1/\hat{T}_{Bad} & 1 - 1/\hat{T}_{Bad} \end{bmatrix} \quad (4.1)$$

where  $\hat{T}_{Good}$  is the mean duration of the network during the good state, set to  $10 \text{ s}$ , equivalent to 2000 slots, while  $\hat{T}_{Bad}$  is the mean duration of the network during the bad state. This latter value has been varied in the range  $[10, 200] \text{ ms}$ , that is equivalent to  $[2, 40]$  slots, in order to evaluate the ability of the proposed system to hide network problems for different durations of them.

The first analysis is aimed at evaluating the impact of the car speed on the TSE prediction ability. For this purpose, the PSNR of the  $x(n)$  trace sent from the

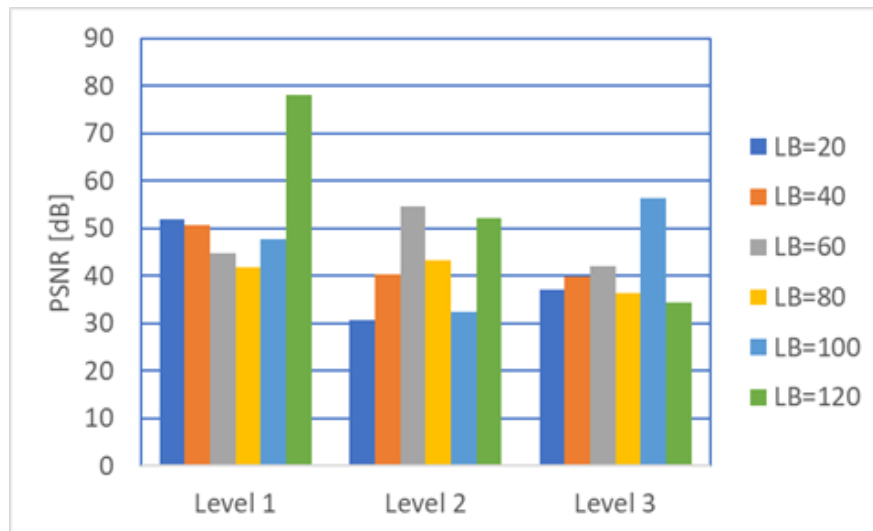
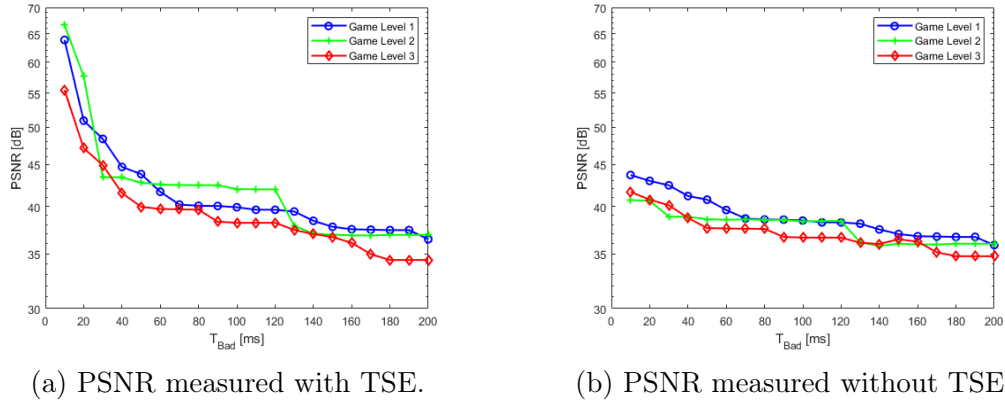


Figure 4.12: TSE. Model comparison for different look\_backs  $L_B$ , at different levels

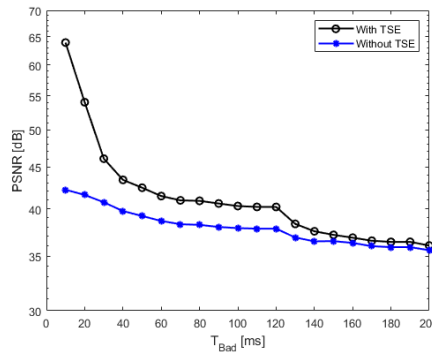
Controlled Domain was measured, considering noise the difference between the value  $x(n)$  achieved with the force actually sent by the Master Domain, and the value  $\hat{x}(n)$  derived by prediction by the TSE whenever samples have not arrived in time. The PSNR measured for the three levels and expressed in decibel is shown in Fig. 4.13a. Since the curves are very close to each other, the figure demonstrates the ability of the model in capturing the player behavior is independent of the car speed. Of course, this depends on the timescale: in this case, the network downtime is of the order of few dozens of milliseconds; in this period the player behavior is strongly autocorrelated because relevant changes in the player actions are visible in a time horizon of the second, as deducible from Figs. 4.10 and 4.11.

Then, in order to evaluate the gain achieved by the proposed TSE with respect to the state of art, i.e., without TSE, the system is simulated with the same traces, but without TSE. In this case, all the skipped values of  $F(n)$  determine that the spring will not be updated sometimes, so causing the touch pen to remain idle. Of course, as soon as the network comes back to the Good state, the first value of  $F(n)$  that arrives in time at the Controlled domain will cause a discontinuity in the position of the spring and, consequently, of the touch pen. Fig. 4.13b presents the PSNR measured for each game level on the trace  $x(n)$ , in the case TSE is not used. In this case, the noise is the difference between the  $x(n)$  value that would have been if  $F(n)$  have arrived at the destination, and the actual value of  $x(n)$  measured on the spring. To better appreciate the gain achieved by using the TSE, in Fig. 4.13c the mean PSNR obtained in the two above cases (with and without TSE) is compared by averaging the PSNR measured during the three



(a) PSNR measured with TSE.

(b) PSNR measured without TSE.



(c) Comparison of the PSNR averaged on the three skill levels

Figure 4.13: TSE Performance analysis

game levels.

Finally, another figure of merit that has to be considered is loss probability that, for example in a Tactile Internet network slice must be less than  $1 \cdot 10^{-7}$ . We have presented its measurement in Fig. 4.14 for the case of no TSE while, for the opposite case, i.e. in the presence of TSE, it is not shown because of the absence of such losses.

Finally, from the plots shown so far, it is possible to observe how performance depends on the network behavior: the worse the network, the lower the PSNR and the higher the loss probability. However, network performance, in terms of both the percentage and the mean duration of the bad state, are not absolute, but depend on the latency constraint imposed by the specific low-latency application scenario. In fact, the variables  $\hat{T}_{Good}$  and  $\hat{T}_{Bad}$  are how the network appears to the TSE and the considered system, when the latency constraint is known. If another system has a lenient latency constraint, it monitors the network better, with a higher  $\hat{T}_{Good}$  and a lower  $\hat{T}_{Bad}$ . Consequently, both PSNR and loss probability result better.

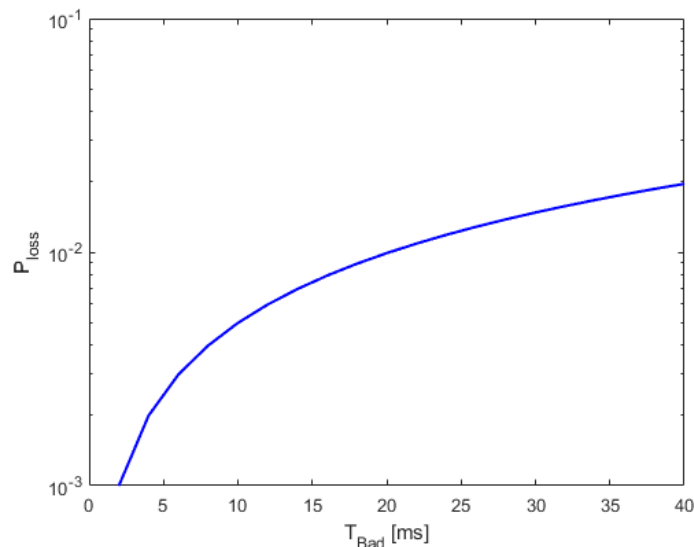


Figure 4.14: Loss probability measured without TSE

## 4.2 Multi-Layer Offloading in Vehicular Networks

In this section, the problem of implementing an optimal offloading technique in a vehicular network slice scenario is addressed. After a first description of the architecture, the characteristics and the requirements of a Vehicular Network in Subsection 4.2.1, the proposed Multi-Layer Offloading technique with the use of the RL is described in Subsection 4.2.2.

### 4.2.1 Vehicular Networks

*Vehicular Networks* will play a crucial role in future Intelligent Transportation Systems (ITS) [68] in which, thanks to the innovations of information and communication technologies, there will be an improvement of aspects like road safety, traffic efficiency and QoS, with a better QoE achieved by the users. The term Vehicular Networks includes not only vehicular-to-vehicular (V2V) communications, but all types of interactions between vehicles and devices placed in the environment in which vehicles move (Vehicular-to-Everything, V2X). The autonomous driving applications is an example of V2X scenario [69] in which vehicles, using devices called On Board Unit (OBU), communicate with each other and with everything around them (pedestrians, traffic lights and infrastructure in general), receiving and processing data continuously.

The nature of this type of applications makes parameters like latency and bandwidth very important, in particular very low latency and high bandwidth availability are required. Also, computational resources are fundamental in this field,

	Vehicular Edge Computing	Vehicular Cloud Computing
Location	At user's proximity	Remote location
Latency	Low	High
Mobility support	High	Limited
Decision Process	Local	Remote
Communication	Real Time	Constraints in bandwidth
Storage Capacity	Limited	Highly Scalable
Context Awareness	Yes	No
Device Heterogeneity	Highly Supported	Limited Supported
Computing capability	Medium	High
Cost of Development	Low	High

Table 4.1: Comparison between Vehicular Edge Computing and Vehicular Cloud Computing [72]

because they allow processing of the received data. Another aspect that has to be considered is the complexity of possible scenarios: vehicles could move with different speeds according to the environment (urban or extra-urban) and the number and type of "objects" with which to communicate could vary in time. All these aspects explain why Vehicular Networks represent one of the most challenging network slices in 5G networks.

As said before, a primary role is played by the computational resources that allow vehicles to process received data locally. To reduce packet loss and latency, each vehicle could offload a certain number of data jobs to external servers. Due to latency requirements, it is impossible to use a typical Vehicular Cloud Computing (VCC) approach, based on a two-level client-server architecture with servers placed in remote clouds, away from the network access point of the vehicle. The solution provided by 5G network technology for this problem is MEC [70] [71]: it represents a low latency solution based on the idea of placing part of computing and storage resources at the edge of the network, near to the end-users. From the combination of Vehicular Networks and MEC derives *Vehicular Edge Computing* (VEC). Table 4.1 highlights the main differences between VCC and VEC paradigms.

VEC is a three-layer architecture (Fig. 4.15) constituted by:

- *User (Vehicular) Layer*: it is constituted by groups of geographically close smart vehicles which share storage and computing resources through V2V links. It is responsible for recovering information from integrated sensors, GPS, cameras, radar and other devices that can be installed on board of the vehicle. This information can be used as input data for applications that run locally or sent to the top layer for processing or storage;

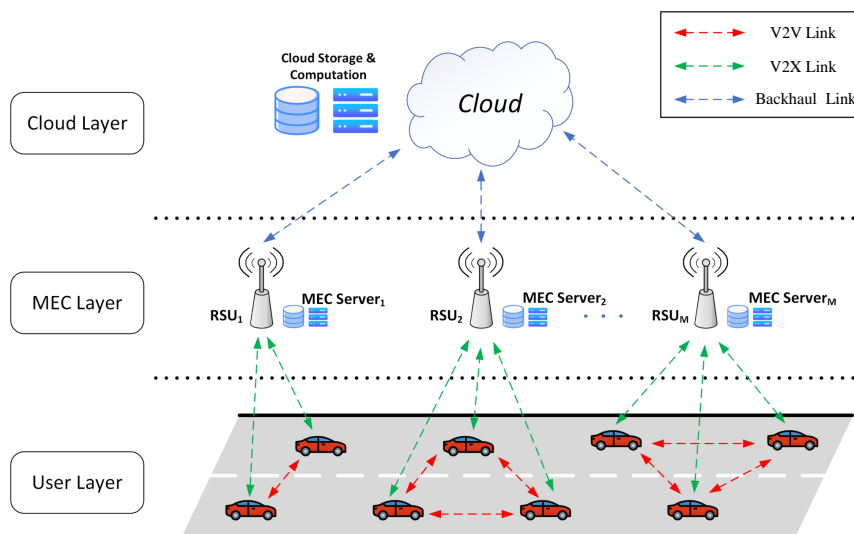


Figure 4.15: VEC Network Architecture

- *MEC Layer*: it acts as an intermediary between the User Layer and the Cloud Layer, guaranteeing communication between the two levels. It is constituted by servers distributed along the road, which act as edge servers and which are characterized by high communication, computing and storage resources when compared with those of vehicles. These servers are integrated with the existing network infrastructure, i.e. Base Station (BSs), Road Side Units (RSUs) or other Access Point (APs). It processes data regarding applications with very strict latency requirements;
- *Cloud Layer*: it consists of a cloud infrastructure that includes both the computational and storage part. It receives information from the lower MEC layer and provides significantly higher computing power and a wider coverage area. The main features provided by this layer are data aggregation, data mining, archiving and computation of complex data regarding applications with no strict latency requirements.

The VEC framework is characterized by several advantages [73] in terms of:

- response time, thanks to the greater proximity of the servers if compared to an only cloud-based solution;
- bandwidth: moving storage resources from cloud to the edge, a VEC network could reduce the enormous stress of the backhaul network due to the possibility of processing data at the edge;
- storage capacity: data can be stored directly on the edge servers and,

through efficient caching techniques, vehicles can access them, guaranteeing time constraints;

- better QoS and QoE: the edge servers placed near the vehicles, could improve these two parameters. For example, by receiving real-time information on the location, vehicle behavior and the surrounding environment, the VEC servers could use them to create high-definition maps or to send content to users based on their interests.

However, there are also some critical aspects:

- high mobility of the vehicles, so the network topology is extremely variable, due also to the frequent handovers that are performed; this causes variations of the mutual position between vehicles and their access points on which the MEC servers are installed;
- hostile communication environment, especially in an urban scenario, in which numerous obstacles can make communication difficult;
- resources management, if compared with the cloud server, the resources in an edge server are limited, so techniques for dynamic allocation and orchestration become fundamental;
- task migration: due to the limited vehicle capacity, there is a need to perform offloading operations towards the MEC servers. The dynamism of the environment and of the network topology makes decision making a fundamental aspect;
- security and privacy: if, on the one hand, the idea of moving the cloud to the edge of the network introduces the numerous benefits described so far, from a security point of view the edge servers could be easily attacked; moreover, they may receive offloaded tasks containing private and sensitive data.

The use of these technologies makes it necessary to implement efficient models to carry out offloading operations between vehicles and MEC servers, considering the dynamism of the environment and of the network topology. Performing or not offloading becomes a critical problem in which it is necessary to take into account features that constrain its feasibility. Research on computation offloading [74] divides the general problem into three macro-areas: *Decision Making*, *Allocation of resources at MEC level* and *Mobility management*.

In particular, at the Vehicular Level, the offloading policies, which deal with managing the spectral and computational resources for the execution of the tasks, can be of three types [72]:

- *Local Computing*: the vehicle generates tasks itself and processes them locally, without offloading procedure;
- *Partial Offloading*: some tasks are offloaded to the MEC servers, the others are processed locally. This is the most difficult offloading procedure because the vehicle has to implement an intelligence to understand if a group of tasks, based on the particular application, could be managed separately or not;
- *Total Offloading*: the simplest offloading technique, because the vehicle sends all the tasks to the MEC server.

Furthermore, each vehicle can be both a Task Vehicle (TaV), so it can make a service request, by delegating the execution of tasks to the MEC servers, and a Service Vehicle (SeV) or relay node, that is, in a cooperative context, it can be itself to help in the execution of tasks, providing its computational resources to the network. Each vehicle can dynamically change its role during the moving and this depends on factors such as the request for the development of services or the availability of sufficient and shareable resources [75].

The Computation Offloading process can be divided into the following phases:

- *Service discovery*: the vehicles identify the surrounding infrastructures within their operating range;
- *Task upload*: vehicles send tasks to be processed to the service provider discovered in the previous phase, through V2X communications;
- *Task processing*: the service provider, following their policies, manage the task or can choose to offload it to other service providers;
- *Result return*: service providers send results of the task processing phase to the source vehicles.

### 4.2.2 Multi-Layer offloading and RL

In the context described before, it has been proposed a multi-layer platform for job offloading. This work can be found in the paper [76], presented at 18th Med-ComNet2020. The considered scenario is shown in Fig. 4.16 where a portion of the



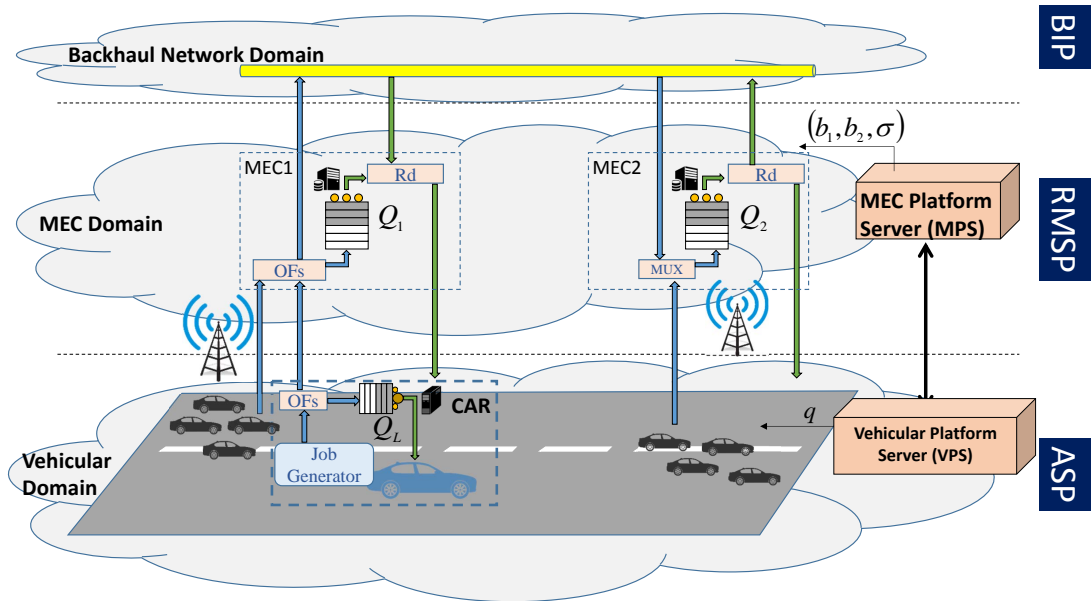


Figure 4.16: Vehicular Scenario: multi layer system architecture

road is depicted, highlighting the presence of APs equipped with MEC Servers to provide vehicles with computing facilities at the edge of the 5G network. It is assumed that one network slice is used for each travel direction and, in the following, vehicles flowing from the left to the right are considered. This means that, for each slice, jobs can be offloaded only to the next MEC Server along the pathway.

It is assumed the presence of three different stakeholders:

- *Application Service Provider (ASP)*: it is the owner and the manager of the application service. Its role is to provide and manage devices to be installed on the vehicles;
- *RAN/MEC Service Provider (RMSP)*: it installs BSs along the road to provide vehicles with connectivity and MEC services;
- *Backhaul Infrastructure Provider (BIP)*: it provides high-speed connection links between BSs. These links can be realized with fiber optical cables, or even with UAV [77].

The relationships between the above providers are structured as client/server: the ASP receives a service from the RMSP, and the latter receives a service from the BIP. Each of them implements an independent optimization strategy to maximize its revenue by optimizing the tradeoff between costs and performance. Each provider is in charge of a specific domain (see Fig. 4.16). The ASP works

in the *Vehicular Domain*, where smart vehicles flowing on the road generate jobs to be processed for a given application. The smart vehicles are equipped with a computing station, provided by the ASP, which allows processing these jobs locally.

When the computing station is busy for processing a job, the other incoming jobs are queued and managed with a FIFO policy, with a delay problem in the processing phase. In order to reduce these delays, each vehicle will offload a percentage  $q$  of its job flow to the *MEC Domain*, percentage that depends on both the requirements of the specific application and the costs that the ASP is available to sustain, due to the price applied by the RMSPP to process jobs offloaded by the Vehicular Domain. For the sake of simplicity, we will assume that  $q$  is equal for all the vehicles. The decision of this  $q$  value is demanded to the *Vehicular Platform Server* (VPS), an entity that is in charge of management and orchestration of the applications at the Vehicular Domain.

The *MEC Domain* is the edge of a 5G network, where APs in the RAN are equipped with a MEC Server that is able to process jobs offloaded by vehicles. Each MEC Server may be installed in a self-consistent station, which is able to work even in conditions of lack of the power grid, so each MEC Server is equipped with an autonomous power generator and a battery. This means that the energy consumption of this element is a critical aspect to evaluate to guarantee the continuity of the service. Moreover, let each MEC Server be equipped with Computing Elements (CEs), such as CPUs, for processing jobs offloaded by the vehicles. The higher the number of active CEs, the smaller is the MEC processing delay for the jobs, but more processing power implies higher energy consumption. These two parameters play an important role in the MEC Domain: depending on the kind of application, a job may require small processing latencies; on the other hand, increasing the energy consumption could represent a threat for the whole system, since this could drain the batteries of the MEC Servers.

Offloading jobs between adjacent MEC Server is a solution to reduce MEC processing delay [78]. For this reason, referring to Fig. 4.16, MEC Server 1 could offload a fraction of the received jobs to the MEC Server 2. A *MEC Platform Server* (MPS) is entitled to decide, for each MEC Server of the MEC Domain, whether to offload jobs or perform local computations. In addition, given the limited availability of energy at each MEC Server, the MPS is also in charge of choosing how many CEs should be activated on each MEC Server at runtime. The choice of the amount of jobs to be offloaded from one MEC Server to the next one,  $(1 - \sigma)$ , and the number of active CEs  $b_i$  for each server  $i$  is done by

the MPS considering the queueing delay in the queues where jobs are queued, the power consumption due to the active CEs, and the cost of offloading jobs from one MEC Server to the next one, due to the price applied by the BIP at the Backhaul Network Domain.

To choose the  $q$  value, the VPS performs some numerical steps. First of all, the local computing station of each vehicle is model as a  $B/B/1/K_V$  queueing system (Bernoulli arrivals, Bernoulli departures, and only one service facility). The parameter  $K_V$  represents the maximum number of jobs that can be accommodated in the local computing station, considering that  $K_V - 1$  jobs are in the queue and one job in the service facility. Let  $p_V$  be the per-slot job generation probability for one vehicle, and  $\xi_V$  the departure probability in the service facility. Considering that a percentage  $q$  of the flow is offloaded to the MEC Domain, the per-slot arrival probability of a job to the queueing system is  $(1 - q)p_V$ . The state of the Markov chain modeling the behavior of this system is defined as  $S^{(QV)}(n)$ , representing the number of jobs that are present in the slot  $n$  in the whole local computing station. Calculate, for each  $q$ , the mean delay  $\bar{\phi}_L$  suffered in the local computing stations, following the discrete-time queueing system theory, constitutes the first step of the VPS workflow. After that, the VPS receives the  $\bar{\phi}_M$  delay calculated by the MPS for a given  $q$ . In this way, the mean delay  $\bar{\phi}_V$  suffered by jobs offloaded from the Vehicular Domain can be obtained by averaging  $\bar{\phi}_L$  and  $\bar{\phi}_M$ :

$$\bar{\phi}_V = q\bar{\phi}_M + (1 - q)\bar{\phi}_L \quad (4.2)$$

Then, it can calculate the reward function 4.3, weighing the cost for offloading,  $\bar{\mu}$ , and the mean delay suffered by jobs to be served by either the local computing station or the MEC Domain:

$$F_{RW}^{(VPS)} = -\gamma_1 \frac{\bar{\mu} - \bar{\mu}_{MIN}}{\bar{\mu}_{MAX} - \bar{\mu}_{MIN}} - \gamma_2 \frac{\bar{\phi}_V - \bar{\phi}_{V,MIN}}{\bar{\phi}_{V,MAX} - \bar{\phi}_{V,MIN}} \quad (4.3)$$

The term  $\bar{\mu}$  represents a penalty and depends on the per-job offloading price applied by the RMSP, and is proportional to the amount of offloaded jobs to the MEC Domain. The constants  $\gamma_1$  and  $\gamma_2$  are weighing parameters that can be properly tuned according to some system management criteria defined by the ASP. The parameters  $\bar{\mu} \in [\bar{\mu}_{MIN}, \bar{\mu}_{MAX}]$  and  $\bar{\phi}_V \in [\bar{\phi}_{V,MIN}, \bar{\phi}_{V,MAX}]$ , are normalized in the interval  $[0, 1]$  in order to be comparable.

Finally, the VPS decides the offloading percentage  $q^*$  as the best value of  $q$  that maximizes the function  $F_{RW}^{(VPS)}$ .

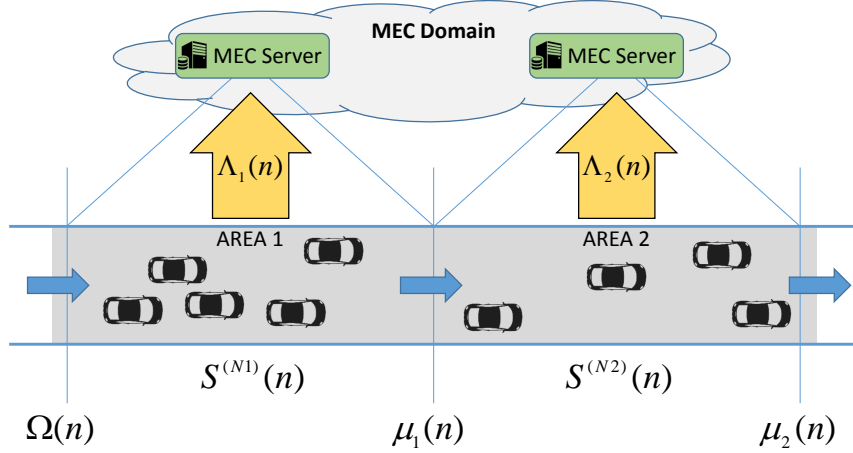


Figure 4.17: Vehicular Scenario: job arrival process

Decisions at the MEC Domain are taken periodically at the beginning of each slot time  $\Delta$ , by MPS using a model-based RL technique. In the sequel, to simplify notation, we will consider a MEC Domain constituted by only two MEC Servers, as depicted in Fig. 4.17.

As said so far, its task is to decide the number of CEs to be active during the time slot for each MEC Server,  $b_1$  and  $b_2$ , and the maximum number of jobs,  $\sigma$ , to be locally managed. Due to the correlation between the number of vehicles in each area, the job arrival process from the Vehicular Domain can be model as a bi-dimensional Markov modulated process  $\underline{\Lambda}(n) = (\Lambda_1(n), \Lambda_2(n))$ , where  $\Lambda_1(n)$  and  $\Lambda_2(n)$  represent the number of jobs offloaded by the vehicles in the area 1 and in the area 2, respectively.

Assuming that there is a maximum number of vehicles,  $N_{MAXi}$ ,  $i \in \{1, 2\}$ , that can stay in the same area, simultaneously, vehicles can enter the next area as long as there is enough room; otherwise, they will stay in the previous one. Also, let  $\mathfrak{S}^{(\Lambda_i)}$  be the set of possible values that the arrival process  $\Lambda_i(n)$  can assume. Due to the assumption that each vehicle can generate at most one job per slot, it is possible to say that  $\mathfrak{S}^{(\Lambda_i)} \equiv [0, N_{MAXi}]$ . Referring to Fig. 4.17, the process representing the number of vehicles that arrive to the area 1 is defined as a SBBP [58]  $\Omega(n)$  modulated by a Markov chain  $S^{(\Omega)}(n)$  with a transition probability matrix  $P^{(\Omega)}$ , and the arrival-process probability matrix  $B^{(\Omega)}$ , whose generic elements are defined as follows:

$$P_{[s'_\Omega, s''_\Omega]}^{(\Omega)} = Pr\{S^{(\Omega)}(n) = s''_\Omega | S^{(\Omega)}(n-1) = s'_\Omega\} \quad (4.4)$$

$$B_{[s''_{\Omega}, \omega]}^{(\Omega)} = Pr\{\Omega(n) = \omega | S^{(\Omega)}(n) = s''_{\Omega}\} \quad (4.5)$$

In the same way, the number of vehicles that can leave the area 1 to enter the area 2, and of the vehicles that can leave the area 2, is modelled as two SBBP processes,  $\mu_1(n)$  and  $\mu_2(n)$ , modulated by the Markov chains  $S^{(\mu_1)}(n)$  and  $S^{(\mu_2)}(n)$ , respectively. Also in this case, it is possible to characterize these two process with the matrices  $P^{(\mu_i)}$  and  $B^{(\mu_i)}$ , two inputs of the problem.  $\mathfrak{S}^{(\Omega)}$ ,  $\mathfrak{S}^{(\mu_1)}$  and  $\mathfrak{S}^{(\mu_2)}$  are the sets of possible values for the processes  $\Omega(n)$ ,  $\mu_1(n)$  and  $\mu_2(n)$ . Each area can be modeled with a bi-dimensional Markov chain defined as follows:

$$\underline{S}^{(Ai)}(n) = (S^{(Ni)}(n), S^{(\mu_i)}(n)), i \in \{1, 2\} \quad (4.6)$$

$S^{(Ni)}(n)$  is the process that represents the number of vehicles in the area  $i$  in the slot  $n$ , while  $S^{(\mu_i)}(n)$  represents the state of the vehicle departure process from the same area. The whole state of the underlying Markov chain of the whole bi-dimensional job emission process  $\underline{\Lambda}(n)$  can be defined as follows:

$$\underline{S}^{(\Lambda)}(n) = (S^{(\Omega)}(n), S^{(A1)}(n), S^{(A2)}(n)) \quad (4.7)$$

After that, it is possible to model the system at the MEC Domain with a Markov Decision Process (MDP)  $\Sigma$ , defining a three-dimensional discrete-time Markov chain:

$$\underline{S}^{(\Sigma)}(n) = (\underline{S}^{(\Lambda)}(n), S^{(Q1)}(n), S^{(Q2)}(n)) \quad (4.8)$$

where  $S^{(Q1)}(n)$  and  $S^{(Q2)}(n)$  are the states of the queues  $Q_1$  and  $Q_2$  located in the two MEC Servers and used to access their CEs. Let  $K$  be the maximum number of jobs that can be buffered in each of those queues, this means that  $S^{(Q_i)}(n) \in [0, \dots, K]$ .

In order to calculate the optimal policy  $\rho$  that specifies an action  $a$  for each state of  $\Sigma$ , the transition probability matrix  $P^{(\Sigma|\rho)}$  and the immediate reward matrix  $\Psi^{(\Sigma|\rho)}$  are needed. The generic element of  $P^{(\Sigma|\rho)}$  represents the transition probability from the state  $s'_{\Sigma}$  to the state  $s''_{\Sigma}$  when, according to the policy  $\rho$ , the action  $a$  is performed at the beginning of the slot  $n$  according to the starting state  $s'_{\Sigma}$ . It is defined by:

$$P_{[s'_{\Sigma}, s''_{\Sigma}]}^{(\Sigma|a)} = Pr\{S^{(\Sigma)}(n) = s''_{\Sigma} | S^{(\Sigma)}(n-1) = s'_{\Sigma}, A(n) = a\} \quad (4.9)$$

It can be derived as in [79] considering here that the arrival processes to  $Q_1$  and

$Q_2$  are correlated to each other and modeled with the process  $\underline{\Lambda}(n)$ .

The generic element of  $\Psi^{(\Sigma|\rho)}$ , representing the immediate reward received performing the action  $a$  at the slot  $n$  when the system transits from  $s'_\Sigma$  to  $s''_\Sigma$ , is defined as follows:

$$\Psi_{[s'_\Sigma, s''_\Sigma]}^{(\Sigma|a)} = E\{R(n)|S^{(\Sigma)}(n-1) = s'_\Sigma, S^{(\Sigma)}(n) = s''_\Sigma, A(n) = a\} \quad (4.10)$$

To calculate it, the expected value of the immediate reward for a given transition from  $s'_\Sigma$  to  $s''_\Sigma$ , and for a given action  $a$  taken according to the state initial state, is necessary to be defined. It is calculated by weighing the three key parameters characterizing the MEC Domain behavior: power consumption, cost/gain for offloading, and delay. The following reward function can be defined as the immediate reward of the implemented decisions:

$$F_{RW}^{(MPS)} = -c_1 \frac{\bar{\xi} - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} - c_2 \frac{\bar{\vartheta} - \bar{\vartheta}_{MIN}}{\bar{\vartheta}_{MAX} - \bar{\vartheta}_{MIN}} - c_3 \frac{\bar{\phi}_M - \bar{\phi}_{M,MIN}}{\bar{\phi}_{M,MAX} - \bar{\phi}_{M,MIN}} \quad (4.11)$$

with  $\bar{\xi}$  proportional to the mean number of active CEs and representing the mean power consumed by these ones;  $\bar{\vartheta}$  considers the mean cost applied by the BIP for offloading towards another MEC Server through the backhaul network, and the mean gain received by accepting offload from the vehicles based on the price applied to the Vehicular Domain;  $\bar{\phi}_M$  is the mean delay for a job processed by the MEC Domain, depending on the mean number of jobs in the queues. The parameters  $c_1$ ,  $c_2$  and  $c_3$  can be properly tuned by the RMSP according to some system management criteria (e.g. if energy saving is the priority,  $c_1$  is chosen greater than  $c_2$  and  $c_3$ ).

After that, the immediate reward associated to that transition is given by:

$$\begin{aligned} \Psi_{[s'_\Sigma, s''_\Sigma]}^{(\Sigma|a)} = & -c_1 \frac{\bar{\xi}(s'_\Sigma, a) - \bar{\xi}_{MIN}}{\bar{\xi}_{MAX} - \bar{\xi}_{MIN}} + \\ & -c_2 \frac{\bar{\vartheta}(s'_\Sigma, a) - \bar{\vartheta}_{MIN}}{\bar{\vartheta}_{MAX} - \bar{\vartheta}_{MIN}} - c_3 \frac{\bar{\phi}_M(s'_\Sigma, s''_\Sigma, a) - \bar{\phi}_{M,MIN}}{\bar{\phi}_{M,MAX} - \bar{\phi}_{M,MIN}} \end{aligned} \quad (4.12)$$

The first term regards the penalty (it becomes a reward thanks to the minus sign) received for power consumption due to the active CEs, when the action  $a = (b_1, b_2, \sigma)$  is performed according to the starting state  $s'_\Sigma$ . It is

$$\bar{\xi}(a) = (b_1 + b_2) \cdot \varphi_{CE} \quad (4.13)$$

where  $\wp_{CE}$  is the power consumption of each active CE. The terms  $b_1$  and  $b_2$  are the actual numbers of CEs that have been decided, as part of the action  $a$ , to be active in the current slot.

The second term is the reward due to offloading. It is constituted by two parts: a) a revenue proportional to the amount of received jobs offloaded by the Vehicular Domain, i.e.  $E\{\Lambda\}$ , with a proportional constant  $\Theta_{V \rightarrow M}^{(OL)}$  representing the price applied by the RMSF to process one job at the MEC layer; b) a penalty proportional to the mean number of offloaded jobs to another MEC Server,  $E\{\Phi|a\}$ , with a proportional constant  $\Theta_{M \rightarrow M}^{(OL)}$  representing the per-job offload cost from the MEC1 server to the MEC2 server:

$$\bar{\theta}(a) = -\Theta_{V \rightarrow M}^{(OL)} \cdot E\{\Lambda_1 + \Lambda_2\} + \Theta_{M \rightarrow M}^{(OL)} \cdot E\{\Phi|a\} \quad (4.14)$$

where

$$E\{\Lambda\} = \sum_{\forall s_\Lambda \in \mathfrak{S}^{(\Lambda)}} \sum_{\forall \lambda_1 \in \mathfrak{S}^{(\Lambda_1)}} \sum_{\forall \lambda_2 \in \mathfrak{S}^{(\Lambda_2)}} (\lambda_1 + \lambda_2) \cdot \pi_{[s_\Lambda]}^{(\Lambda)} \cdot B_{[s_\Lambda, \lambda_1, \lambda_2]}^{(\Lambda)} \quad (4.15)$$

$$E\{\Phi|a\} = \sum_{\forall \lambda_1 \in \mathfrak{S}^{(\Lambda_1)}} \sum_{\forall \lambda_2 \in \mathfrak{S}^{(\Lambda_2)}} (\lambda_1 - \min\{\sigma, \lambda_1\}) \cdot B_{[s_{N1}''', s_{N2}''', \lambda_1, \lambda_2]}^{(\Lambda)} \quad (4.16)$$

Finally, the third term in 4.12 regards the delays suffered in the MEC1 and MEC2 queues. To this end, let define the number of jobs arriving from the areas 1 and 2 and not lost, as  $\tilde{\lambda}_1$  and  $\tilde{\lambda}_2$ , respectively. In order to calculate  $\tilde{\lambda}_1$ , due to the assumption that departures occur before arrivals, arrivals find the  $Q_1$  queue state, whose value at the beginning of the slot  $n$  was  $s'_{Q_1}$ , to the value  $s''_{Q_1, INT} = \max\{s'_{Q_1} - b_1, 0\}$ . Therefore, considering that the final state of this queue at the slot  $n$  will be  $s''_{Q_1}$ , it is possible to define:

$$\tilde{\lambda}_1 = s''_{Q_1} - s''_{Q_1, INT} \quad (4.17)$$

Likewise, in order to calculate  $\tilde{\lambda}_2$ , we have to account that the  $Q_2$  queue state goes through two different steps before the arrival of the offloaded jobs coming from MEC1: starting from the state  $s'_{Q_2}$ , after the departures it reaches the state  $s''_{Q_2, INT1} = \max\{s'_{Q_2} - b_2, 0\}$ ; then, after the arrivals from the area 2, it reaches the state  $s''_{Q_2, INT2} = \min\{s''_{Q_2, INT1} + \lambda_2, K\}$ . Therefore, the number of job arrivals from the area 2 is:

$$\tilde{\lambda}_2 = s''_{Q_2, INT2} - s''_{Q_2, INT1} \quad (4.18)$$

Finally, the number of offloaded jobs that enter the queue  $Q_2$  is:

$$\tilde{\lambda}_{OL} = \min\{\lambda_{OL}, K - s''_{Q_2,INT}\} \quad (4.19)$$

where  $\lambda_{OL} = \lambda_1 - \min\sigma$ ,  $\lambda_1$  is the number of jobs offloaded by the MEC1. Assuming that conditions of those queues remain constant, and applying the total probability theorem to the number of jobs arriving from both the areas,  $\bar{\phi}_M$  is defined as:

$$\begin{aligned} \bar{\phi}_M(\underline{s}'_{\Sigma}, \underline{s}''_{\Sigma}, a) = & \sum_{\forall \lambda_1 \in \mathfrak{S}(\Lambda_1)} \sum_{\forall \lambda_2 \in \mathfrak{S}(\Lambda_2)} B_{[s''_{N_1}, s''_{N_2}, \lambda_1, \lambda_2]}^{(\Lambda)} \cdot \frac{1}{\tilde{\lambda}_1 + \tilde{\lambda}_2 + \tilde{\lambda}_{OL}} \cdot \\ & \cdot [\tilde{\lambda}_1/b_1 \cdot [s''_{Q_1} + (s''_{Q_1,INT} + 1)]/2 + \tilde{\lambda}_2/b_2 \cdot [s''_{Q_2,INT_2} + (s''_{Q_2,INT_1} + 1)]/2 + \\ & + \tilde{\lambda}_{OL}/b_2 \cdot [s''_{Q_2} + (s''_{Q_2,INT_2} + 1)]/2] \end{aligned} \quad (4.20)$$

where the first addendum inside the squared brackets is the mean delay suffered by the jobs queued in the  $Q_1$  queue, the second one is the mean delay suffered by the jobs generated by the area 2, while the third one is the mean delay suffered by the jobs generated by the area 1 and offloaded to  $Q_2$ .

The optimal policy  $\rho^*$  is calculated by solving the Bellman optimality equation system applying the function 4.3 as the Immediate Reward  $R(n)$ . The Cumulative Reward, indicated as  $G(n)$ , is defined as follows:

$$G(n) = \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1), \gamma \in [0, 1] \quad (4.21)$$

where  $\gamma$  is the discount factor, as explained in the previous section.

In the end, the steady-state probability array of the MEC Domain state defined in 4.8 and the main performance parameters characterizing the reward function in 4.12 can be calculated. More specifically, the mean delay suffered by jobs offloaded to the MEC Domain is defined as:

$$\bar{\phi}_M = \frac{E\{\tilde{\lambda}_1\} \cdot \bar{\delta}_{Q_1} + E\{\tilde{\lambda}_2\} \cdot \bar{\delta}_{Q_2}}{E\{\tilde{\lambda}_1\} + E\{\tilde{\lambda}_2\}} \quad (4.22)$$

where the first addendum inside the squared brackets is the mean delay suffered by the jobs queued in the  $Q_1$  queue, the second one is the mean delay suffered by the jobs generated by the area 2, while the third one is the mean delay suffered by the jobs generated by the area 1 and offloaded to  $Q_2$ .

The optimal policy  $\rho^*$  is calculated by solving the Bellman optimality equation system applying the function 4.3 as the Immediate Reward  $R(n)$ . The Cumulative



Reward, indicated as  $G(n)$ , is defined as follows:

$$G(n) = \sum_{k=0}^{\infty} \gamma^k \cdot R(n+k+1), \gamma \in [0, 1] \quad (4.23)$$

where  $\gamma$  is the discount factor, as explained in the previous section.

In the end, the steady-state probability array of the MEC Domain state defined in 4.8 and the main performance parameters characterizing the reward function in 4.12 can be calculated. More specifically, the mean delay suffered by jobs offloaded to the MEC Domain is defined as:

$$\bar{\delta}_{Q_1} = \bar{N}_{Q_1} / \bar{\Lambda}_{Q_1} \quad (4.24)$$

with  $\bar{N}_{Q_1}$  (mean number of jobs in the  $Q_1$ ) and  $\bar{\Lambda}_{Q_1}$  (mean number of jobs arrived to  $Q_1$ ) respectively :

$$\bar{N}_{Q_1} = \sum_{\forall s_{\Sigma} \in \mathfrak{S}(\Sigma)} s_{Q_1} \cdot \pi_{[s_{\Sigma}]}^{(\Sigma)} \quad (4.25)$$

$$\begin{aligned} \bar{\Lambda}_{Q_1} = & \sum_{\forall s'_{\Sigma} \in \mathfrak{S}'(\Sigma)} \sum_{\forall s''_{\Lambda} \in \mathfrak{S}''(\Lambda)} \sum_{\forall \lambda_1 \in \Omega(\Lambda_1)} \sum_{\forall \lambda_2 \in \Omega(\Lambda_2)} \min\{\sigma, \lambda_1, K - s'_{Q_1}\} \cdot \pi_{[s'_{\Sigma}]}^{(\Sigma)} \cdot \\ & \cdot P_{[s'_{\Lambda}, s''_{\Lambda}]}^{(\Lambda)} \cdot B_{[(s''_{N_1}, s''_{N_2}), \lambda_1, \lambda_2]}^{(\Lambda)} \end{aligned} \quad (4.26)$$

The mean delay in the queue  $Q_2$  can be calculated as in 4.24, considering that the number of job arrivals in one slot to this queue is  $\lambda_2$  from the area 2, plus the jobs arriving from the area 1 for offloading. So, it is possible to define:

$$\bar{N}_{Q_2} = \sum_{\forall s_{\Sigma} \in \mathfrak{S}(\Sigma)} s_{Q_2} \cdot \pi_{[s_{\Sigma}]}^{(\Sigma)} \quad (4.27)$$

$$\begin{aligned} \bar{\Lambda}_{Q_2} = & \sum_{\forall s'_{\Sigma} \in \mathfrak{S}'(\Sigma)} \sum_{\forall s''_{\Lambda} \in \mathfrak{S}''(\Lambda)} \sum_{\forall \lambda_1 \in \Omega(\Lambda_1)} \sum_{\forall \lambda_2 \in \Omega(\Lambda_2)} \pi_{[s'_{\Sigma}]}^{(\Sigma)} \cdot P_{[s'_{\Lambda}, s''_{\Lambda}]}^{(\Lambda)} \cdot \\ & \cdot B_{[(s''_{N_1}, s''_{N_2}), \lambda_1, \lambda_2]}^{(\Lambda)} \cdot \min\{\lambda_2 + (\lambda_1 - \min\{\sigma, \lambda_1\}), K - s'_{Q_2}\} \end{aligned} \quad (4.28)$$

To show how the proposed multi-layer offloading platform works, in the following an use case is presented with some numerical results. As already said, only two wireless cells are considered and is assumed that, at most, six vehicles can stay simultaneously in each cell. This means that  $N_{MAX1} = N_{MAX2} = 6$ .

About the vehicle arrival process in the first cell, these two matrices are defined as input parameters for the SBBP  $\Omega(n)$ :

$$P^{(\Omega)} = \begin{bmatrix} 0.864 & 0.136 \\ 0.143 & 0.857 \end{bmatrix} \quad (4.29)$$

$$B^{(\Omega)} = \begin{bmatrix} 0.02 & 0.03 & 0.04 & 0.10 & 0.43 & 0.18 & 0.20 \\ 0.02 & 0.02 & 0.01 & 0.06 & 0.26 & 0.24 & 0.40 \end{bmatrix} \quad (4.30)$$

Regarding the transition of the vehicles from the first cell to the second, and from the second cell to outside, these matrix are considered:

$$P^{(\mu^1)} = \begin{bmatrix} 0.95 & 0.05 \\ 0.25 & 0.75 \end{bmatrix} P^{(\mu^2)} = \begin{bmatrix} 0.85 & 0.15 \\ 0.17 & 0.83 \end{bmatrix} \quad (4.31)$$

$$B^{(\mu^1)} = \begin{bmatrix} 0.19 & 0.52 & 0.08 & 0.06 & 0.09 & 0.03 & 0.03 \\ 0.01 & 0.43 & 0.44 & 0.05 & 0.04 & 0.02 & 0.01 \end{bmatrix}$$

$$B^{(\mu^2)} = \begin{bmatrix} 0.07 & 0.15 & 0.10 & 0.18 & 0.25 & 0.23 & 0.02 \\ 0.01 & 0.02 & 0.11 & 0.07 & 0.24 & 0.33 & 0.22 \end{bmatrix} \quad (4.32)$$

In the Vehicular Domain, each vehicle generates one job in a slot with a probability  $p_V = 0.9$ , the maximum length of the local queue is  $K_V = 4$  and the probability of serving one job with the local computing station is equal to  $\zeta_V = 0.5$ . In the MEC Domain, each MEC Server is equipped with  $L = 4$  CEs and has a queue length equal to  $K = 6$ . The chosen discount factor for the equation 4.23 is  $\gamma = 0.8$ . The MEC Domain applies a job offloading price  $\Theta_{V \rightarrow M}^{(OL)} = 0.2$  PUs, where PU is the unit of price, for job coming from the Vehicular Domain, while sets a price  $\Theta_{M \rightarrow M}^{(OL)} = 3$  PU to offload job from a MEC server to the other.

To evaluate the performance of this platform, three different strategies have been applied by the MPS, each one characterized by particular values of  $c_1$ ,  $c_2$  and  $c_3$  in the rewart function  $F_{RW}^{(MPS)}$ :

- $MS_1$ , sets the same weights for the three terms, that is,  $c_1 = c_2 = c_3 = 1$ ;
- $MS_2$ , provides a little more importance to the mean delay, by using  $c_2 = 2$ , while  $c_1 = c_3 = 1$ ;
- $MS_3$ , that gives a strong importance to the gained revenue and again a little more importance to the delay, by using  $c_1 = 1, c_2 = 3, c_3 = 6$ .

Regarding the Vehicular Domain, two different strategies, each characterized by a different set of weights in the reward function  $F_{RW}^{(VPS)}$ , were considered: the first,  $VS_1$ , with  $\gamma_1 = \gamma_2 = 1$ , and the second,  $VS_2$ , with  $\gamma_1 = 1$  and  $\gamma_2 = 2$ .

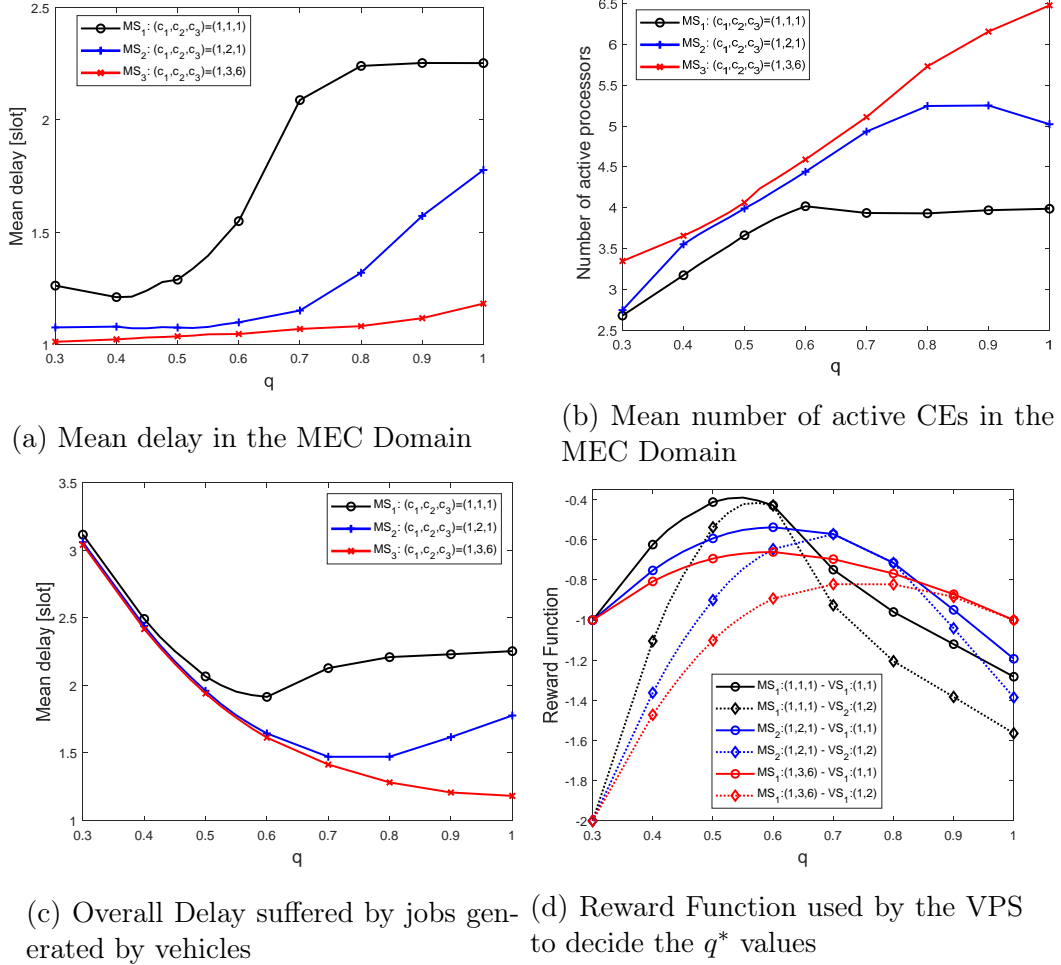


Figure 4.18: Multi-layer offloading in a Vehicular Network: numerical results

Using a Matlab-based tool, the MDP  $\Sigma$  was solved, calculating the best policy  $\rho^*$ .

After that, in Fig. 4.18a, the mean delay  $\bar{\delta}_M$  suffered by jobs offloaded to the MEC Domain, for the three different strategies and with the offload rate  $q \in [0.3, 1]$  range is shown. As expected, an increasing trend against  $q$  is observed for all the curves since the mean job arrival rate to the MEC Domain increases with  $q$ . The increasing trend is more accentuated for the  $MS_1$  strategy with which the mean delay is taken into consideration less than the other two strategies. On the contrary, with the third strategy,  $MS_3$ , which takes extra care of the delay, there is the best delay performance. However, as shown in Fig. 4.18b,  $MS_1$  maintains the mean number of active processors low, while the other strategies achieve the best performance by activating a higher number of processors.

Fig. 4.18c shows the mean delay  $\bar{\phi}$  suffered by all the jobs generated from the Vehicular Domain, obtained by averaging  $\bar{\delta}_M$  and  $\bar{\phi}_L$ . There is a non-monotonic

behavior for the curves calculated for the  $MS_1$  and  $MS_2$  strategies because, for low values of  $q$ , the delay is high since the most of traffic overloads local vehicular resources, while high values of  $q$  give higher delays as shown in Fig. 4.18a.

Finally, the reward function used by the VPS to decide the best value of of-flooding rate,  $q^*$ , is shown in Fig. 4.18d, for all the strategies applied by the MPS and the VPS. More specifically, knowing which is the strategy used by the MPS, the VPS decides the  $q$  value that maximizes the function  $F_{RW}^{(VPS)}$  for the specific strategy it is using.

# Chapter 5

## 5G-enabled smart services

During the PhD course, thanks to the participation in European projects, in collaboration with the National Inter-University Consortium for Telecommunications (CNIT), vEyes and Healthcare Technology Lab (HTLAB), it was possible to work on the main issues typical of the 5G network, not limiting the discussion to only the theoretical aspects, but also being able to address them from a practical point of view. Topics such as the softwarization of functions and applications, orchestration of resources and positioning of applications at the edge of the network, were the basis of the proposed projects and subsequent implementations. This also made it possible to carry out performance measurements on 5G networks implemented by the consortia in charge of the projects and analyze the benefits that this technology will provide to future Smart Services in a smart city context.

Each of the proposed projects represents a typical example of smart services that could be deployed in a smart city context and that, thank to the functionality of the 5G network, will be able to express their full potential. The possibility to test them in 5G platforms developed by the different consortia, allowed discovering both the possible limits of the proposed projects and the requirements that 5G has to guarantee to allow deploying of smart services.

In the following sections, each project will be described, with an overview of the platform in which it was tested and, in two cases, the results of the final measurement campaigns. More in deep, in Section 5.1, the DiMoViS framework is presented and the underlying Triangle platform described. Sections 5.2 and 5.3 present TouristEyes and 5Gamer, two frameworks realized in the context of the 5GINFIRE Project. Finally, Section 5.4 describes VISION, a framework proposed for the Flame Project and currently in its final phase.

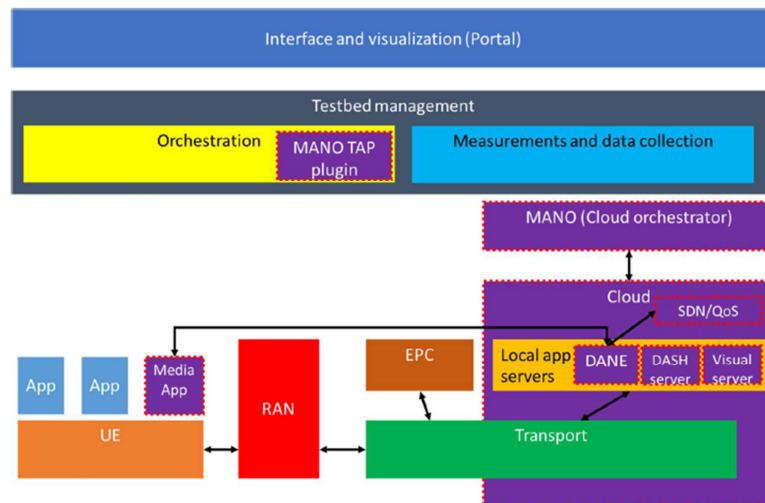


Figure 5.1: High level architecture of the Triangle testbed [81]

## 5.1 DiMoViS: Distributed Mobile Video Surveillance System

The *DiMoViS* framework provides customers of a Telco Operator (TO) with a video surveillance service both in mobility and at home. It consists of the virtualization of a video-surveillance set-top box (vsSTB), allowing the end-user to receive video flow both on smartphone and DLNA compliant devices at home. It was implemented in the context of the Triangle Project. Using the Triangle testbed, it was possible to achieve measurements of DiMoViS and this work was presented in [80], accepted at 2019 IEEE International Symposium on Measurements and Networking.

In Subsection 5.1.1, the architecture of the 5G testbed, realized by Triangle Consortium, is described. Next, in Subsection 5.1.2 the implementation of DiMoViS and its performance evaluation are reported.

### 5.1.1 Triangle Testbed

The Triangle Project was an European Future Horizon2020 project [81]. Its main objective was to promote the testing and benchmarking of mobile applications and devices in Europe and also provides a path towards certification of qualified mobile developments. Triangle's testbed is the result of studies that have identified the requirements to providing testing means to increase the quality and reliability of mobile communications and applications across multiple domains, with the aim of deploying an e2e test architecture.

An overview of the Triangle testing architecture is shown in Fig. 5.1. The entry

point to the testbed was the Triangle Portal, a web interface offered to the end users, designed to adapt to the requests of the latter. The complexity of testing was contained in high-level scenarios that represented the network conditions in terms of cell power, channel model, noise/interference, uplink/downlink bandwidth, modulation and so on, and prevented users from having to manage the entire series of these configurable parameters. Typical high-level scenarios were Urban, SubUrban, HighSpeedTrain, and their subcases. Based on the chosen scenarios, the *Orchestration block* configured the physical components, scheduled the execution of tests and managed the *Measurements and data collection block* to execute and storage the required measurements, with the aim of verifying the performance of the applications and devices under test. The testbed provided detailed reports to the user, based on information gathered by the reporting and automation tools running on both the testbed and the device. The Orchestration block implemented in the testbed was the Keysight's Test Automation Platform (TAP). Each component of the testbed was controlled via a TAP driver, which acted as a bridge between the TAP engine and the interface of the real component. One of the most important plugin inside the Orchestration was the MANO TAP plugin, that allowed the Orchestration block to interact with the MANO Cloud orchestrator. In particular, one of the project partner, TNO organisation, realized and integrated in the original testbed, a cloud environment based on VIM Openstack (to allow the deployment of virtual functions) and Open Source MANO (to manage the life cycle of the deployed functions). An important role, during the execution of the experiments, was played by the Quamotion's automation tool [82]. Quamotion is a test automation framework for use with native, hybrid and mobile web apps. Quamotion WebDriver is able to automate the actions of users, allowing to manage the entire life cycle of the app (installation, start and execution) using the WebDriver protocol. It automates apps on real devices (iOS, Android), simulators (iOS) and emulators (Android). Only a valid application package (apk or ipa file) and, in the case of iOS, a valid DeveloperProfile and an iOS DeveloperDisks, are needed. Furthermore, a license needs to be requested and uploaded in order to use all Quamotion features. The Quamotion WebDriver design follows the W3C Web Driver specifications. Quamotion WebDriver is an extension of the Web Driver specifications, and adds specific support for managing mobile devices and mobile applications. Scripts can be written in multiple languages (PowerShell, Java, C#). More details on the steps to be performed in order to create a valid app user flow using Quamotion, are present in the paper [83], presented at the 15th Internation Symposium on Wireless Communication

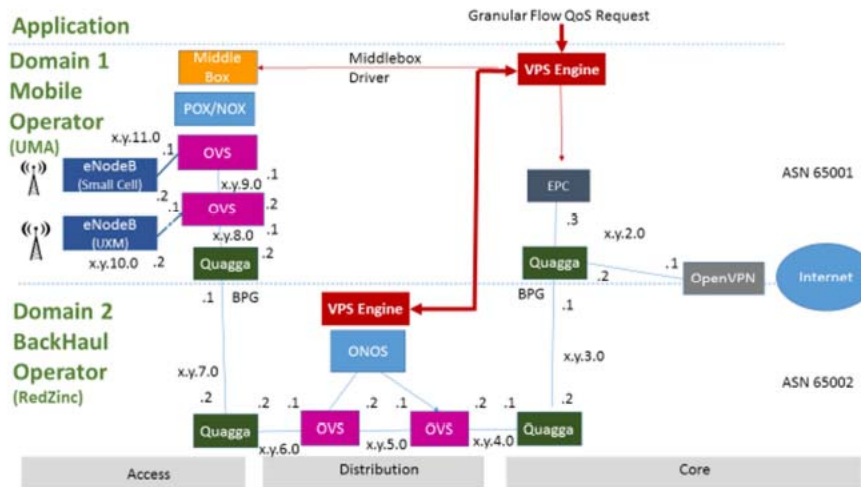


Figure 5.2: Transport layer in the Triangle testbed [81]

Systems in 2018.

About the Measurements and data collection block, the testbed provided several probes to extract the required measurements: software probes running in the UE that included AT4 Agents and the TestIDroid tool of UMA, while hardware probes included a power analyzer connected to the UE to measure energy consumption. Triangle gave the possibility to app developers to use an instrumentation library in order to perform additional measurements.

The RAN obviously played a major role in the Triangle testbed. It was provided by Keysight’s UXM Wireless Test Set, a mobile network emulator. Some of its key features for testing included Inter Cell Interference Coordination (eICIC) schemes, WLAN offloading, IMS/e2e VoLTE communications between multiple devices. This Keysight emulator was capable of emulating a 2G, 3G, 4G and NB-IOT base station. In order to provide an e2e system, an EPC by Polaris Networks was integrated into the testbed, with the main elements of the standard core network: Mobility Management Entity (MME), Security Gateway (SGW), Packet Data Network Gateway (PGW), Home Subscriber Server (HSS), and Policy and Charging Rules Function (PCRF). The EPC also includes the EPDG (Evolved Packet Data Gateway) and ANDSF (Access Network Discovery and Selection Function) components.

The transport layer was composed by three network domains, connected each other through a virtualized routing environment managed by SDN elements: RAN or Access network, Distribution Network and Core Network. The interconnections between elements of the three domains was provided by three components: OpenvSwitch, Quagga (a software router) and ONOS (a SDN network orchestrator).



### 5.1.2 DiMoViS architecture and performance evaluation

The DiMoViS framework has three main capabilities:

- real-time streaming of video flows captured by IP cameras installed on geographically remote sites to either a smartphone or a smart TV at home;
- recording of video flows;
- playout of recorded video flows.

The main difference from the plethora of today video surveillance systems is that these last ones send video streams to a centralized control station, while DiMoViS allows to have a distributed control station. Users can decide dynamically the video cameras they are interested to, and each IP camera can be selected by all the users that are authorized to use it. Video flows can be received both in mobility on a smartphone or a tablet, or by any DLNA compliant device, like a smart TV, at home. The underlying network infrastructure allows a multipoint-to-multipoint communication and the deployment of the service chains to realize the service.

The vsSTB is constituted by four main parts: *UE*, *Video Players*, *vsSTB Engine* and *vsSTB Shared Chain*.

The UE provides the user with the following functionalities:

- *Remote Controller*: it allows the user to remotely control the vsSTB. It enables access to a selected IP camera for live transmission, or to a recorded video flow from his personal storage. Moreover, it allows to create virtual connections between the vsSTB and the DLNA-compliant smart TV in the user home network;
- *Mobile Player*: it allows the user to watch live or stored video flows directly on a smartphone or tablet.

The Video Players are the devices where watching video flows coming from IP cameras or the video library of registered flows. They can be either a mobile user device (smartphone or tablet), or a DLNA-compliant smart TV.

The vsSTB Engine, whose service chain is represented in Fig. 5.3, implements a personal instance of the vsSTB. An instance of the vsSTB Engine service chain is deployed for each customer of the DiMoViS service. The vsSTB Engine is the core element of the vsSTB, and represents the virtualization of the physical STB device for video surveillance; it is composed by four Virtual Functions (VFs):

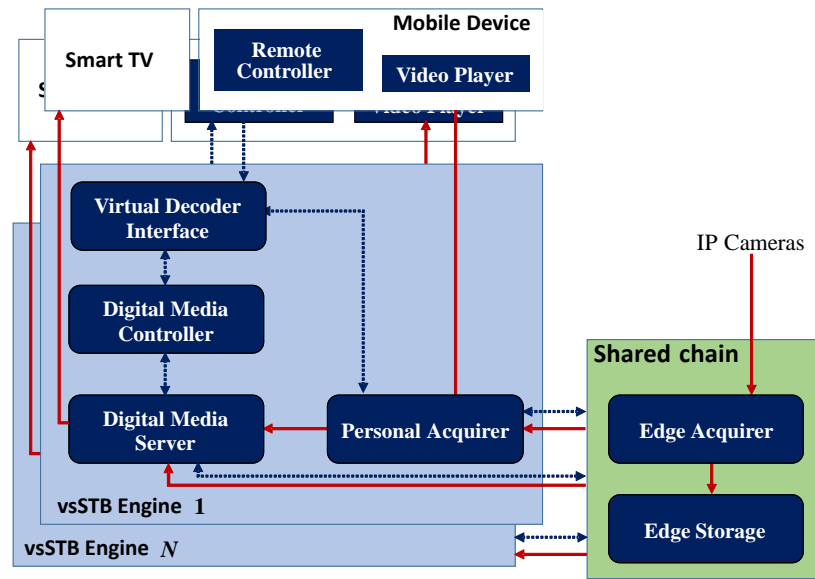


Figure 5.3: The vsSTB service chain

- *Virtual Decoder Interface* (VDI): it represents the interface between the vsSTB app running on the user mobile device and the vsSTB Engine. The VDI communicates with the Digital Media Controller to select the player which reproduce the video flows, and with the Personal Acquirer, with the objective of either selecting one of the available contents to be played out (live or recorded) or enabling the recording of a selected event;
- *Digital Media Controller* (DMC): according to the DLNA standard, it maintains the list of the DLNA players that are active in the user home network, and communicates with the Digital Media Server to know the list of the available video contents (both the live streams from IP cameras or the recorded ones);
- *Digital Media Server* (DMS): it is DLNA-compliant, too; it maintains and updates the list of the available video flows exposed by both the Personal Acquirer (live streams coming from IP cameras) and the Edge Storage where the recorded video flows are stored;
- *Personal Acquirer* (PA): it receives commands from the VDI to enable the live streaming from an IP camera or the recording of a selected flow. It communicates with the Edge Acquirer in the shared chain, requiring a video flow from a specific IP camera selected by the user or indicating that a specified flow has to be forwarded to the Edge Storage to be recorded.

The vsSTB Shared Chain is a service chain implementing services shared among

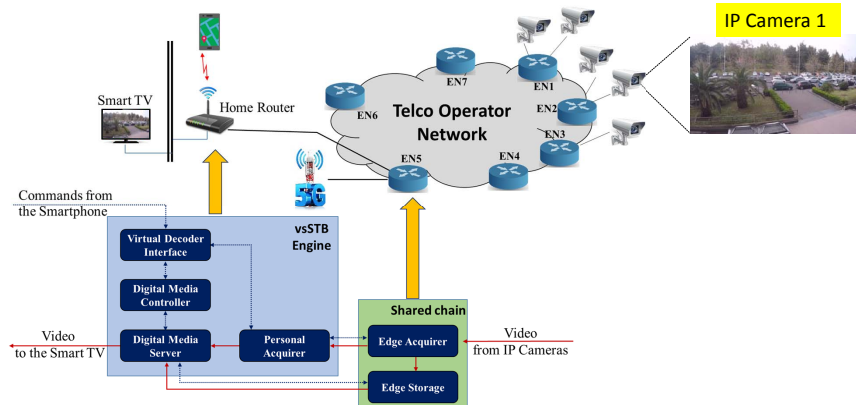


Figure 5.4: DiMoViS deployment when user is at home

different users, specifically, the users accessing the network through the edge node whose MEC facilities are supporting this service chain. The vsSTB Shared Chain is constituted by the following VFs:

- *Edge Storage* (ES): it is in charge of saving the recorded video flows;
- *Edge Acquirer* (EA): it receives video flows from the cameras and replicates each of them to the associated user’s PA.

In Fig. 5.3 the internal interactions among VFs are shown. More specifically, continuous lines (red) represent video flow transmissions, while dotted lines (blue) indicate signaling communications among entities composing the chains.

As said before, there are two most relevant scenarios for the DiMoViS platform: service access from devices at home and service access in mobility. In the first scenario the user mobile device accesses the Telco Operator network from the user home network, that is, flows generated by the user mobile device enters the network through the softwarized router of the home network. In this case, the service chain deployment is realized as shown in Fig. 5.4. In this case, in order to achieve low latency in accessing the DiMoViS service, the vsSTB Engine is deployed on the Home Router, leveraging on its MEC capabilities. Moreover, the vsSTB Engine is connected to the vsSTB Shared Chain deployed on the edge node with MEC facilities used as ingress node for the considered home network.

The second scenario, on the other hand, is characterized by the user in mobility. In this case, his mobile device accesses the Internet through a 4G/5G cellular connection towards a RAN of the Telco Operator. The service chain deployment for this scenario is shown in Fig. 5.5. With the aim of achieving also in this case low latency in the interaction between the vsSTB app on the mobile device and the video surveillance Server platform, the vsSTB Engine is deployed on the same edge

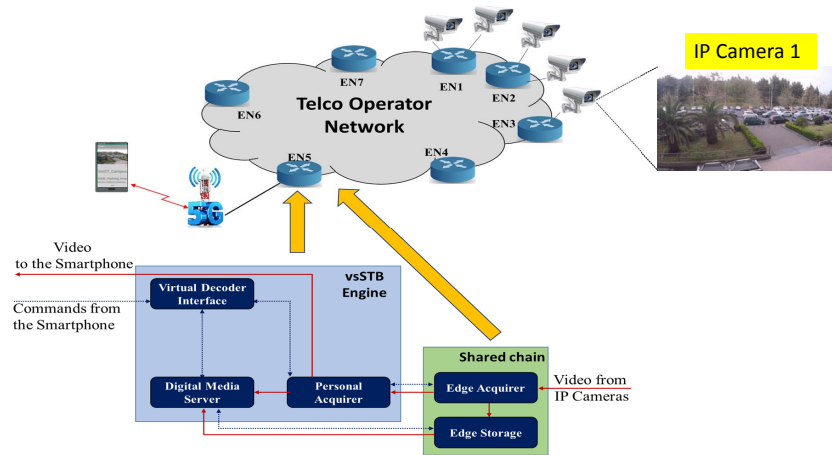


Figure 5.5: DiMoViS deployment when user is in mobility

node used as entrance node, leveraging on its MEC facilities. The vsSTB Engine is connected to the vsSTB Shared Chain deployed on the same edge node.

The scenario considered during the testing and measurements phases is the second described before. As already said, in the Triangle testbed different high-level scenarios are implemented to change the configuration of the network during an experiment. In this case, the “Urban-Office” scenario was used to test. “Urban-Office” scenarios are characterized by a very dense urban network, in which in addition to the macro sites at a reduced distance from each other, it is also possible to find small cells. This dense structure arises from the need to transport a high amount of traffic. In this scenario, users will be static, with internal network access points that can be either WiFi or small cells. The characteristics of the channel and the level of interference experienced will also depend on the signals coming from the external macro cells.

As already said at the beginning of this section, after the implementation of DiMoViS inside the Triangle platform, some measurement campaigns were conducted to evaluate the performance of the proposed framework. The main Key Performance Indicators (KPI) to evaluate performance of the DiMoViS vsSTB were defined at three levels: the network level, in terms of mean delay, the application level, in terms of dependence of the frame rate degradation on the VFs placement in the NFVI-PoP, and at the RAN level in terms of RSRP, Received Signal Strength Indication (RSSI), RX and TX Data Rate. All these indicators directly affect the quality of experience perceived by the end user.

About the network level, the crossing delay through the two VFs of the vsSTB chain, PA and EA, was monitored. Two Wireshark-based monitoring points, each at the output point of the above VMs, were placed (Fig. 5.6) and a Linux stress

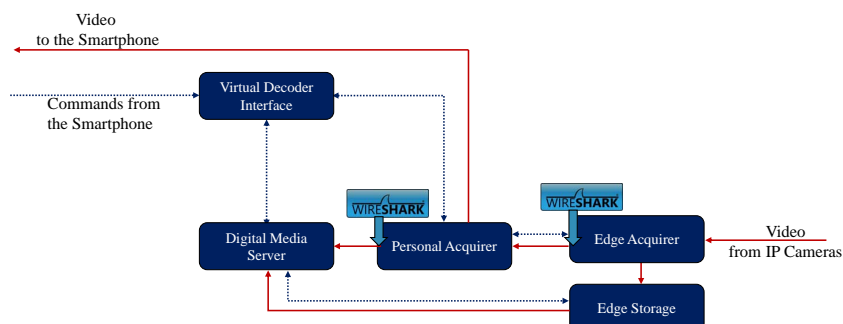


Figure 5.6: DiMoViS in Triangle. Network level measurement scenario

Delay to cross the Personal Acquirer		
	No background computational load	High background computational load
Minimum Delay [ms]	0,137	0,25323
Mean Delay [ms]	1,275709	4,135432
Maximum Delay [ms]	11,276	21,07634

Table 5.1: DiMoViS in Triangle. Delay in the Personal Acquirer

tool was used in order to determine the impact of the background computational load of each host. This tool was run with the two options “cpu” and “io”. The first option runs N parallel threads that perform mathematical calculation of the  $\text{sqrt}()$  function. Instead, the second option has been used to generate an amount of computational load. Combining the two options the case of “High background computational load” was considered and compared with the case of “No background computational load”. Results in terms of minimum, medium and maximum delay are summarized in Tables 5.1 and 5.2.

Delay to cross the Edge Acquirer		
	No background computational load	High background computational load
Minimum Delay [ms]	0,003823	0,004912
Mean Delay [ms]	0,02578434	0,053264
Maximum Delay [ms]	0,089943	0,149145

Table 5.2: DiMoViS in Triangle. Delay in the Edge Acquirer

In the second set of measurements, the impact of the positioning of the VFs composing the vsSTB service was evaluated. To this purpose, the network topology shown in Fig. 5.7 was considered, where a video surveillance camera provider shares its IP cameras providing the related video streams to the requesting users. In Fig. 5.7, there are three NFVI-PoP nodes, associated to three different edge

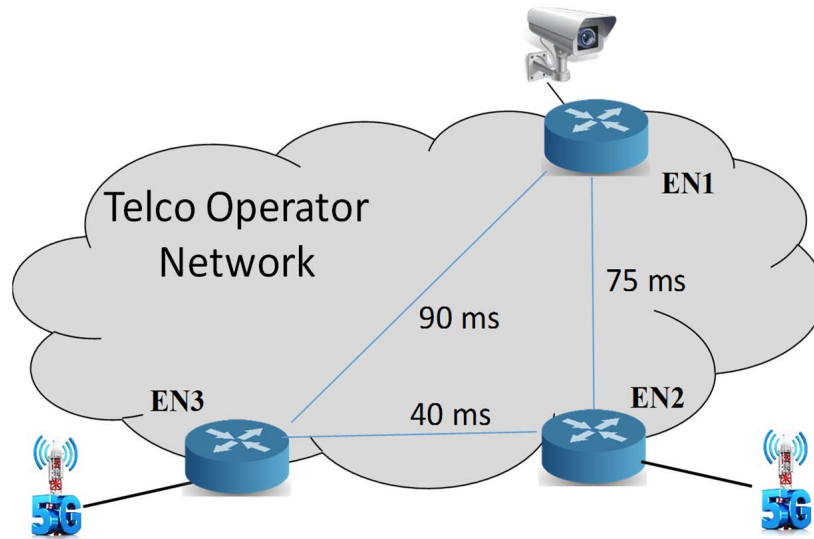


Figure 5.7: DiMoViS in Triangle. Application level measurement scenario

nodes (EN):

- EN1: the ingress node of the video surveillance camera provider;
- EN3: the ingress node of the mobile user at the beginning of the experiment;
- EN2; the ingress node of the mobile user at the end of the experiment.

Each link has been realized with a transmission bandwidth of 50 Mbit/s, while the link latency, also shown in Fig. 5.7, has been achieved by using VMs running the Linux Traffic Control tool, timely configured. The study has been carried out according to the following 5 steps:

- STEP 1: Initial configuration of the experiment, with the user accessing the network through the edge node EN3, and all the VFs placed on the same node (Fig. 5.8a);
- STEP 2: The user has changed his access node, now being the node EN2, but all the VFs composing the vsSTB are still on EN3 (Fig. 5.8b);
- STEP 3: The VFs VDI and DMS deployed in the edge node EN2 substitute the ones deployed in EN3, according to the new position of the user (Fig. 5.8c);
- STEP 4: Also the VF PA in EN3 is substituted with the one in EN2 (Fig. 5.8d);

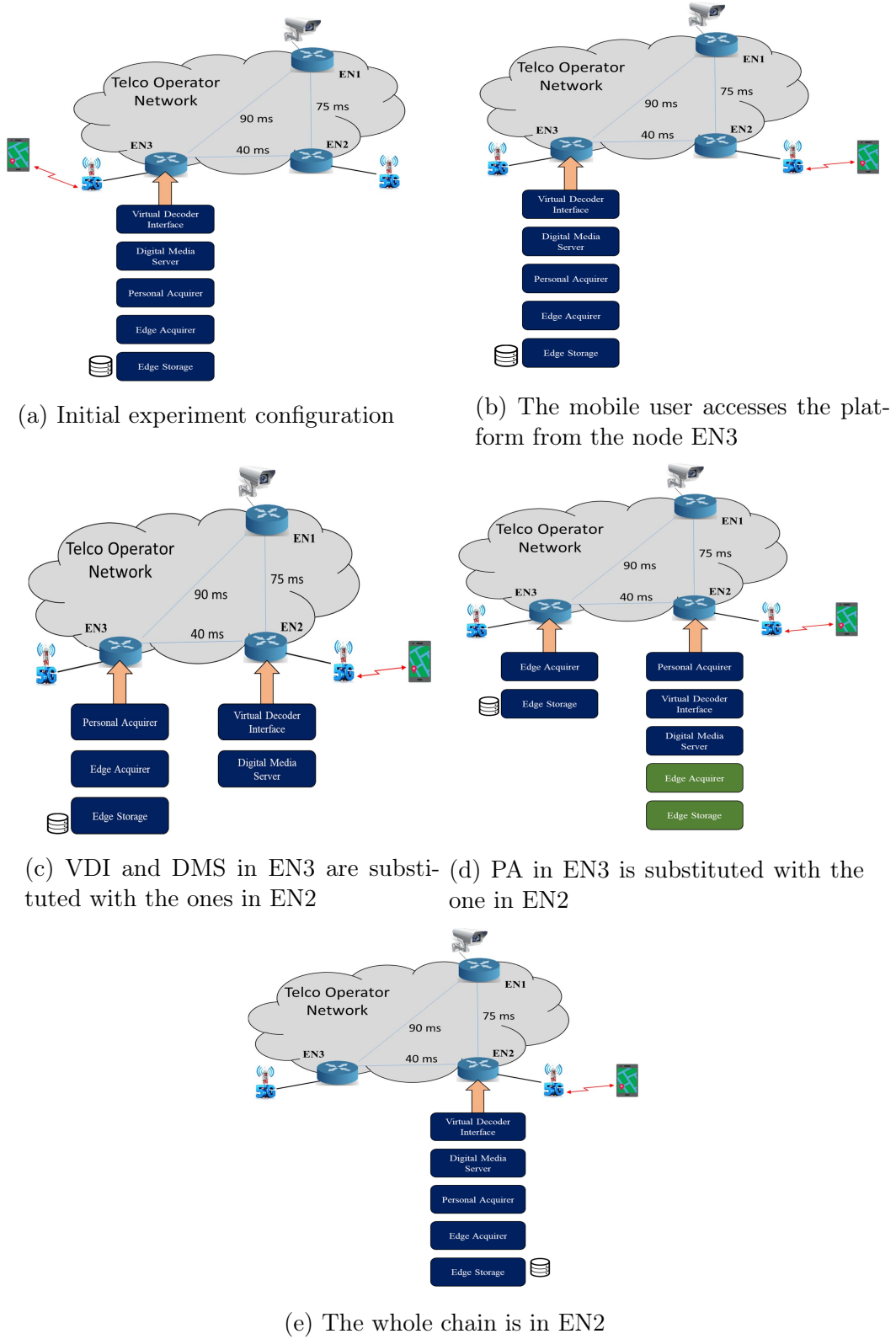


Figure 5.8: DiMoViS in Triangle. VFs migration in 5 steps

User position	Latency Mean Value [s]	95% Confidence Interval [s]
Step1	1.16	0.51
Step2	1.19	0.52
Step3	1.30	0.57
Step4	1.28	0.56
Step5	1.27	0.56

Table 5.3: DiMoViS in Triangle. Latency due to IP Camera changing events during live video streaming

- STEP 5: The whole vsSTB is deployed on EN2, very close to the user (Fig. 5.8e),

In order to evaluate the impact of the position of the VFs on the perceived performance, two different ways to access and use the DiMoViS vsSTB were considered:

- watching live video streams from IP cameras transmitted from a remote IP camera provider; in this scenario, the user changes video stream, jumping from one IP camera to another one;
- playout of a pre-recorded event; in this scenario, the user performs the action of time jumping on the seek bar to change the playout instant of the event being watched.

In both the above cases, some frames can be lost, causing a QoE degradation, in the first case while waiting for the new video stream, in the second case while waiting video related to the new playout point. Frame rate reduction during both the events of changing IP camera and jumping on the seek bar will be considered as KPI.

To this purpose, by using Wireshark, the time occurred between the transmission of the first packet sent by the smartphone to start the action, and the last packet received by the smartphone reporting the successful action completion, was measured. In the presented analysis, the same action has been repeated for twenty times in order to estimate the mean value and the standard deviation of the latencies during the total measurement periods, so providing results with a 95% confidence interval. Numerical results are listed in Tables 5.3 and 5.4.

Considering again the network topology shown in Fig. 5.7, the next step was to analyze the frame rate degradation due to IP Camera changing events during live video streaming and Time-jumping events on the seek bar during an event video playout. Two cases were compared, according to the above list: the first



User position	Latency Mean Value [s]	95% Confidence Interval [s]
Step1	1.36	0.510.38
Step2	395.24	71.75
Step4	383.65	68.24
Step5	1.33	0.37

Table 5.4: DiMoViS in Triangle. Latency due to time jumping on the seek bar

one at the STEP 2 and the second one at the STEP 5. Frame rate degradation is reported in Fig. 5.9 and 5.10 as a function of the interval between two consecutive IP camera change events.

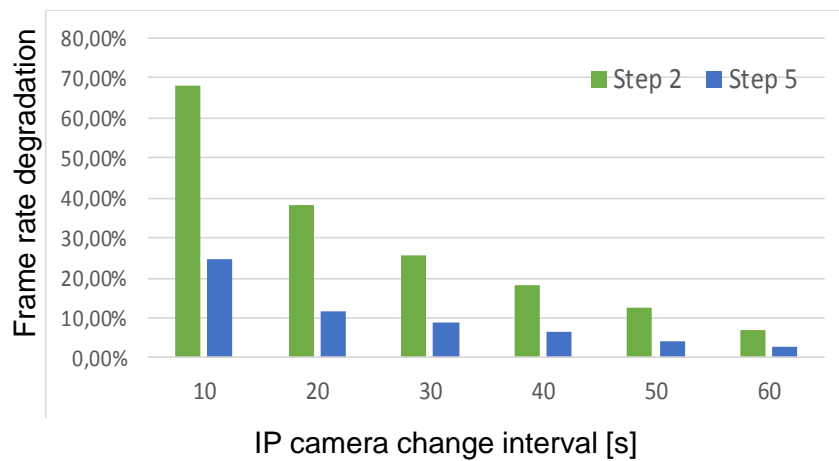


Figure 5.9: DiMoViS in Triangle. Frame rate degradation when IP Camera changing events during live video streaming

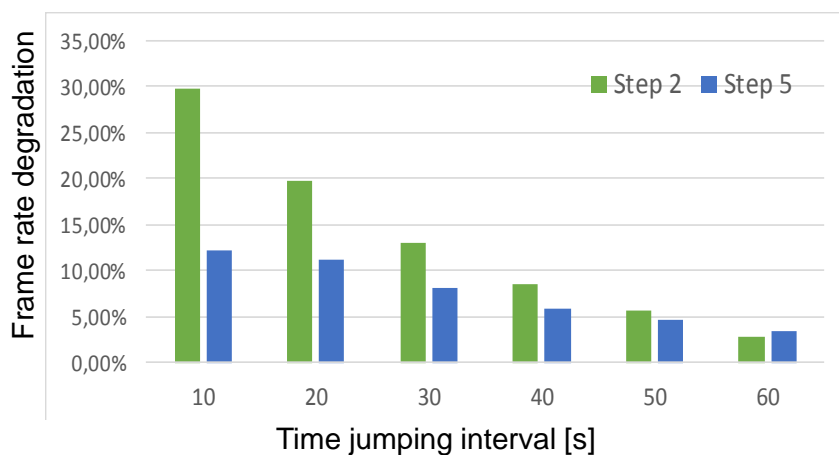


Figure 5.10: DiMoViS in Triangle. Frame rate vs time jumping on the seek bar during playout of a pre-recorded event

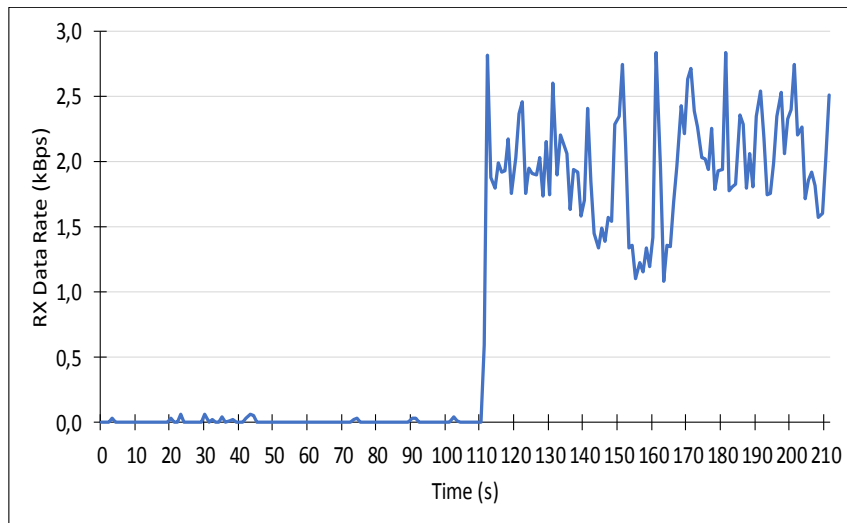


Figure 5.11: DiMoViS in Triangle. RX Data Rate

Finally, it was possible to test also the RAN of the Triangle platform. The duration of the experiment was 210 seconds, during which the vsSTB app started, the Provider and one of its IP cameras were chosen, and finally the video was displayed on the UE. The measures regarded in particular :

- RX data rate and TX data rate (Fig. 5.11 and 5.12);
- RSRP and RSSI (Fig. 5.13).

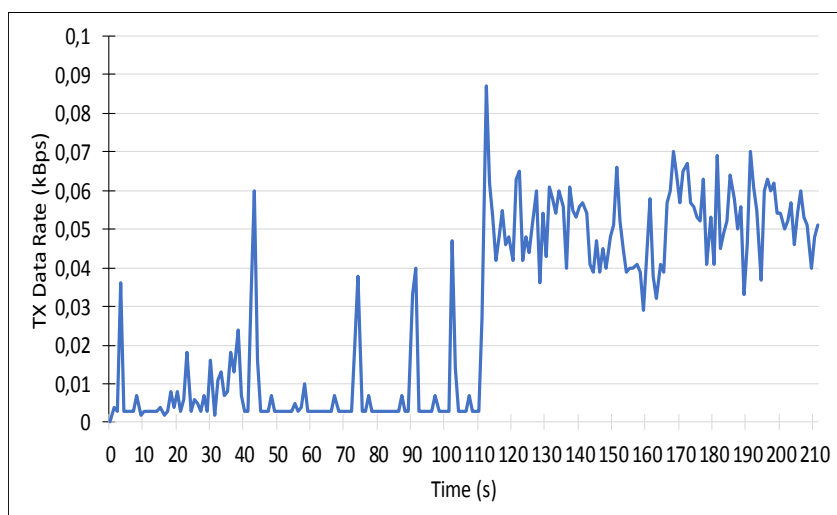


Figure 5.12: DiMoViS in Triangle. TX Data Rate

Fig. 5.11 shows the downstream (RX) data rate, while in Fig. 5.12 the upstream (TX) data rate is represented. In the first one hundred and ten seconds, the RX

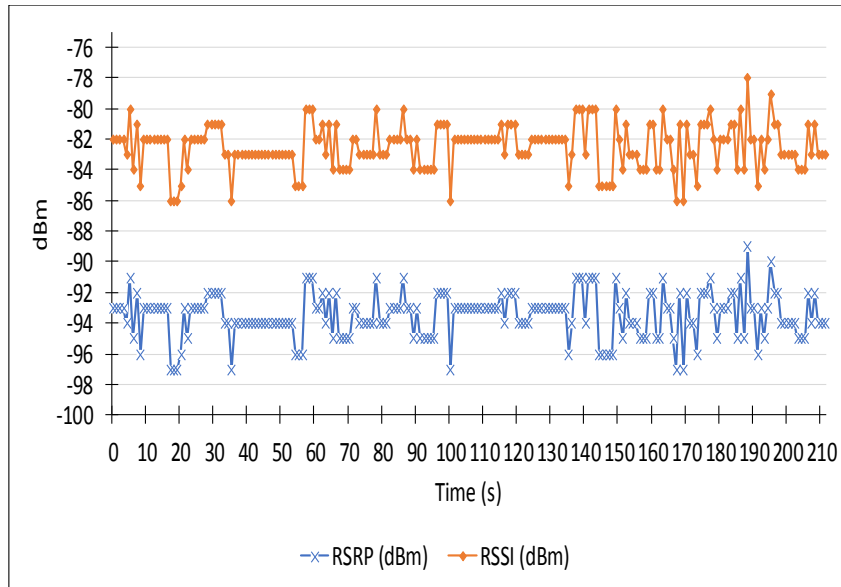


Figure 5.13: DiMoViS in Triangle. RSRP and RSSI values

data rate is very low, because there is only a few information that arrive from the RAN to the smartphone, information that regards only the list of Providers in the system and, after the user selects one Provider, the list of IP cameras that it controls. Starting from one hundred and ten seconds, the RX data rate is increased because the video streaming flow begins to arrive in the user device.

The TX data rate has a behavior similar to the RX data rate. Also in this case, in the first one hundred and ten seconds TX data rate takes on low values but, as shown in Fig. 5.12, there are some instances in which it has peak values due to the transmission of information from the UE to the network. Starting from one hundred and ten seconds the traffic increases due to ACK messages that the app sends to the network. In fact, to receive the video flow, an HTTP connection is established between the UE and PA VM. The HTTP protocol uses the TCP at the transport layer, which requires the use of ACK messages to confirm the receipt of data.

In Fig. 5.13 it is possible to see the RSRP and RSSI values to evaluate performance on the radio channel. RSRP was already described in Chapter 3 while RSSI is defined as the linear average of the total received power observed only in OFDM symbols carrying reference symbols by UE from all sources, including co-channel non-serving and serving cells, adjacent channel interference and thermal noise.

For RSRP, to evaluate signal quality, if it is in the interval  $[-65, -80]$  dBm, the signal is good, while if it is in the range  $[-80, -95]$  dBm, the signal is satisfactory.

As depicted in Fig. 5.13, in the considered scenario, RSRP values are between -89 dBm and -97 dBm, so signal level can be considered satisfactory. For RSSI, a good channel is characterized by higher values than RSRP, of at least 25 dB. In the figure, it is possible to observe that RSSI values are 11 dBm higher than RSRP, so this confirms the satisfactory level of the signal.

## 5.2 TouristEyes: a 5G-based platform for blind Tourist

Blind people encounter enormous difficulties in visiting unknown places, even if they have traditional supports like a white stick and a guide dog (if they are not allergic to animal), or some Tactile Ground Surface Indicators installed in the visited places. A possible solution that has been adopted in the last few years is to run a GPS-based application on the smartphone of the blind person; however, this does not give a sufficiently high-precision (to guarantee safety for the blind person when he/she moves in the city) and does not work indoor. In addition, orientation and point-of-interest (PoI) finding result very difficult operations with the currently available technologies.

The *Tourist Eyes* platform has the objective of providing blind tourists, visiting a smart city, with a framework for supporting their activities and demonstrating how 5G technologies are enablers of everyday life services also for impaired persons. Tourist Eyes was realized in the context of the 5GINFIRE Project in collaboration with vEyes, a non-profit organization whose mission is to realize assistive technologies for blind people.

5GINFIRE was a three years Research and Innovation action/project under the EU programme Horizon 2020 [84]. The main 5GINFIRE objective was to build and operate an Open and Extensible 5G NFV-based Reference (Open5G-NFV) ecosystem of Experimental Facilities that not only integrates existing FIRE facilities with new vertical-specific but also lays down the foundations for instantiating fully softwarised architectures of vertical industries and experimenting with them. In particular, to execute the experiments, the infrastructure provided by the University of Bristol replica site was used.

Tourist Eyes is an enhancement of Poseidon 2.0 [85] testbed, developed by vEyes, but with the possibility of using 5G technology to work with more complex scenarios in a smart city. The advantages of that testbed with respect to the current technologies are evident because it provides a very useful support to blind tourists, both outdoor, when guide dogs are not available, and indoor, where GPS

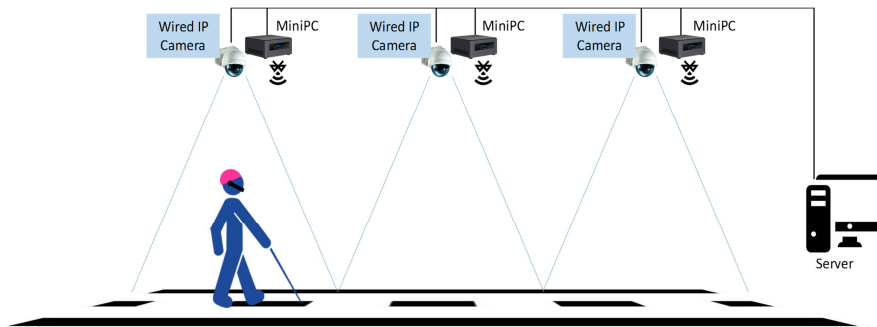


Figure 5.14: Tourist Eyes. State of the art

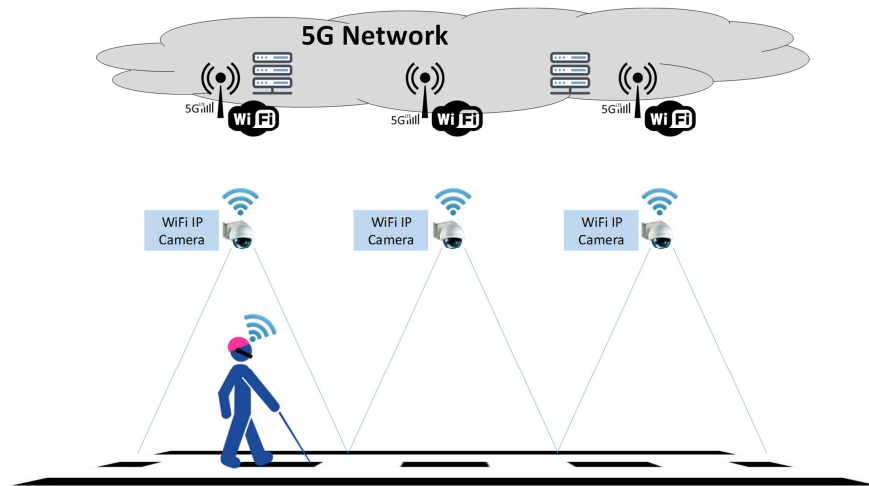


Figure 5.15: Tourist Eyes. New testbed

does not work. Now, considering this testbed as the state-of-the-art system (Fig. 5.14), a comparison between it and the Tourist Eyes framework (Fig. 5.15) shows the advancements achieved through the proposed experiment.

In particular:

- deployment of the Tourist Eyes framework in a smart city is strongly simplified. Furthermore, no cabling, no hardware for computing, and no wearable devices are needed. IP cameras can be easily placed anywhere and can also be added when the service is already running. Each IP camera will be easily connected to the platform automatically thanks to the WiFi/LTE/5G coverage, and their flows will be automatically redirected to the desired chain thanks to the SDN infrastructure. Users, i.e. blind tourists, can join the platform very easily because it is only needed that they have the app installed on their smartphone, and wear a coloured hat or T-shirt, in such a way he can be tracked by the platform;

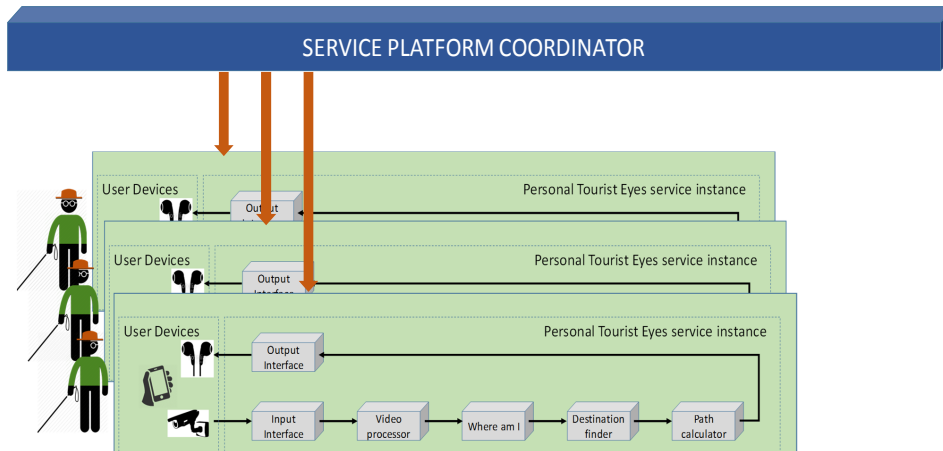


Figure 5.16: Tourist Eyes architecture

- thanks to the underlying softwarization paradigms (SDN/NFV/MEC), the Tourist Eyes service can be extended or easily integrated with other services.

To describe the Tourist Eyes platform, it is possible to refer to Fig. 5.16. For each blind person, an instance of the Tourist Eyes Service is created and connected to the Service Platform Coordinator that:

- manages the database of user and the database of IPcams;
- coordinates all the instances of the personal Tourist Eyes service;
- manages the database of service users and the database of IPcams.

Thanks to the SDN/NFV paradigm available in the Bristol testbed, the personal Tourist Eyes service is realized as a set of elementary VxFs, as shown in the bottom part of Fig. 5.16. In this way, each service component can be migrated from one NFVI-PoP node to another one, in order to follow the blind person in his movements, or can reside inside the same NFVI-PoP node.

When a blind tourist needs to be guided in a particular path, runs the Tourist Eye app on his/her smartphone thanks to the vocal assistant. In this way, the user will just have to log in to be associated to a Personal Tourist Eyes Service chain. Now the user has to specify his/her starting point and where he/she wants to go, so the system is aware of his/her presence in a specific point of the city (in the case of the experiment, the Millennium Square in Bristol), activates an instance of the Personal Tourist Eyes service chain and associates it to the requesting user. Blind users can also send voice commands to request help regarding some PoI, like restaurants, restrooms, museums, and ATM machines. The Personal

Tourist Eyes Service chain is then migrated to the MEC server at the closest edge node or otherwise run from the scratch. Execution of service chain instances is achieved thanks to the OSM orchestration platform. At this point, the blind tourist can start to move following the sound signals received by the application in the headphones connected to his/her smartphone. To support the complete path, a set of IP cameras need to be installed at a distance of approximately 10 m from each other, in such a way that the blind tourist is always monitored by one IP camera. The IP cameras are connected to the network access points (WiFi APs or LTE/5G) deployed in the city.

The software parts of the Tourist Eyes testbed are basically four: the *Tourist Eyes Console*, the *Tourist Eyes Manager*, the *Tourist Eyes Training App* and the *Tourist Eyes App*.

The Tourist Eyes Console (briefly Console) is implemented as a C# application and aims at simplifying the set-up process of the experiment by providing an intuitive graphical user interface that enables the creation of new routes and the configuration of the route parameters. Therefore, the Console drastically reduces the set-up time of the experiment while enabling the management of the system set-up without directly interacting with the database. The operations that must be performed to create and set the route parameters are two:

- insertion of a new camera in the list of the system cameras, specifying an ID, the IP address and port used to send video flow requests, and the username and password needed to access the camera in a security way;
- route creation and parameters setup: a route is made up of several modules, each one associated to one camera.

About the route creation, first of all it is necessary to select the starting PoI from which the user will be tracked the specific route. Then it is possible to configure the orientation, the direction and the message type of the Starting Module. The orientation can be both horizontal or vertical, the direction depends on the orientation and can be both right or left for horizontal modules and up or down for vertical modules. Finally, the camera associated to the module must be selected. The message type represents the audio signal that will be sent to the user when he/she crosses a specific guideline. There are two types of guidelines: yellow and red. The yellow lines delimit the virtual track in which the blind user must walk: when the user crosses one of these two lines, a message is sent from the Manager to prompt the user (e.g make a step on the side) to re-enter the route. The red lines define the prompted message that the Manager will send to

the user when he/she is approaching the next camera. In this way, the user is prompted to make a right/left turn or just to keep going straight. The next step is to set the position of the red and yellow lines. This is done by right-clicking on the module that has just been created and by selecting "show". In this way, the console shows the real time video streamed by the camera, thus facilitating the setting of guidelines position. Finally, it is necessary to specify the radius of the hat, which needs to be reconfigured for each camera, since each camera can be set at different heights w.r.t. the ground. It is also possible to set the radius for the umbrella, since it can be used as an option to the hat. This last step concludes the configuration of the starting module. As already said before, a route is made up of several modules; two of them must be the starting and the arrival module. A new module can be added to an existing route by clicking on one of the already-created modules and by selecting "add new module". The operations to be performed are those just described, with the only addition of having to specify whether the module just added is the last of the route (the arrival) or not. It is important to note that the outward and return journeys are created in two different routes since there are touristic sites like museums where the routes are unidirectional.

The Tourist Eyes Manager (briefly Manager) is the core of the entire system. After the system boot, the Manager listens for connections on a certain port. Through the Tourist Eyes App the user sends a request to the Manager to be tracked and guided to safely arrive at the desired destination. After selecting the starting point and the ending point from the Tourist Eyes app, a connection request is sent to the Manager. The Manager then executes a query in the database to retrieve the appropriate route (for each starting point and ending point there is only one route). Afterwards, the first two cameras of the route begin to send video streams to the Manager, as shown in Fig. 5.17: the tracking algorithm is performed on the first camera, while the second one is ready to welcome the blind user.

In this way, while the user leaves the first cam, the second one is already sending its video stream to the Manager, which then "shifts" its tracking algorithm from the video stream of the first camera to the stream of the second one while guaranteeing a soft handover procedure. When the blind user leaves the first camera and enters the second one, the first camera stops sending video streams and the third camera simultaneously starts its transmission. The tracking algorithm searches for a pseudo-spherical object of colour previously chosen by the user through the app; moreover, to reduce interference from external objects of the same colour



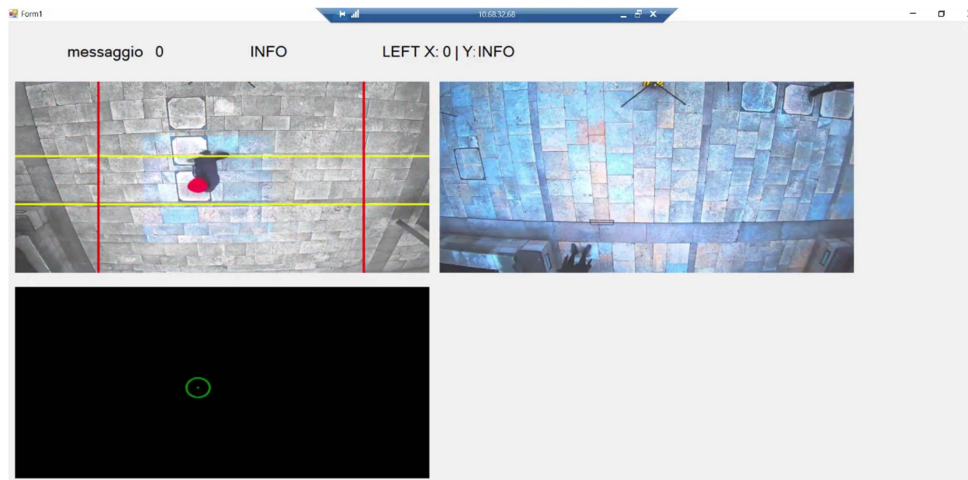
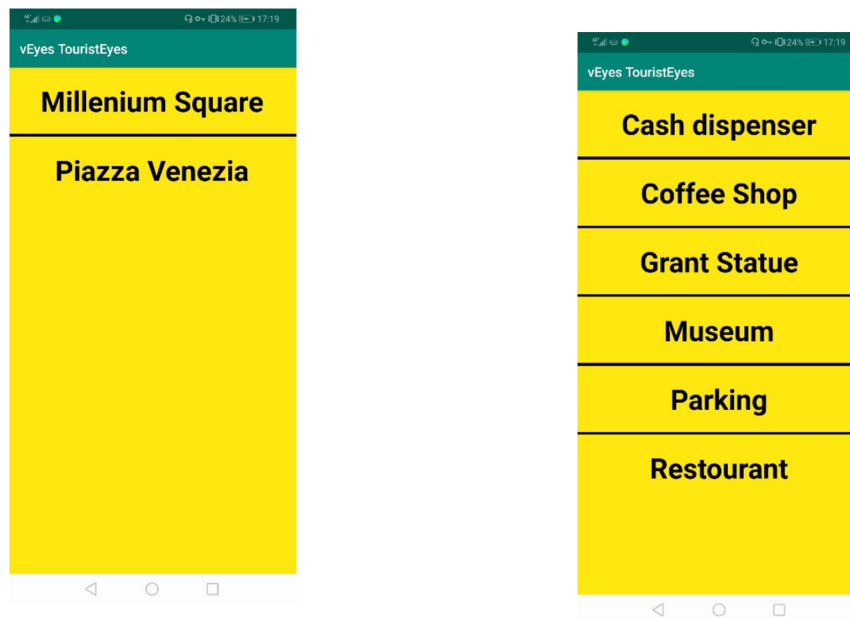


Figure 5.17: Tourist Eyes Manager

in the surrounding environment, a greyscale filter is applied to each frame of the video, grayscaling everything but a squared area near the user. Another filter can be applied to enhance certain colours, so that the system will be able to recognize the hat more easily. When the user crosses one of the guidelines described before, the Manager sends a message to the app. This message is decoded by the app and transduced into an acoustic signal. The user, after receiving the acoustic beep, will change direction to re-enter the route. Once at the destination, another acoustic beep is sent to the user, and the Manager starts listening for new connections.

The Tourist Eyes Training App is used to “train” the user to recognize the actions to be performed for each acoustic signal. The "Sound SX" and "Sound DX" signals tell the user to take a sidestep respectively to the right and to the left. The "Turn to SX" and "Turn to DX" signals tell the user to make respectively a 90° leftwards or rightwards rotation. When the user arrives at destination, he/she will be notified with the "Destination Hit" sound. The “Lost tracking” sound indicates instead that the user is currently not being monitored by any camera. This last signal has been particularly useful to successfully set up both cameras and guidelines.

The Tourist Eyes app is the only software component that each user of the system must have installed on his/her smartphone. Through the app, the user sends a request to the Manager to be tracked and guided to safely arrive at the desired destination. The blind user can easily interact with the application thanks to the support of the smartphone’s voice assistant. The first step is select the "Places" bottom from the main screen of the app to display the list of places where the Tourist Eyes system is active. From the received list (Fig. 5.18a), by selecting, for example, Millenium Square, a list of PoI is displayed (Fig. 5.18b);



(a) List of places covered by Tourist Eyes

(b) Point of interest of a specific place

Figure 5.18: Tourist Eyes app

after selecting the starting point, a second list shows the possible points of arrival. Note that the retrieval of all this information is done via Web Services, which are used by the Tourist Eyes app to receive the lists of available places and relevant points of interest from the SQL Server. After selecting the point of departure and arrival, a request is sent to the Manager. Afterwards, the user will only have to select GO!. The Manager will query the database to retrieve the route that connects the starting point to the destination point, the modules of the route, and the list of cameras associated with the modules. Afterwards, the Manager will send a video request to the first and second camera of the route and starts the tracking algorithm. Finally, the Manager will begin to send messages to the app; these messages will be transduced by the app into acoustic beeps that the user will receive through his/her headphones. When the user arrives at the destination, tracking ends and new connections can be accepted by the Manager. In Fig. 5.19, it is possible to see the data flows of the messages exchanged between the system components.

About the setup-up of the experiment, the first step regarded the upload in the 5GINFIRE portal of the image containing all the software components required to execute the experiment. After that, the VNFD (Fig. 5.20) and the NSD (Fig. 5.21) were uploaded in the OSM portal.

The second phase of the setup of the experiment was carried out at Millennium

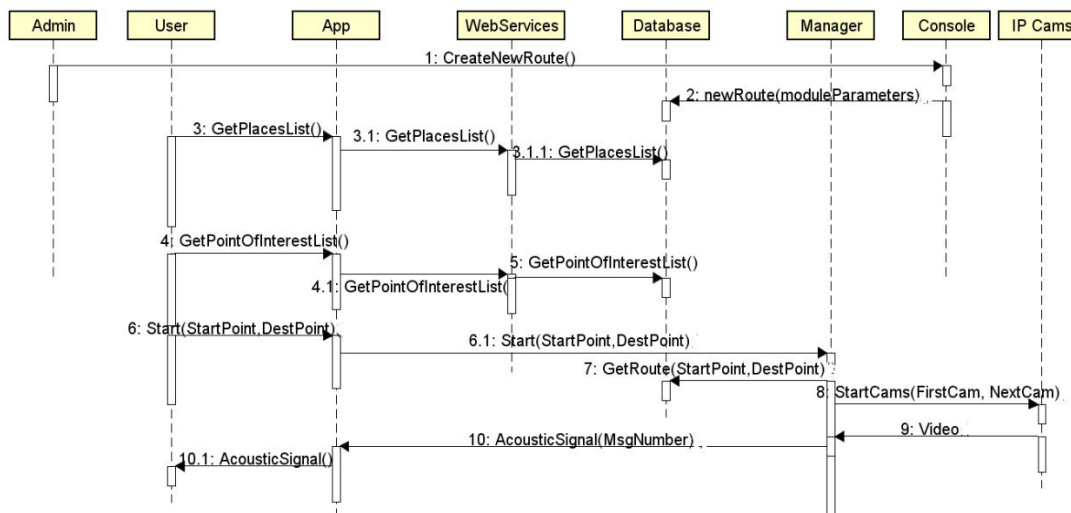


Figure 5.19: Tourist Eyes. Data Flow

IP Cam	Access Point	Distance
1	6	12.3 m
2	6	16.5 m
3	6	19 m
4	5	14.4 m
5	5	14.7 m

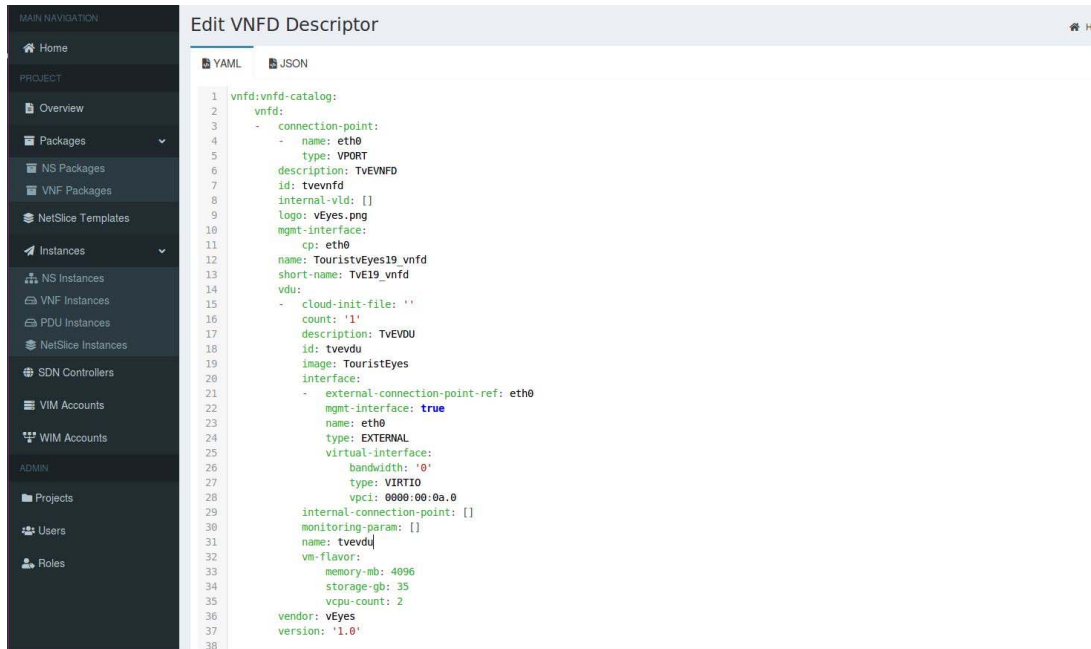
Table 5.5: Tourist Eyes. Distance between IP Cams and Access Point

Square, where IP cameras have been installed. A preliminary check involved the test of the connectivity between the IP cameras and the VNF running in the MEC server. Every IP camera that has been used for the experiment could only support the 2.4 GHz band option. There were TWO access points at Millennium Square to get connected to the Tourist Eyes network. The first AP was fixed on top of “Tower 5” of Millennium Square, while the second one was deployed on top of “Tower 6”.

To better understand the position of the cameras with respect to the access points, see Fig. 5.22. Table 5.5 shows the distance of each IP camera from the access point. The installation and configuration of the cameras was carried out through the Console, that allowed to create the routes from Grant Statue and 640East (Fig. 5.23) and configure the 5 modules of which the route was composed. The creation of the forward route (from the Grant Statue to 640East - Fig. 5.24a) and the return route (from the 640East to the Grant Statue - Fig. 5.24b) concluded the experiment set-up phase.

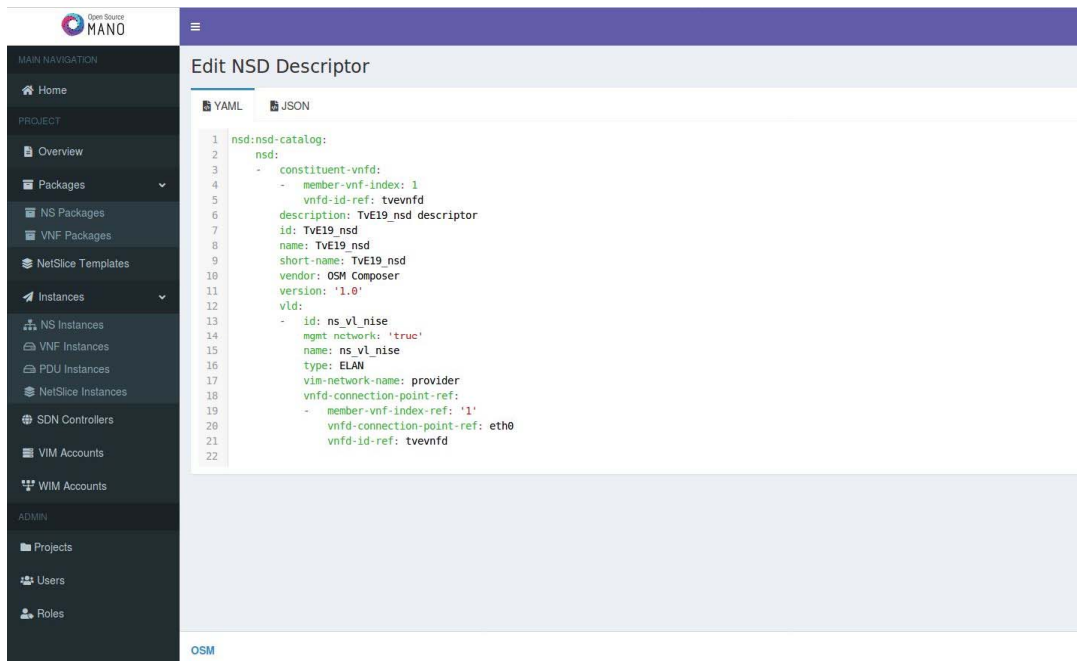
During the experiment phase, three different working regions were identified:

## CHAPTER 5. 5G-ENABLED SMART SERVICES



```
1 vnfd:vnfd-catalog:
2   vnfd:
3     - connection-point:
4       - name: eth0
5         type: VPORT
6         description: TvevWFD
7         id: tvevndf
8         internal-vld: []
9         logo: vEyes.png
10        mgmt-interface:
11          cp: eth0
12          name: TouristvEyes19_vnfd
13          short-name: TVE19_vnfd
14          vdu:
15            - cloud-init-file: ''
16              count: '1'
17              description: TVEVDU
18              id: tvevdu
19              image: TouristEyes
20              interface:
21                - external-connection-point-ref: eth0
22                  mgmt-interface: true
23                  name: eth0
24                  type: EXTERNAL
25                  virtual-interface:
26                    bandwidth: '0'
27                    type: VIRTIO
28                    vpci: 0000:00:0a.0
29                  internal-connection-point: []
30                  monitoring-param: []
31                  name: tvevdj
32          vm-flavor:
33            memory-mb: 4096
34            storage-gb: 35
35            vcpu-count: 2
36          vendor: vEyes
37          version: '1.0'
```

Figure 5.20: Tourist Eyes. VNFD in OSM



```
1 nsd:nsd-catalog:
2   nsd:
3     - constituent-vnfd:
4       - member-vnfd-index: 1
5         vnfd-id-ref: tvevndf
6         description: TVE19_nsd descriptor
7         id: TVE19_nsd
8         name: TVE19_nsd
9         short-name: TVE19_nsd
10        vendor: OSM Composer
11        version: '1.0'
12        vld:
13          - id: ns_vl_nise
14            mgmt network: 'true'
15            name: ns_vl_nise
16            type: ELAN
17            vim-network-name: provider
18            vnfd-connection-point-ref:
19              - member-vnfd-index-ref: '1'
20                vnfd-connection-point-ref: eth0
21                vnfd-id-ref: tvevndf
```

Figure 5.21: Tourist Eyes. NSD in OSM

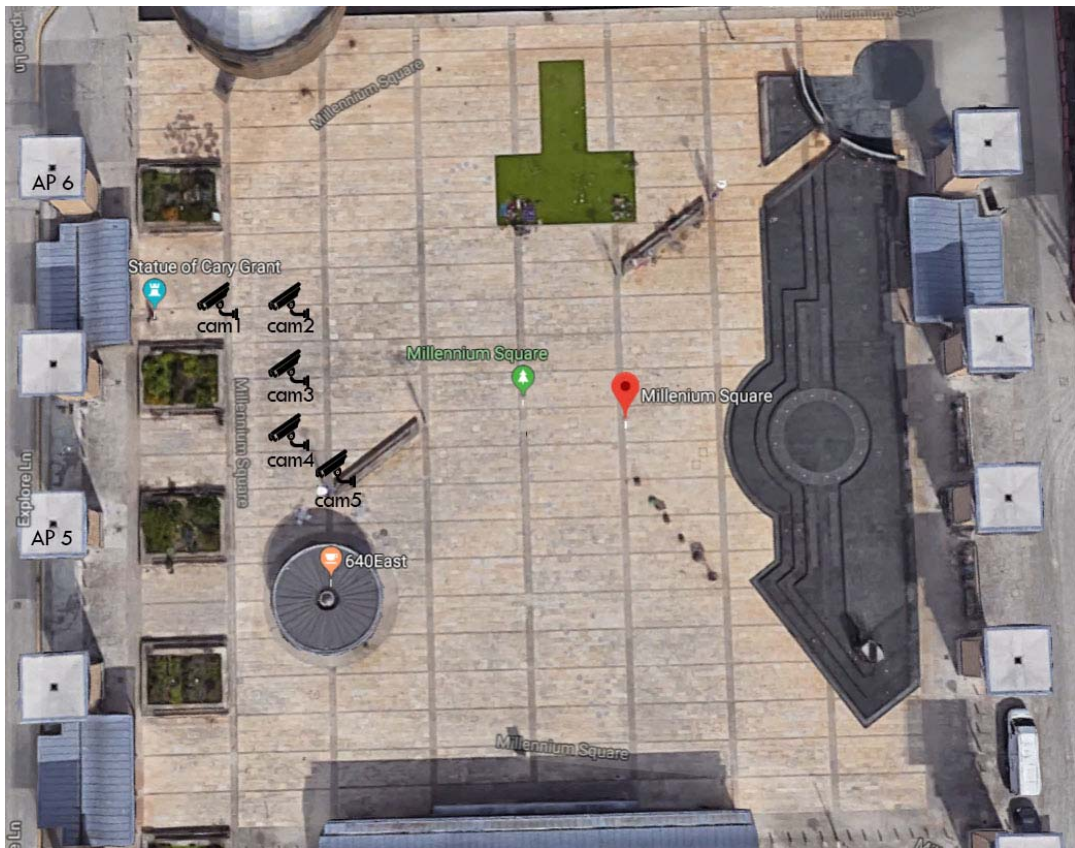


Figure 5.22: Tourist Eyes. Position of the installed camera in Millennium Square

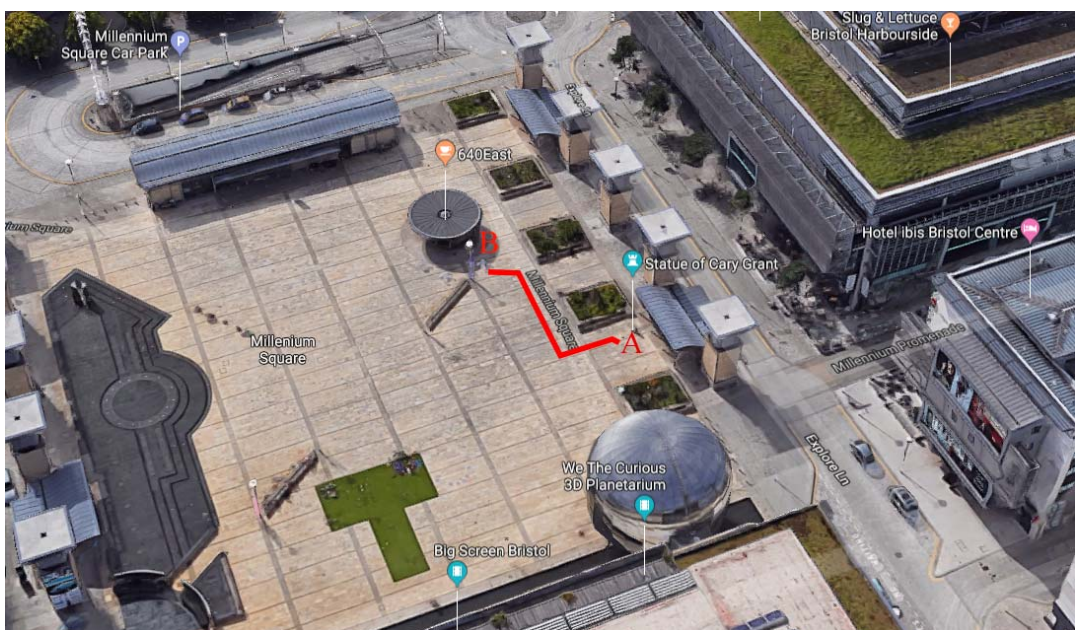


Figure 5.23: Tourist Eyes. Path during experiment execution

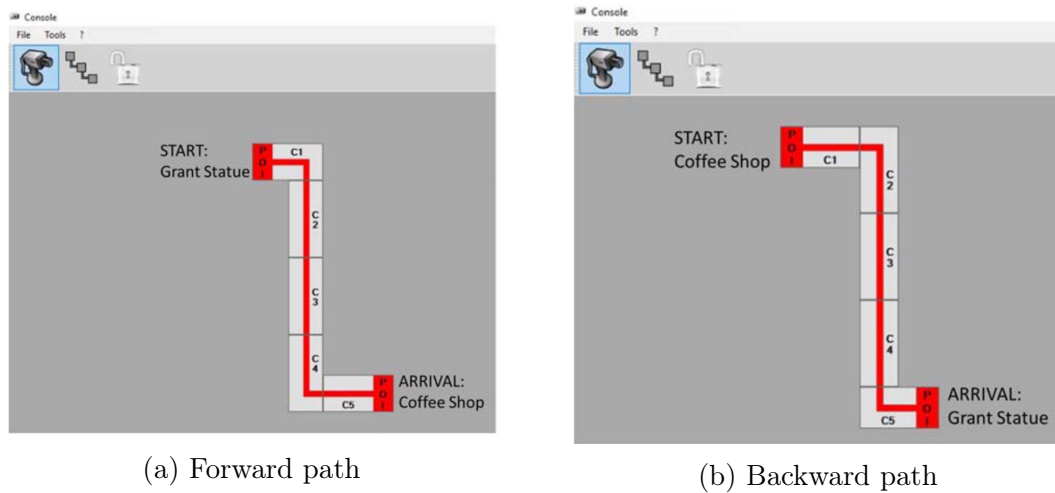


Figure 5.24: Tourist Eyes. Forward and backward patch during the experiment

- Time Interval P1 (Tuesday, October 22nd, from 10:45 to 11.55 am): the system experienced high quality performance, since the video was flowing smoothly through the system;
- Time Interval P2 (Wednesday, October 23rd, from 12:45 to 1.45 pm): the system performed poorly in terms of quality of service, with recurring image freezing events and consequent system non-responsiveness, as detected by the user. Even though the blind user was diverting from his path, the system gave no feedback to correct the trajectory, or gave feedback with a very long (unacceptable) delay;
- Time Interval P3 (Wednesday, October 23rd, from 4:45 to 5.45 pm): the system worked with rather good quality. All the experiments during this time interval were successful, the user was able to receive all the expected feedback and complete his/her path. In fact, when his/her diversions from the established path resulted in deviations from the planned trajectory, the feedback worked properly, although with short but still acceptable delay.

Note that the distinction of these working regions comes from problems which were external to the Tourist Eyes experiment and mainly related to traffic and users accessing the network, considering that the Millennium Square area in Bristol was very crowded at certain day time and many users at the time of the experiment execution were using their own applications and accessing the Internet. Also the weather conditions were very variable and unstable, and thus our experiment suffered of the bad propagation conditions.

For each of the above time interval, some numerical results in terms of CPU

and memory utilization in the VM, and delay and message loss percentage for the network layer, will be shown.

As regards the CPU and the memory, the measurements in the three scenarios considered showed that both of these parameters are absolutely not influenced by the quality of the radio access link. Moreover, varying the bitrate used to encode the video flows, during the analysis phase of them both CPU and memory does not result overloaded in any case.

About P1 scenario, referring to Fig. 5.25 it is possible to see that at the beginning (a) the user is halfway through the path section visible to CAM3. The user keeps on walking, leaves the section visible through CAM3 (b) and enters the section of the path under coverage of CAM4; finally, the user is captured by CAM4 (c). By observing the circle in the black rectangle representing the user as tracked by the system, it is possible to note that the system perfectly works and allows promptly making handover between different cameras.

As shown in Fig. 5.26, upon increasing the encoding bit rate, a higher bit rate is measured. In particular, when the encoding rate is not more than 512 kbit/s, the bit rate is very close to the encoding target. Instead, when the target is too high, i.e. 2 Mbit/s, the maximum bitrate is generated, which is not higher than 87% of the target encoding rate.

Moreover, two highly significant parameters like the average delay and the loss percentage are shown in Fig. 5.27. In particular, note that in Fig. 5.27a the average delay of CAM3 and CAM4 are significantly higher than the average delay of all the other cams. This specific behavior was also detected for CAM3 in the loss percentage graph (Fig. 5.27b). Thanks to the help of the Millennium Square Network Staff, was discovered that this behavior was due to a poor quality of the wireless link between these cameras and the WiFi access points.

Regarding P2 scenario, in Fig. 5.28 is shown the monitoring of user's movement. In particular, at the beginning (a), the user is halfway through the path section visible to CAM3. The user keeps on walking but suddenly the video freezes (b). Therefore, the system perceives the user as if he/she has stopped. At the same time the video from CAM4 flows regularly (c). Since the algorithm is applied to CAM3 (frozen), the user receives a false feedback, because he/she is still moving as shown in CAM4 (d) but the system detects it inside CAM3 coverage area. The reason for the algorithm still relaying to CAM3 is that the user has not entirely left CAM3 but CAM3 is frozen; therefore, the system has not realized that the user left CAM3 and entered the region served by CAM4.

The average transmission bitrate, the average delay and the loss percentage

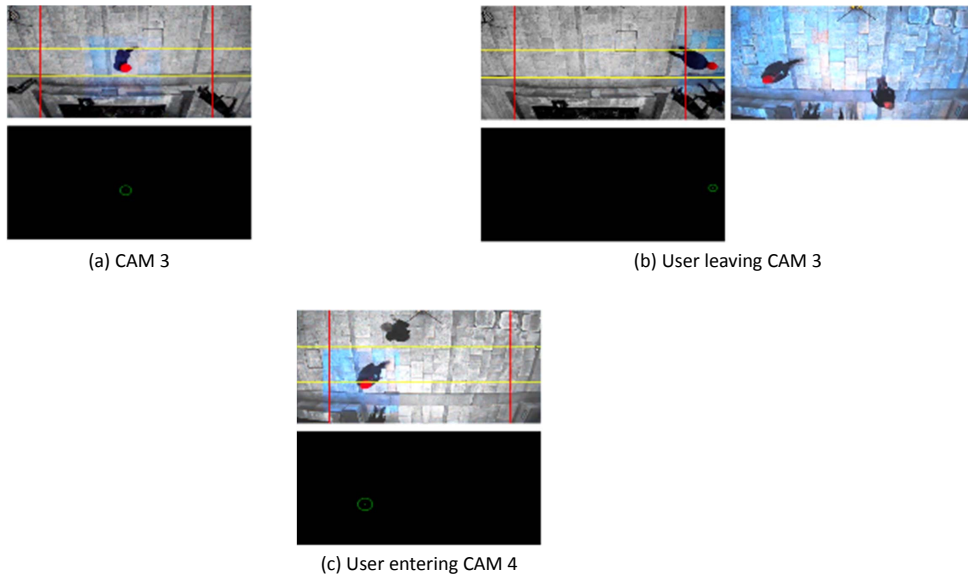


Figure 5.25: Tourist Eyes Console Screen - High Quality Scenario

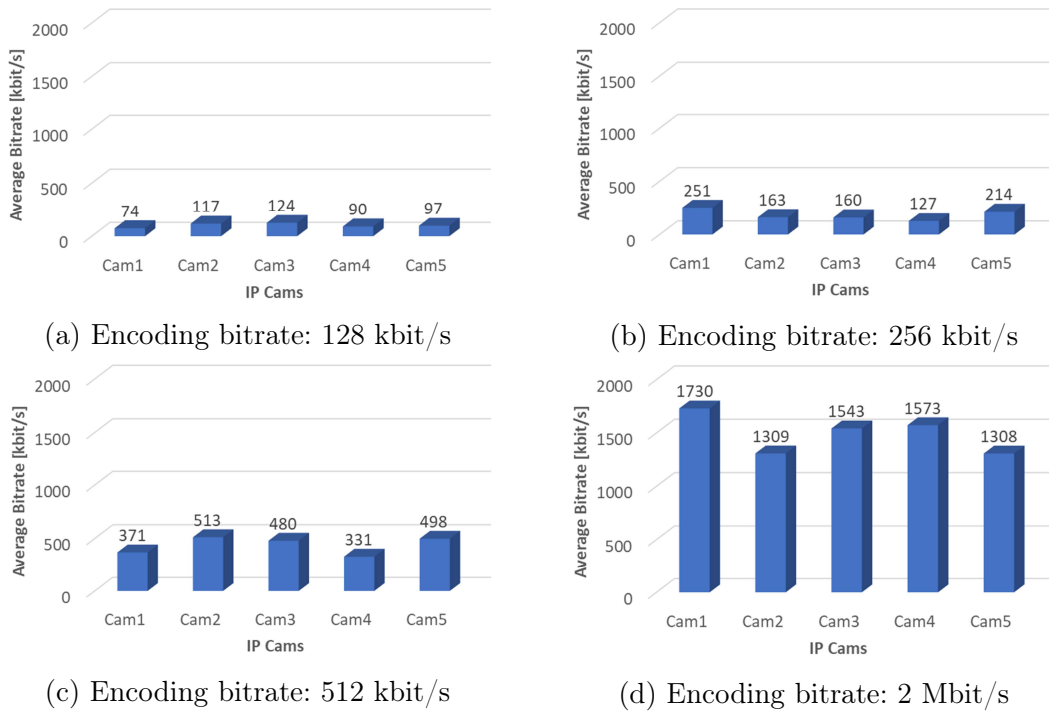


Figure 5.26: Tourist Eyes. Average transmission bitrate from IP Cams in P1

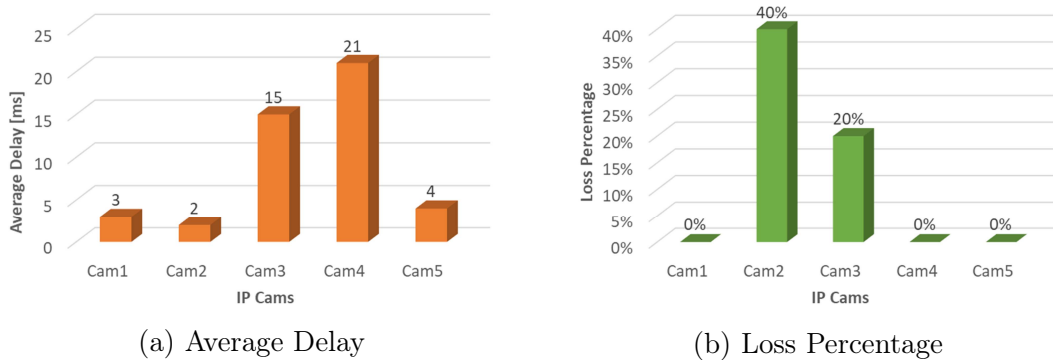


Figure 5.27: Tourist Eyes. Delay and Loss Percentage in P1



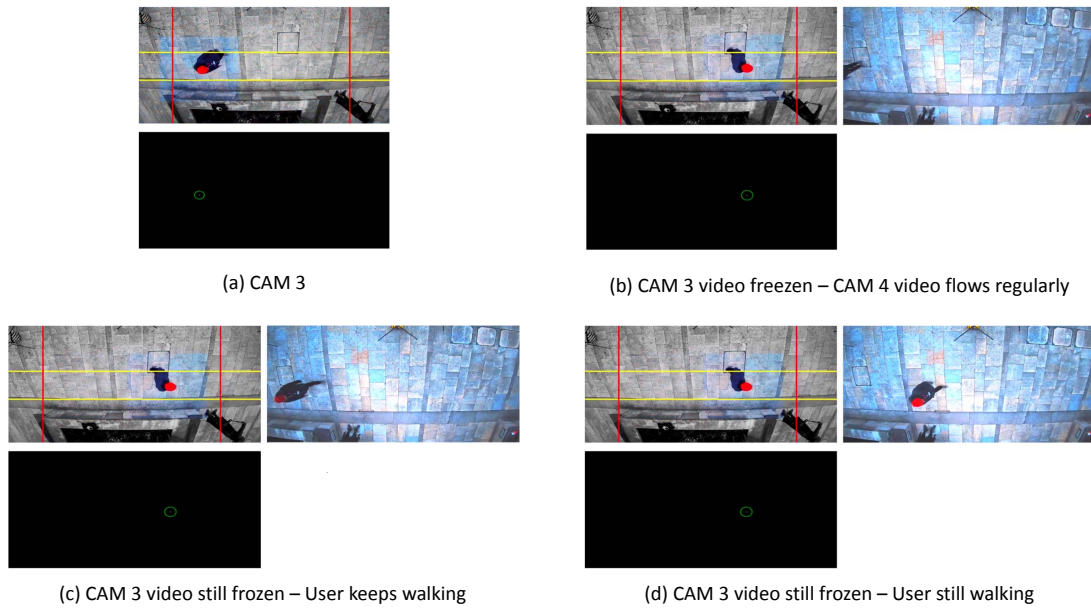


Figure 5.28: Tourist Eyes Console Screen - Poor Quality Scenario

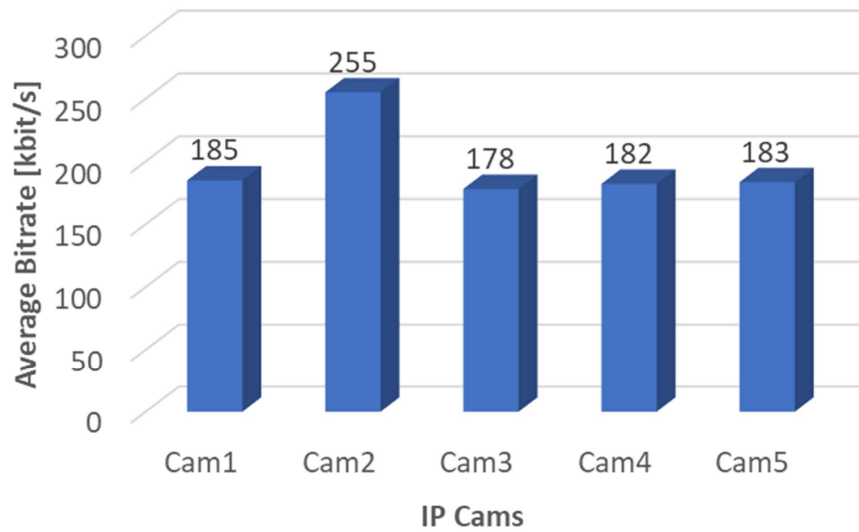


Figure 5.29: Tourist Eyes. Average transmission bitrate in P2

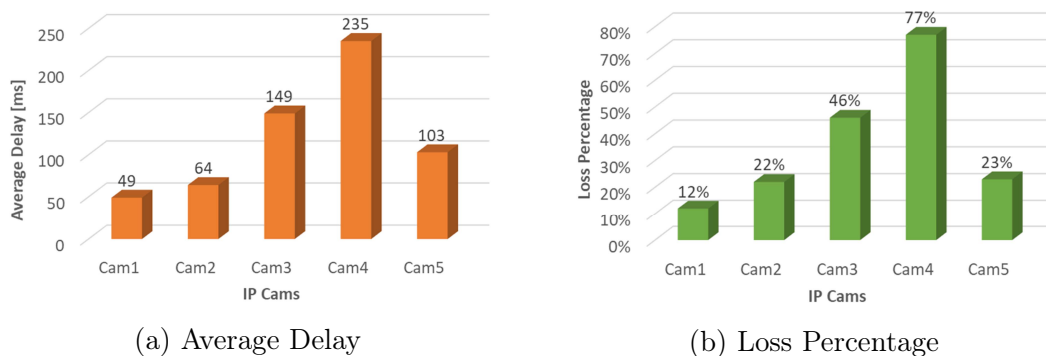


Figure 5.30: Tourist Eyes. Delay and Loss Percentage in P2

are reported in Fig. 5.29, Fig. 5.30a and Fig. 5.30b. Each IP camera presents a different average transmission rate, that depends on the current scene it is focusing, the movements of the user in the covered area, and the presence of other persons in the same area. As regards average delay and loss percentage, it is observed a similar behavior of the previous scenario. In fact, although with higher values, CAM3 and CAM4 present the worst values, again due to the badness of the wireless link used for connection to the structured network. The too high value of both delay and loss percentage, especially for CAM3 and CAM4, motivate why the experiment is not working in this case.

Finally, about the P3 scenario Fig. 5.31 shows that at the beginning (a), the user is halfway through the path section visible to CAM3. The user keeps on walking; the user (b) leaves the section visible by CAM3 and enters the section of the path under coverage of CAM4. The network delay affects the commutation process from CAM3 to CAM4; as evident in (c) unfortunately some delay is experienced from when the user entered the area served by CAM4 and when his new position was identified.

The average bitrate generated by each CAM is presented in Fig. 5.32. Here, again, it is observed that each camera has a different behavior, and CAM4 is the one more stressing the network. The average delay and the loss percentage are shown in Fig. 5.33, respectively. In Fig. 5.33a the average delays of CAM3 and CAM4 are significantly higher than the average delay of CAMs 1, 2 and 5. This specific behavior of CAM3 and CAM4 compared to the behavior of CAM1, CAM2 and CAM5 was also detected in the loss percentage graph shown in Fig. 5.33b.

The results carried out during the experimentation show that there are a number of critical and unexpected issues which need to be taken into account and that are related to traffic conditions, delay, weather conditions, possible temporal malfunctioning of certain cameras which strongly impact on the performance of the system. However, the Tourist Eyes system is very robust and reliable and showed to well absorb these unexpected features.

### 5.3 5Gamer: a 5G-assisted online game

Online gaming represents a very large portion of today's videogame scene, as most players like to compete against other players online. Online games are gaining more and more popularity thanks to many online games being released for free and made available to everybody (e.g. Fortnite, Apex Legends, League of Legends,

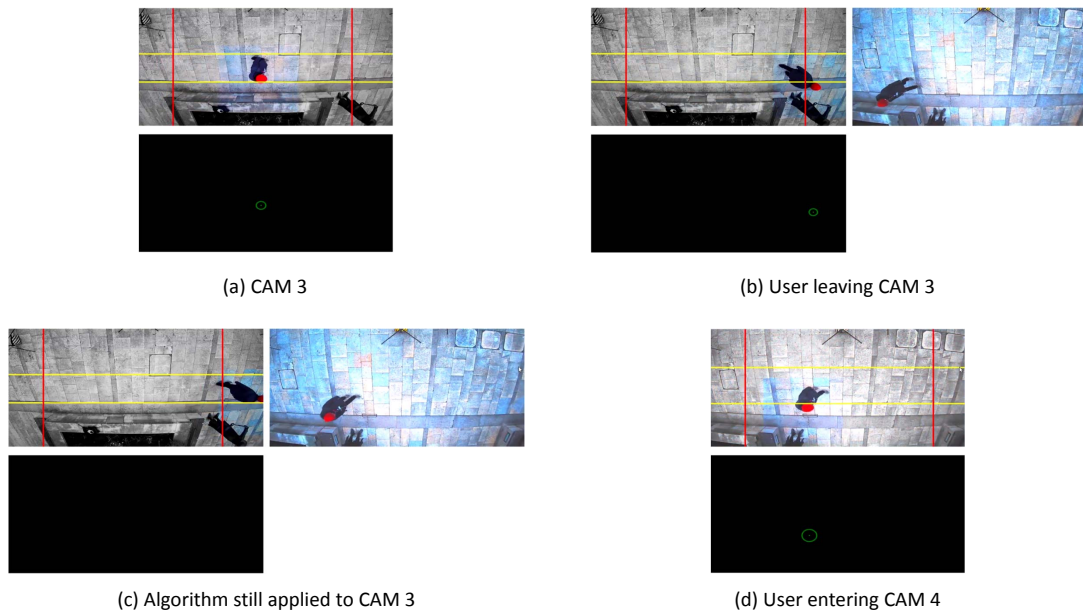


Figure 5.31: Tourist Eyes Console Screen - Good Quality Scenario

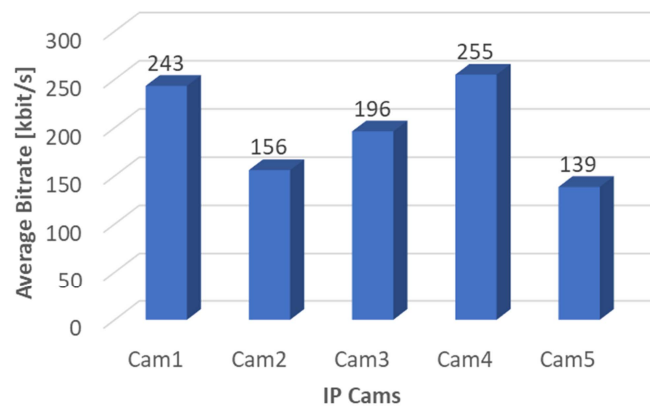
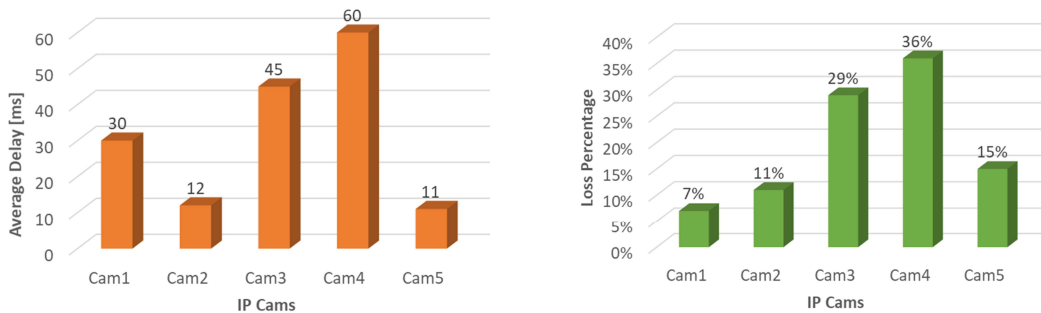


Figure 5.32: Tourist Eyes. Average transmission bitrate in P3



(a) Average Delay

(b) Loss Percentage

Figure 5.33: Tourist Eyes. Delay and Loss Percentage in P3

etc.). Furthermore, competitive games have been recently officially recognized as sports (E-Sports), with national and international tournaments taking place all over the world, making online games of great importance even in fields outside the gaming industry.

One of the most critical issues in realizing new generations of online games is that these games are affected with all sorts of network problems: the most common is lag, the delay between the performing of an action by the player and the actual happening of the action online caused by the command arriving late to the hosting server. Other problems are represented by accidental connection interruptions, delays introduced by NATs, excessive queueing waiting in network device, etc. Most games, in case of an accidental disconnection by a player, simply remove that player from the match (in some cases, the entire game is interrupted), or use a pre-scripted bot to control the former character left uncontrolled by the player. Scripted AI hardly have skills comparable to that of a human player (they usually are way weaker or sometimes way stronger), and in team games, having an ally player being substituted by a bot is always seen as a huge handicap for the whole team. Furthermore, the AI is often the same for all players, meaning that a bot can raise or drop the overall skill level of a team depending of the human player's original skill level. An additional problem is that game servers are run on remote clouds, and this is braking the diffusion of a new generation of online games with very high interactivity among two or more players that, on the other hand, require ultra-low latency guarantees.

5G network will constitute an enabler technology for the bootstrap of such kind of online games. Videogames providers can run servers at the edge, so assuring ultra-low latency in the interaction between players playing together the same game. A fundamental role will be played by AI tools placed at the edge, which are able to substitute remote control when the network is not able to guarantee the required quality, in such a way that these malfunctions will be hidden to the final users.

To demonstrate the gain received by online games thanks to the 5G technologies support, *5Gamer* was deployed inside 5GINFIRE project. The experiment is a Pong reproduction (see Fig. 5.34) with the addition of network and ML functionalities. The idea is to have a 1 vs 1 online game. The application object of this experiment can be easily extended, as a future work, to a multi-player scenario with high transmission bitrate to sustain high-quality video transmission, even including virtual-reality interaction among the players.

At the edge of the network, close to each player, a virtual machine hosting an

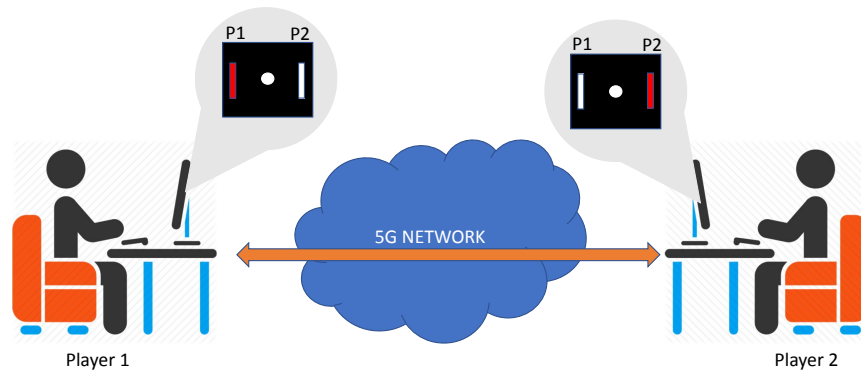


Figure 5.34: 5Gamer. Application scenario

entity called Digital Twin (DT) is in running state in order to add stability to the connection among players, reducing the impact of random network problems, which can make the difference in an online match, and improving the overall QoS of the game as a result. Each DT observes the game, trains a Neural Network (NN) model by imitating the opponent’s movements and actions, and applies techniques of ML. If the player app senses a slow or faulty connection from his opponent, the near DT enters the game as opponent, in order to hidden this network malfunctioning to the human player. When the connection regains stability, the DT leaves the game, and the remote human player is re-enabled. The human opponent has no awareness of being substituted. This is feasible only thanks to the computing facilities that the 5G network, in this case given by the Bristol testbed, are able to provide at the edge of the network.

In particular, the workflow of this experiment is presented in Fig. 5.35: it needs four machines, two for the real players and two for the DTs. All these machines (both physical and virtual) need to have installed some components: the Unity Engine, Anaconda 3.4 (including the setup of a virtual environment with all required Python libraries installed, such as TensorFlow) and the Unity ML-Agents Toolkit.

Two remote users are connected through a 5G network and play to the same game: one player hosts the game (as if it were a server) and the other one connects to him as a client. Shortly after the beginning of the game, the DTs connect to the ongoing game. The DT1, running on the network edge close to the player 2, aims at learning the behavior of the player 1. DT2 works in a symmetric way. To this purpose, each DT runs an Imitation Learning algorithm, realized using the experimental Unity’s ML-Agents Framework, in order to train a Deep NN model (made with TensorFlow) to behave like the opponent of the player which is close to. The trained model is constantly updated and sent to the close player, which

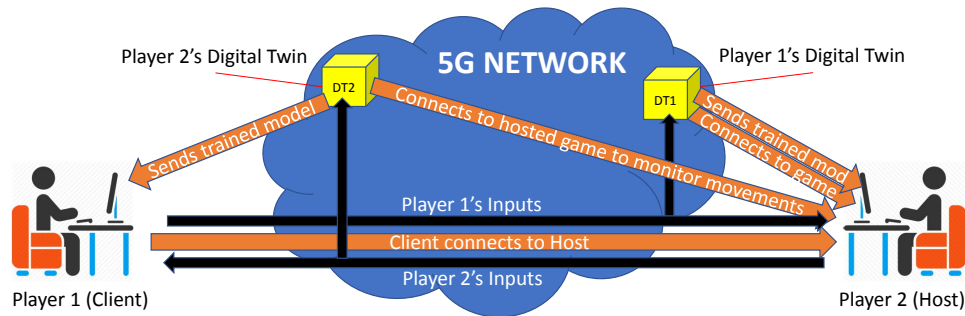


Figure 5.35: 5Gamer. System Architecture

uses it to generate a bot in case of network malfunctions from his opponent. As the training continues, the model will be more and more well trained in imitating the original player’s actions and, in the ideal scenario of a very long training, the bot becomes impossible to be distinguished from the human.

The Imitation Learning is realized by two game “Brain” objects: Follower and Trainer. The Follower has the objective to recognize the player to imitate and start following it, copying its inputs. The Follower acts as a “Teacher” to the Trainer (which represents the “Student” Brain), which uses the Follower’s control signals as inputs during the training process. During the experiment, the training progress can be seen on the command prompt, specifying the current training step, the reward value (as this is a special kind of Reinforcement Learning Algorithm) and when the `.nn` file (containing the trained model) is saved and updated.

Direct communication between DTs and Players is used to update the file of the models as they are generated and updated. Every time a new `.nn` file is generated, is then sent to the player. Each player game project comes with a bot prefab, which is instantiated every time the quality of connection drops below a set threshold. The bot uses the `.nn` file as a Brain in order to make decisions in game.

To test the application, some network malfunctions can be introduced, like traffic, lag, and other problems. At this point, communication from Player 1 is completely blocked, Player 2 senses a disconnection and enables the bot, to guarantee the quality of the game. The bot will use data provided by the DT1 at the edge of the network corresponding to the results of the imitation learning algorithm performed on the movement of Player 1. The bot is spawned immediately and in the same exact position of the Player right before disconnecting, making the whole process completely transparent for both Players, as Player 1 will keep playing as normal (because connection from Player 2 is still functioning properly) and Player 2 will play with the bot. The whole scenario returns seamlessly to the

normal behaviour as the communication problems from Player 1 are solved.

One of the most important peculiarities of this experiment is that, taken away the modelling of the smart agent, this system can be potentially applicable to any kind of online game. In the case of the current experiment, there is only one degree of freedom, as the Pong paddle can only move in one direction. In most complex games, more degrees of freedom, control signals and observations need to be considered, and this can be easily done with Unity's ML-Agents Toolkit. The challenge of the 5Gamer experiment is to demonstrate how 5G technologies are enablers of everyday life services, before unthinkable due to the severe limitations of the current technologies. 5G provides all active nodes with:

- ultra-low latency, here needed for real-time interaction between each player and the instance of his personal DT virtualized at the edge of the network;
- huge transmission rates, here needed by the DT to both retrieve large amount of data critical for training and to continuously send the model to the Players
- a huge number of connected devices, because even if not in this case, in a more realistic scenario of an online multiplayer game, there can be thousands of Players simultaneously connected, and it has to be considered that, in this scenario, the number of actual connected devices is more than doubled, because each Player has at least one DT.

As it is possible to observe from the comparison between the state-of-the-art system depicted in Fig. 5.36a, and the framework of the proposed experiment, shown in Fig. 5.36b, the gains achieved with the proposed experiment can be synthesized as follows:

- the simple fact that the game is running on a 5G network grants the overall QoS of the game a sharp raise, with much less lag, less disconnections, and a far higher potential number of interconnected players at the same time;
- 5G technologies enable heavy computing in the network as well, here represented by several training algorithms running at the same time. It goes without saying that the use of an AI modelled on the playstyle of a certain player that overtakes control of the situation if that specific player disconnects is a gigantic step forward in comparison of standard, pre-scripted bots;

- another enhancement, even if it may seem unimportant at a first glance, is represented by the overall improvement of the players' mood and degree of satisfaction while playing. Although the proposed experiment aims to sharply decrease, if not completely eliminate, the negative impact of accidental disconnections during an online game, which can be caused by malfunctions in the network, in the hardware or in the electrical system, some players use to disconnect from the game on purpose (usually, when the game is going bad for them), resulting in a handicap for the entire team, that snowballs to the worsening in the players' mood and liking of the game, to a drop of the game's popularity as a result. The proposed system eliminates the negative effect of these disconnections as well, making the so-called "rage-quitters" completely unoffensive to other players, which won't notice the difference before and after the rage-quitting. Hopefully, this can cause other beneficial effects, like discouraging negative behaviours such as these, making the online community less "toxic" and capable of attracting new players, with great advantage of both players and software houses. It goes without saying that, despite the proposed prototype eliminates its effects, rage-quitting is still a bad habit that has to be detected and punished nonetheless, as most online games already do.

Results coming from this experiment had been exploited in several ways. It has to be noticed that the nodes of this prototype can be strategically moved and modified to adapt to different situations. In fact, the proposed service has enormous potentialities when applied to much more complex architectures and more realistic scenarios: for example, consider a classic situation where ten players start a 5 vs 5 team match, all of them are connected to a central server, that hosts the game. The DT of each player can be placed physically near to the player itself, at the edge of the network, learning his movements while playing, and finally storing the trained model in the server. Therefore, the game's servers can collect information on the playstyle of each player, update it at every match, and use it in case of bad connection to control their entities locally (as the server is the host of the match, the bot control is performed with zero delay). Another realistic scenario to be considered is that of a Battle Royale game, where there are a hundred or more players connected, or a MMORPG (Mass Multiplayer Online Role-Play Game), where the number of simultaneously connected players normally exceeds the order of thousands (for example, a World of Warcraft server can hold up to 7000-7500 players at the same time) and where the use of 5G technologies can really make an enormous difference. Not only it makes the implementation of Imitating AIs



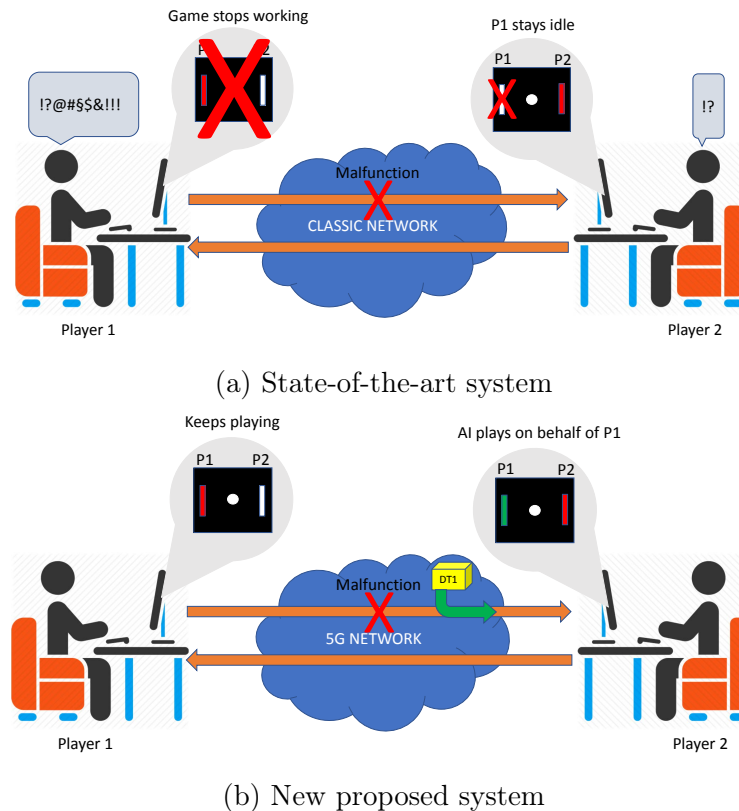


Figure 5.36: 5Gamer. Comparison with the state-of-the-art in case of malfunction

possible, but it opens a lot of possibilities for these kinds of games, that are used to a “quantity over quality” approach: for example, games like Fortnite or Final Fantasy XIV still exhibit poor graphics compared to the standards, because the computational power used to process graphics is kept to a minimum in order to maintain the large amount of users reliably connected.

## 5.4 VISION: a platform for smart-city video surveillance services

The *VISION* (VIdeo Surveillance for Impaired persons) platform was realized in collaboration with HTLab, an innovative startup originated by the previous described vEyes. The mission of HTLab is targeted to all kinds of impairments and any application of technology to healthcare.

The objective of VISION is to develop a video surveillance framework for smart cities, aimed at providing a social and shared video surveillance tool to help impaired people (blind people, people with limited-mobility, deaf people, old people) which could need to be guided/monitored safely within the city. VISION deals

with the application of 5G network softwarization for verticals with differentiated QoS requirements. It was presented to the 3rd Flame Project Open Call and it is currently implemented in the replica site of Bristol.

In Subsection 5.4.1, the architecture of the 5G testbed, realized by Flame Consortium, is described. Next, Subsection 5.4.2 presents the implementation of VISION and all its components.

### 5.4.1 Flame Project

The Flame project is a Research and Innovation action/project under the EU programme Horizon 2020 [86]. The objective of the FLAME platform is to provide a system characterized by low latency distributed computing as well as content over a 5G-enabled programmable infrastructure, providing the user with faster access to media and services. Through the platform's fast and dynamic service request routing capability, media service providers will have fine-grained control over load and therefore costs across the network. This offers the potential to significantly reduce the overall costs while ensuring fast availability of services towards end users. Another important capability of FLAME platform is the possibility of a cheaply broadcast content to multiple users without any need for adopting clients and services to specific multicast protocols, significantly reducing the predicted cost increase for video delivery.

In Fig. 5.37, it is shown the FLAME platform architecture [87], operating on top of an infrastructure exposing an ETSI MANO compliant interface, that allows FLAME platform to reserve compute, storage and networking resources. The definition of experiment API (compatible with HTTP/IP standard) through the implementation of management and monitoring interfaces, allows the placement of compute, storage and network resources during the experiment. In fact the Flame platform orchestrates the deployment of media components considering them like internal service function because the same resources provided by the infrastructure layer, are in turn provided by Flame as retail resources to the media services at the top of the platform through management interfaces. There is also a monitoring interface that allows information to be exchanged between the media services and the platform. Based on this information, it is possible to configure some alerts used to trigger actions inside the service function. This aspect is important, because it allows the experimenters to discover particular relations between the service under test and the resource specifications. All that said, the unique requirement of the media service deployed on FLAME platform is that it is composed by a set of media components, communicating each other

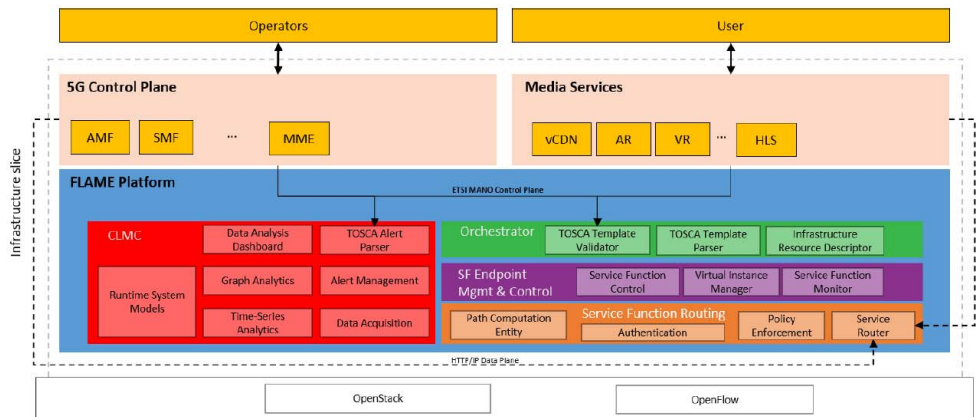


Figure 5.37: Flame Platform Architecture [87]

through an HTTP/IP compliant data plane interface. As shown in Fig. 5.37, the FLAME platform is able to support the realization of 5G control plane services, such as the Session Management Function (SMF), for vertical control planes as envisioned for, e.g., vehicular or industrial scenarios.

The different components of the FLAME platform are: *Orchestrator*, that interfaces with the infrastructure resource management through the aforementioned ETSI MANO compliant interface. Its objective is managing the compute/storage/network resources towards the media service provider, while it utilizes the surrogate policy control interface towards the *Service Function Endpoint Management and Control* (SFEMC) component to realize the orchestration-level management policies as well as to set suitable shorter-term control policies for service function endpoints.

The *Cross-Layer Management and Control* (CLMC) component gathers information across various layers. Data are collected at various levels [88]: metrics from the SDN level describing routes and latencies; metrics from the container of each deployed service function (service function endpoint or SFE) describing the memory, disc, CPU and network; metrics describing the performance of application containers or web servers such as Tomcat or Nginx if they are used; and application-specific metrics where they are implemented. These data are useful and needed for control-level decisions, such as the activation of service endpoints, but they also provides a rich pool of data for media service providers. The CLMC brings together timeseries and graph analytics to understand demand, resourcing and performance properties of media service function chains deployed within the FLAME platform

For the realization of the configured service function endpoint policies, the SFEMC layer utilizes the FQDN registration interface to control the registra-

tion and deregistration of the service endpoints towards the *Service Function Routing* component. The service routing layer will use the OpenFlow interface to suitably configure the switching fabric of the underlying infrastructure. More details about the interfaces of each component and the connections between these ones are available in [87].

The file descriptors used to specify the deployment of media services are based on TOSCA YAML and TOSCA NFV descriptors [89]. The idea is to take advantage of these standard specifications and definitions provided by them. TOSCA offers the possibility to define virtual instance templates and their nodes to be deployed using the Orchestrator, which is in charge of to collect the requirements of the templates, interpret these requirements and to leave ready these requirements. Characteristics of these machines can be defined in the TOSCA template (e.g., nodes, properties, hardware and software requirements, policies, machine status, etc.).

To run the experiment, two TOSCA files are needed. One includes a description of the machines that compose the NS, in terms of name and computational resources (memory and disk), the hypervisor used to create the machine (kvm or lxc), the URL where is located the image that has to be used to run the machine, and the Fully Qualified Domain Names (FQDN). This last one is needed to assign one or more unique DNS alias to the node. In this same file a section about the policies is present. More policies can be defined, but the *init* policy is the only one that has to be always defined, because specifies the initial state of the node. In particular a node can be in one of these states (Fig. 5.38):

- *Not\_Placed*: the node is not deployed in any cluster. The image for a subsequent launch is not loaded and the Orchestrator does not reserve the necessary resources. A node placed in this state is unusable in the start-up state of the NS;
- *Placed*: the node is instantiated, but it is off and without connection. In this case the Orchestrator reserves in the cluster the resources needed to power-on the node in a second moment;
- *Booted*: the node is on, but without an assigned IP address;
- *Connected*: the node is on and connected.

There are other policies regarding the change of the node states during the experiment whose activation depends on some triggers. These triggers are defined in the second TOSCA file. This file is sent to the CLMC that manages these events

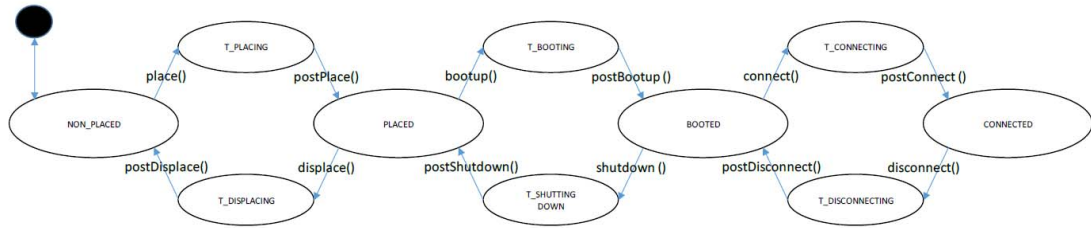


Figure 5.38: FLAME: Possible node status and transitions from one status to another

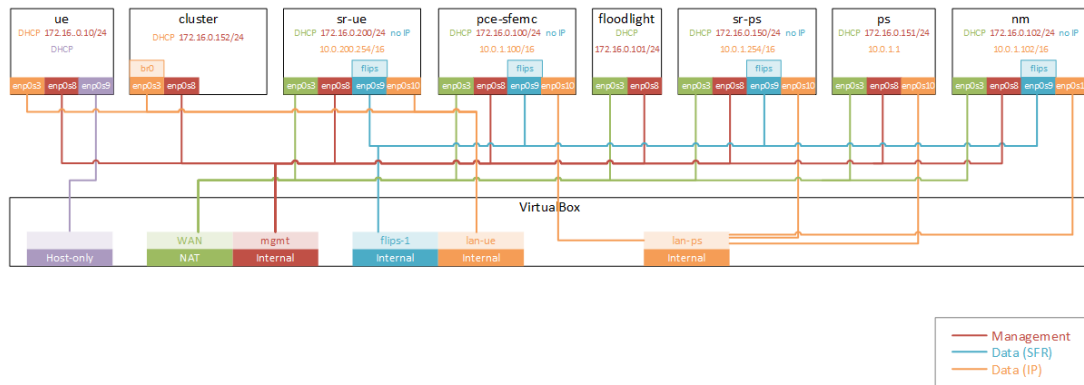
and the related alert messages. When certain conditions occur, the state of one or more machines within the cluster could be change.

FLAME provides a pipeline of increasing complexity and realism to support experiments and trials. In particular, before to deploy the experiment in the Replica site (Bristol for VISION project), is needed to try it on Flame-in-a-Box (FiaB) and Sandpit [88].

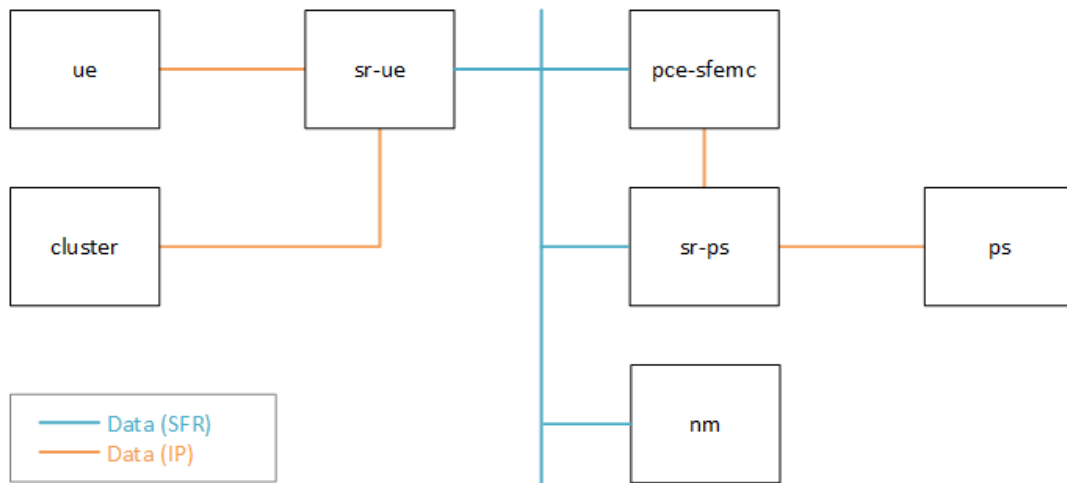
FiaB (Fig. 5.39) is a VirtualBox-base mini Flame platform, with the aim of allowing a first interaction between the media service and the complete platform deployed in the Replica. In particular the experimenter can test SFC orchestration templates (the first TOSCA file needed to deploy the service), SF provisioning and basic communications between the service and the other elements of the platform.

As show in Fig. 5.39a, FiaB is composed by eight VMs:

- *UE*: an emulated client used by the experimenters to test the application. It represents a bridge between FiaB and external systems;
- *Cluster*: the cluster in which the SFC is deployed;
- *SR-UE*: a service router for UE and Cluster;
- *SR-PS*: a service router that works like GW for all the other IP endpoint in the platform;
- *PCE-SFEMC*: the path computation and service function management and control instance;
- *NM*: The SR manager allowing you to configure TLS certificates enable service function routing for HTTPS;
- *Floodlight*: the SDN controller;
- *PS*: the instance that hosts DHCP server. DNS, IP GW and SF repository.



(a) Physical topology of the VMs inside FiaB



(b) Logical topology of the VMs inside FiaB

Figure 5.39: Flame-in-a-Box architecture [88]

The CLMC component is not present in FiaB implementation: for this reason only the first TOSCA file can be tested in this first step of the experiment. Furthermore, the presence of only one cluster reduces the possibility to deploy the experiment in more than one cluster.

The next step is the deployment of the experiment on Sandpit. IT Innovation provides a server ([givry.it-innovation.soton.ac.uk](http://givry.it-innovation.soton.ac.uk)) which has the FLAME platform deployed in a combination of containers and virtual machines to emulate a replica with multiple small data centres such as Bristol. The principal difference with FiaB is the presence of CLMC component and the possibility to use more resources, due to the four "cluster" that can be used to deploy and test the SF.

Thanks to the CLMC and SFEMC, in the Sandpit step is possible to test also the second TOSCA file with triggers and alerts: this allows the experimenters to test the SF in a complete version, but it is not possible to test its performance.

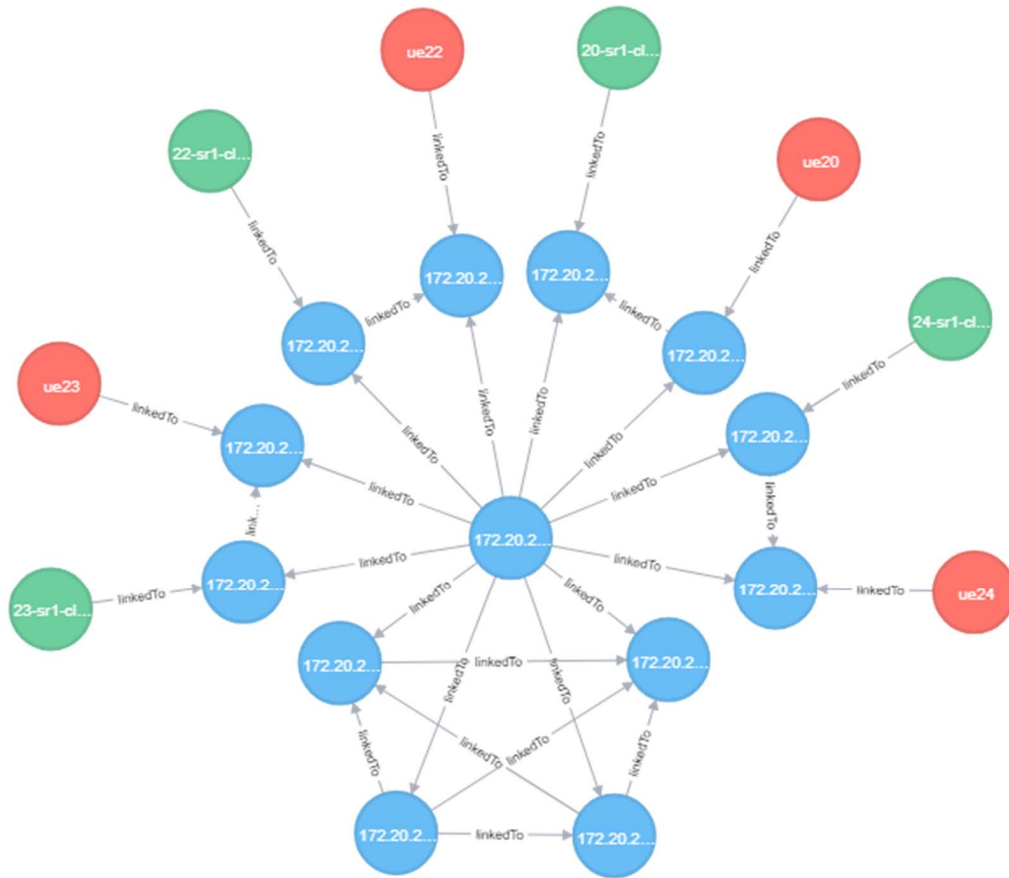


Figure 5.40: Topology of Sandpit with clusters (green), emulated UE (red) and SDN switch (blue) [88]

Finally, the experiment has to be test in the Replica site [88]. The performance (network and hardware resources) in the Replica will be different to the sandpit. The connectivity of mobile phones must be also tested (in terms of signal strength, bandwidth and which access point is used in different locations). All the data collected during the experiment, as in Sandpit, can be explored through the CLMC interface. As said before, the chosen Relica site for the experiment was Bristol, where the FLAME infrastructure is spread over Univerity of Bristol, Millennium Square, We The Curious Museum, and Multimedia Shed. The sites are interconnected via a 10km long fibre link across the city. The Flame platform topology is shown in Fig. 5.41.

Millenium Square is about 75m by 75m and each of the six towers is roughly 3m wide. Due to the close proximity of all WiFi access points on the square two options are available to experimenters (Fig. 5.42):

- a single SSID across the entire square where all traffic is served by a single

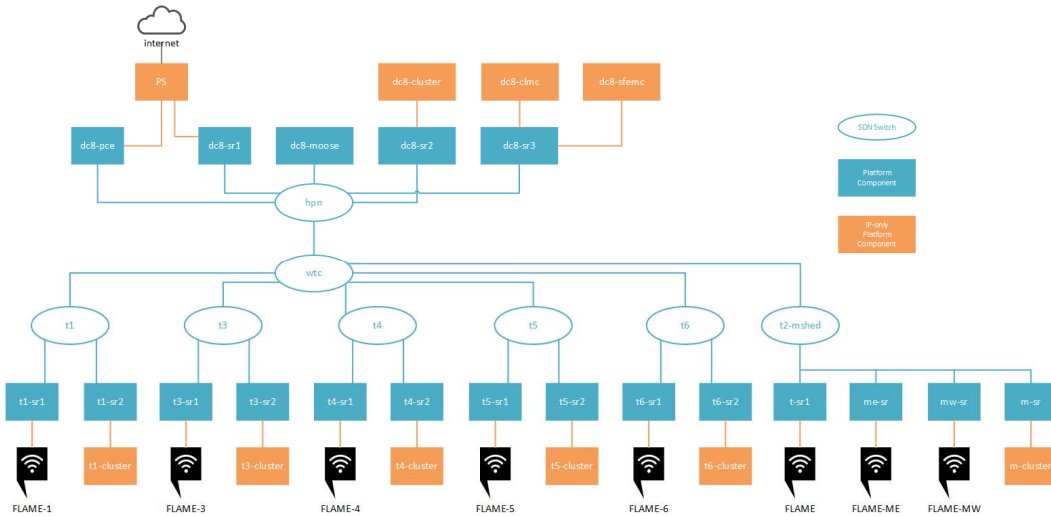


Figure 5.41: Bristol Flame platform topology [90]

SR located at Tower 2. The WiFi access points perform some sort of mobility to let clients connect to the strongest access point - but fully transparent to the clients in case of a "handover";

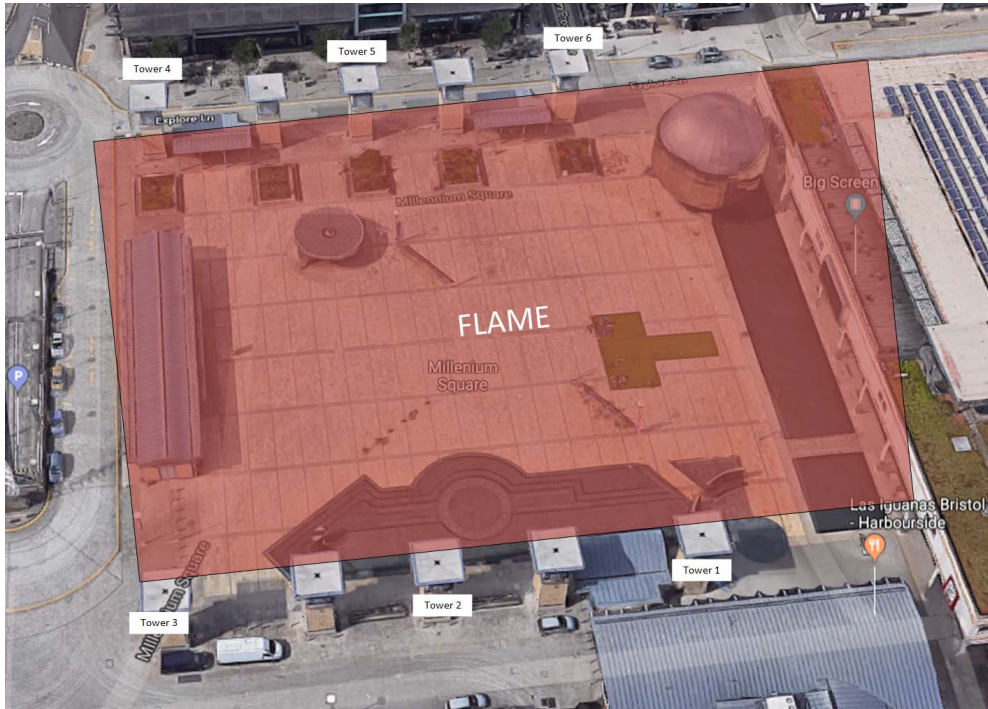
- a dedicated SSID per across all towers (except Tower 2) which allows experimenters to use some sort of location beacon (QR code, tower selection by user, etc) to determine which SSID to be used. Each access point uses a sector antenna to lower interference across all APs.

### 5.4.2 VISION architecture

VISION is composed of three architectural elements (see Fig. 5.43), specifically the *DiMoVis platform* and *Tourist Eyes* (described in the previous section) and the *Lying Person Recognition* (LPR). So, the VISION project represents the possibility of integrating the three components together and inside the FLAME platform. In addition, user-provided video sources and mobile devices, that will be connected to the Flame Access Network, complete the system.

The DiMoVis platform runs over the FLAME platform to provide basic video surveillance services and the possibility of customizing them with additional features. This platform allows development of a multitude of new services for smart cities. In this project it is used to support two third-parties services for impaired persons: persons that need some help because lying on the ground and are not able to request help autonomously, and blind tourists. The way in which DiMoViS considers third party applications should be emphasized: these applications are





(a) Single SSID across the square



(b) Dedicated SSID per across all towers

Figure 5.42: FLAME. Coverage area in Millennium Square [90]

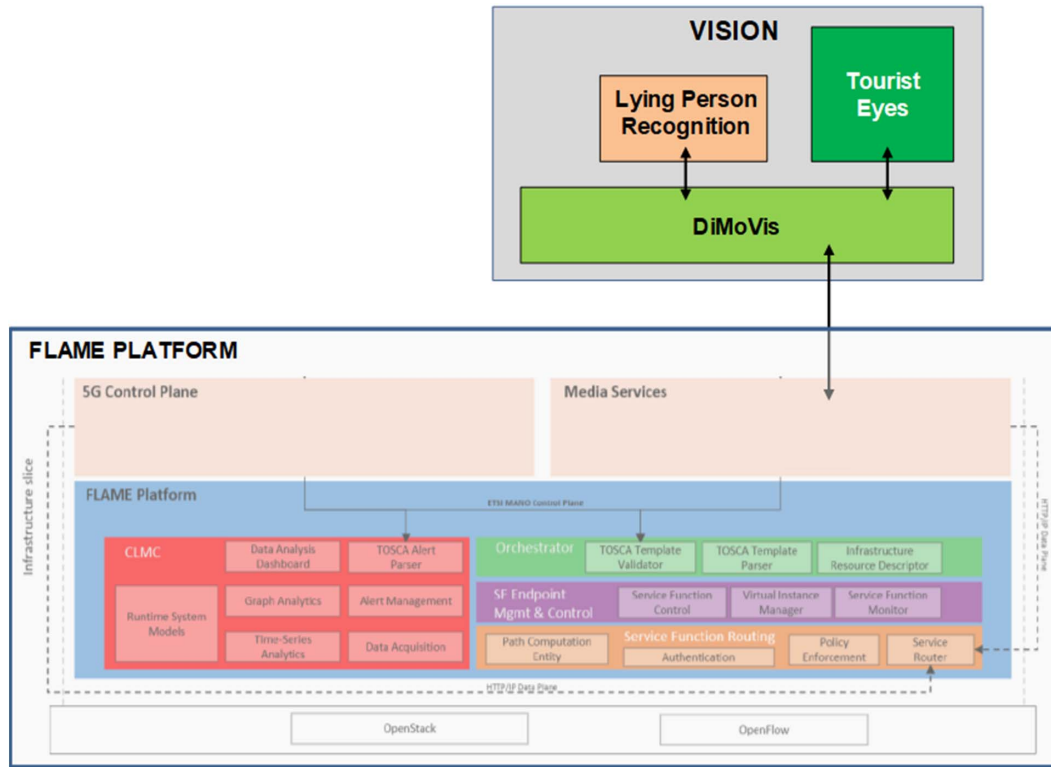


Figure 5.43: VISION Architecture and interaction with the FLAME Platform

seen as DiMoViS users. This means that, for example, when Tourist Eyes decides to use the services offered by DiMoViS, it will have to log in to the platform as if it were a normal end-user and only then can it register its own cameras on DiMoViS and/or request access to cameras managed by the latter.

Each user of the platform can easily share his own video source device (e.g. webcam, IP cam, smartphone or tablet) by connecting it to the network (with either a WiFi or a cellular connection), and register it to the DiMoVis platform through an Android app or a web portal. During registration, the owner of the video source device can specify one or more access profiles to restrict access to different groups of users. Each user, on the other side, through the same Android app or web portal, has a map of the area covered by the DiMoVis platform service (e.g. a smart city), with all the active cameras. Association of one or more cameras is done by a registered user in a very easy way, by clicking on the map viewed on the screen, or through a QR-code that is available in proximity of the camera. The user can then customize the received video flows and the events associated with each camera by adding optional functions running in the underlying softwarized network as VF (for example, motion detection, mosaic view, video transcoding, and so on). These functions will be automatically included in the chains between

video sources and the device of the requesting users.

The Impaired Person Support element, proposed specifically for the VISION experiment, aims at providing support to impaired persons to be remotely guided by a relative or a friend (in the following called remote guide), for example when he/she has lost his spatial reference points or if he/she wants to reach an unknown place. More specifically, the impaired person, walking in an area covered by the VISION service, can request a remote visual guide. So he/she starts a phone call with the remote guide, informing him/her about his approximate position and the remote guide activates the VISION app. By browsing on the map, the remote guide selects all the cameras that are available in the zone where the impaired person might be and, looking at the video flows received by these cameras, starts to guide the person by speaking on the phone. The remote guide can follow the person during his path by means of different video flows transmitted by various cameras. This service is complementary to the Tourist Eyes block, since it can be invoked either when the impaired tourist is on a path registered in the Tourist Eyes platform and needs to reach a place not included in the PoI list, or when he needs to reach a Tourist Eyes path from an external place. All the IP cameras that are installed for the Tourist Eyes service can be associated with the DiMoVis platform, and so can be used by all the services provided by it. The Impaired Person Support element, thanks to the installed cameras, can also automatically recognize if a person is lying on the ground in indoor or outdoor environments, and send an alert to someone (e.g. a friend or a relative of that person, or to the local emergency or healthcare provider). For example, this service can be activated by a relative of an old person if he/she wants to monitor the old person. In this case, video flows generated by all the IP cameras installed close to the places where the impaired person moves, are sent to an artificial intelligence tool running inside the network as VF, which is able to recognize persons lying on the ground through an image recognition algorithm. If a person is realized to lay down in an area served by a camera where the person monitored also through the Impaired Person Support element is located, an alert will be sent to the relative who requested to monitor the impaired person.

Finally, the proposed platform is completed with the external devices, represented by the user-provided IP cameras and the mobile devices, both connected via WiFi/LTE/5G to the FLAME platform. Video sources, on the one hand, are any kind of video transmitting devices (like IP cameras, action cams, cameras installed on-board drones, etc) transmitting HTTP-based flows or, alternatively, any kind of cameras whose flows are transcoded to this format with a software

running on a Raspberry Pi board attached to the IP camera. Mobile devices, on the other hand, are Android smartphones or tablets where the VISION app will be installed.

The architectural structure of DiMoViS, in this project, differs from the one mentioned in the section 5.1. It is constituted by the following elements:

- *Front-end DiMoViS*: this component represents the interface between the users/third-part applications and the other internal components of DiMoViS. It maintains a database with the tables of registered users of the DiMoViS service, of the cameras, and of all the security groups that allow to add a level of security in accessing the cameras. Thanks to these security groups, when a user wants to view the list of cameras to be accessed, the results of this query will be conditioned by the security group settings made. Cameras that will be set with "Open" access, will be directly accessible by each user, while cameras set with particular security groups, will require a first step in which the user interested in that stream owes the camera owner the possibility of access it. The DiMoViS Front-end, in addition to managing the database and providing an interface to users, also manages the exchange of basic information between the other components inside the system.
- *Digital Twin (DT)*: the presence of this component is necessary depending on the type of camera that is installed in the system. Based on the mechanism used to send the video stream, we can identify two types of cameras: those that send the stream only on specific request, and those configured in such a way as to continuously send the stream to an external server. The use of the DT is necessary with the first type of cameras: its task is in fact to send the request to the camera and forward the received video stream to the FlowReplicator-EA. To do this, it uses FFmpeg [91], a complete, cross-platform solution for recording, converting and stream audio and video. Specifically, it will behave as a client towards the FlowReplicator-EA. For the cameras used in this project, the request uses the Real Time Streaming Protocol (RTSP));
- *FlowReplicator EA (FR-EA)*: this component also implements FFmpeg. It acts as fserver for the DT and the FlowReplicatorPA, since from the first it receives the video stream coming from the camera while from the second it receives the requests for the retransmission of the stream. Inside it uses a configuration file to encode the stream according to what is specified in the

request received from the DT: the stream coming from the same source can therefore be treated with different qualities in terms of audio and video. It is launched in the edge node near the DT and camera, in such a way to reduce latency in the video flow in ingress to the network;

- *FlowReplicator PA* (FR-PA): it is launched in the user’s network access edge node. Precisely because of its proximity to the user, it takes care of managing user’s requests for receiving a video stream. This component also implements FFMEG and, as the FR-EA, uses a configuration file for retransmission of the stream to the user who requested it.

and it works according to the following steps:

1. IP Camera registration: Each IP Camera is connected to a WiFi/LTE/5G access point and then registered to the database of the Front-end DiMoViS. Since now, they can be used by both the DiMoViS mobile and fixed users (Fig. 5.44);
2. a user logs into the DiMoViS service through the app, contacting the Front-end, receives a list of available cameras and chooses which one he want to see (Fig. 5.45);
3. if not yet active so far, a DT and a FR-EA are run at the edge of the network near the camera, on the MEC server closest to its access point, while a FR-PA is run on the MEC server closest to the user. The Front-end communicates to the DT the IP of the camera to which to request the video stream and the IP of the FR-EA to which to send the video, the user communicates the IP of the FR-PA to which to send the request and FR-PA starts the IP of the FR-EA to which to forward the flow request. If these three components are already in running state and are transmitting the requested video flow, the Front-end immediately communicates the IP of the FR-EA to the user (Fig. 5.46);
4. if not yet active, the communication between DT, FR-EA and FR-PA starts and the user receives the video flow in the app (Fig. 5.47).

When a Blind User requests a Tourist Eyes service for a given path, the following steps, will be performed (Fig. 5.48: the request of the Tourist Eyes user arrives to the Tourist Eyes server; if some cameras of the required path is managed by DiMoViS platform, Tourist Eyes server starts a connection with the Front-end

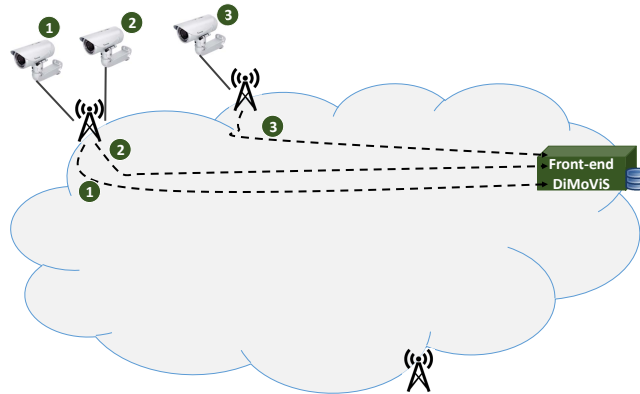


Figure 5.44: VISION. DiMoViS service: cam registration

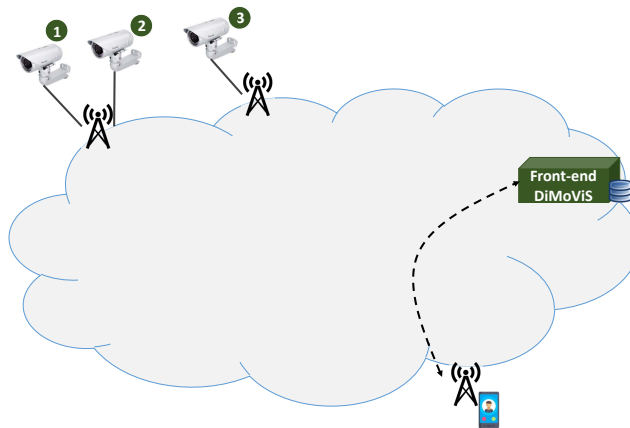


Figure 5.45: VISION. DiMoViS service: user access and cam selection

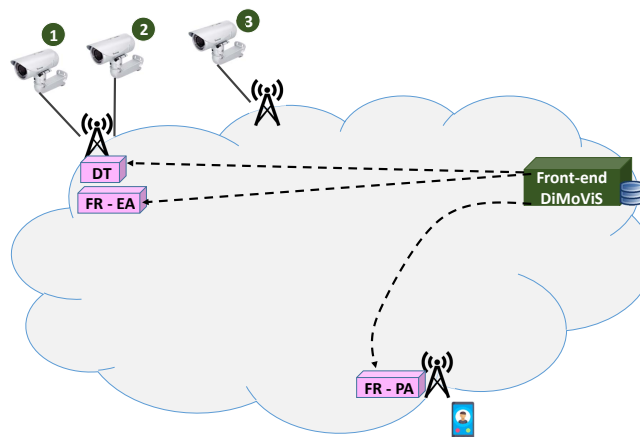


Figure 5.46: VISION. DiMoViS service: power on, if required, of the other components

DiMoViS. As said before, Tourist Eyes server connects to DiMoViS with username and password, because for DiMoViS service it appears like a user. The Tourist Eyes server requests a set of video flows to the Front-end DiMoViS, exactly using

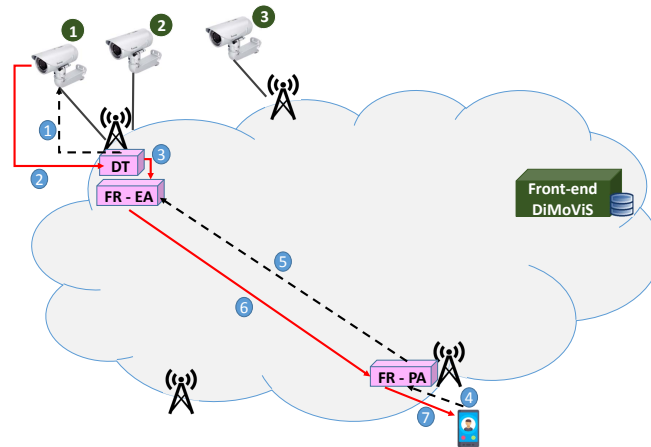


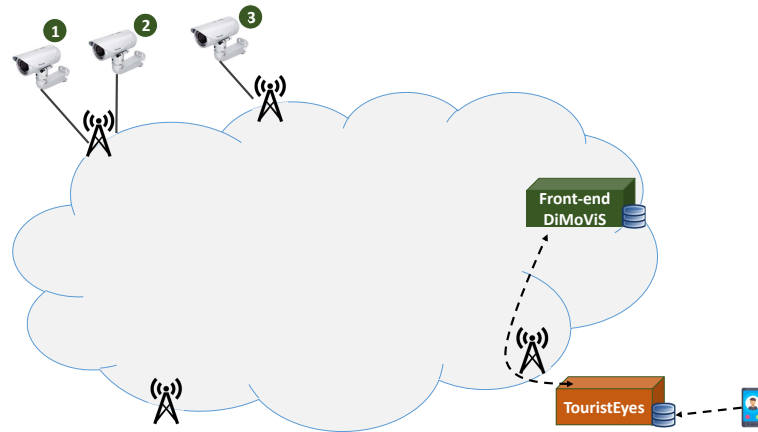
Figure 5.47: VISION. DiMoViS service: video flow transmission

the same steps listed above for a user of the basic DiMoViS service. At this point, the same procedure described in the above steps 3 and 4 are executed, with the difference that the request for a specific video flow is performed by the Tourist Eyes server and not by the final user. The video flow received by the Tourist Eyes server is analyzed as described in section ??.

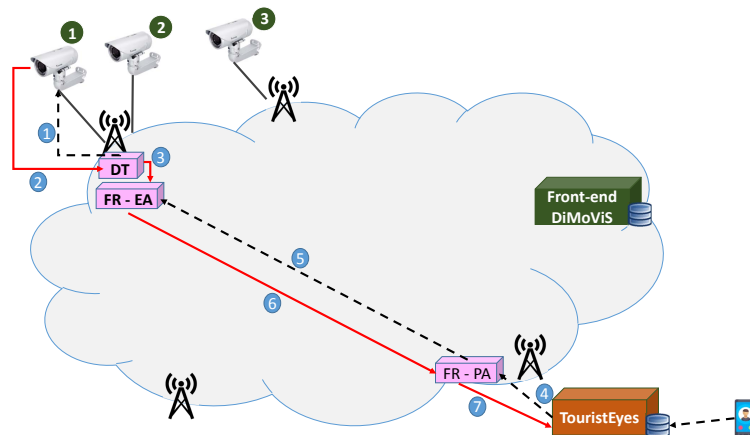
When a User requests a LPR service for a given relative who is moving, the same steps described above for the Tourist Eyes case are performed. Also in this case, LPR server has to connect to the Front-end DiMoViS as an user with username and password, and finally it will receives the video flow directly. The LPR server starts to analyze video it is receiving and, as soon as it recognizes a lying person in one of the monitored areas, it sends an alert message on the smartphone of the requesting user as a notification, including the position of the relevant IP camera and a picture captured by that camera (Fig. 5.49).

Fig. 5.50 shows the interaction between each level of the whole system and the key stakeholders. The *Infrastructure Operator* and the *FLAME platform provider* coincide with the stakeholders defined in the official FLAME documentation, specifically in Deliverable D3.2 [87]. More specifically, the Infrastructure Operator deploys the FLAME platform on the infrastructure in a smart city, and is able to manage specific resource requirements that deployment and usage of the FLAME platform brings. The FLAME platform provider is the provider of the software platform, including all the software components required to support the VISION service (e.g. orchestration and deployment of services, networking technologies, and configuration, planning, monitoring and control systems).

Another stakeholder is the *DiMoVis-based Service Developer*, whose task is the deployment of new services using the distributed video surveillance service pro-



(a) VISION. Tourist Eyes service: connection to Front-end and cam selection



(b) VISION. Tourist Eyes service: video flow transmission

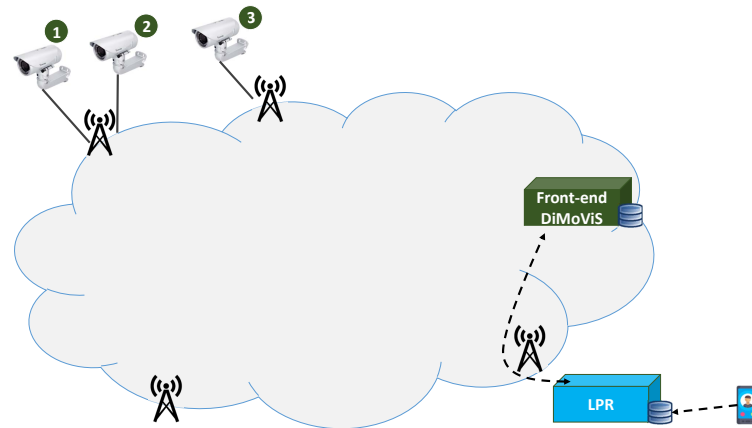
Figure 5.48: VISION: interactions between DiMoViS platform and Tourist Eyes service

vided by DiMoViS. Two examples of these services are Tourist Eyes and LPR, included in this experiment as user-level services.

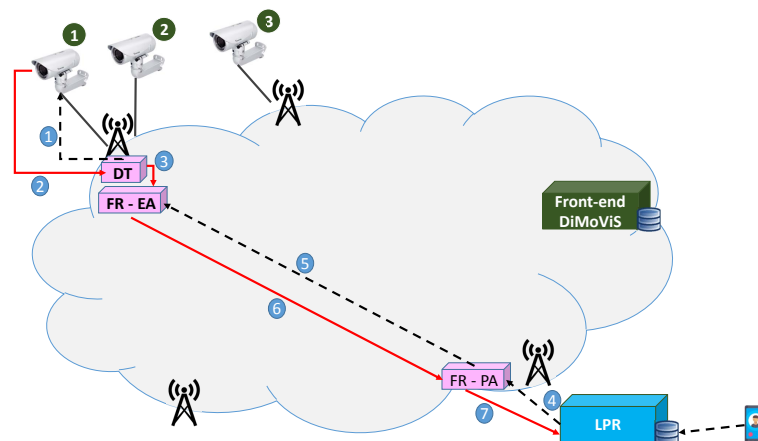
Stakeholders of the Tourist Eyes service are *Blind Tourists*, who are the Tourist Eyes service recipients, and the *Tourist Office*, who is in charge of managing the Tourist Eye service, defining the tourist paths for blind people, and deciding the IP cameras to be included in the service.

On the other side, stakeholders of the LPR service are: *Relatives of impaired persons*, who start and configure the service indicating the IP cameras to be used to monitor the lifetime of the relatives and *Old/Impaired persons*, who are the monitored persons. Additional stakeholders may be *Hospital personnel* that are automatically alerted when the system has deduced that there is a person lying





(a) VISION. LPR service: connection to Front-end and cam selection



(b) VISION. LPR service: video flow transmission

Figure 5.49: VISION: interactions between DiMoViS platform and LPR service

on the ground.

Finally, additional stakeholders at the user level, directly connected to the DiMoViS platform, are any stakeholder of the smart city, like for example *Law Enforcement*, or simple *citizens* that request for monitoring of some areas of the city.

Thanks to the exploitation of the FLAME technology, a number of key advantages are achieved, mainly aimed at assessing the effectiveness of the proposed platform. Advantages are:

- reduction of network traffic, with consequent performance improvements. In fact, thanks to the usage of some patterns made available by the FLAME platform, the data stream generated by each video-flow source is automatically rerouted to the “interested receiver(s)” only, in a point-to-multipoint

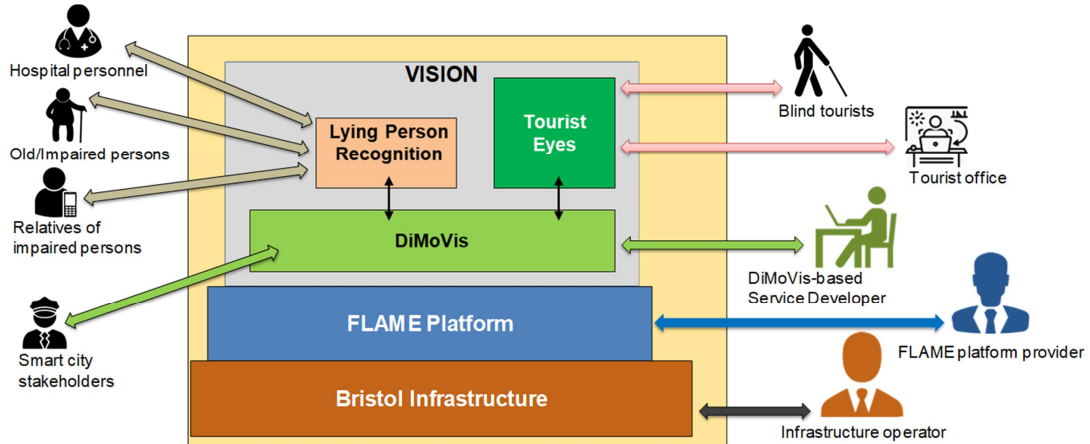


Figure 5.50: Vision. System Architecture and Stakeholders

fashion, within the network. The gain is more evident when destination users access the network through the same ingress node and, even more, if this node is the same access node also for the selected video sources;

- scalability: network traffic does not increase when a user requesting a given data flow accesses the network from an access node where at least another user is receiving the same flow;
- low end-to-end latency. There are two reasons for this: on the one hand, the fact that video flows are directly managed inside the network instead of being forwarded to external servers providers; on the other hand, the application of the MEC paradigm will maintain end-to-end latency low even in presence of function offloading, given that time-critical VFs are provided to the users directly by their access nodes.
- OpEx and CapEx reduction;
- general advantage related to the softwarization of services usually implemented in hardware. The use of software tools running on general-purpose hardware allows to decrease costs incurred for installing and maintaining hardware;
- ease in service deployment: installation of new cameras or other video-flow sources is trivial because they do not need to be configured. Destinations of video streams are automatically decided and located according to the requests coming from the users;

- ease in VISION platform extension. Platform is able to support a large number of personalized services (e.g. video cryptography, area monitoring, area obscuring, target follower, mosaic) installed as VFs, and included in the service chains according to the users' requirements.

The target of the VISION experiment is to achieve a high degree of innovation by taking into account multiple features such as personalization, interactivity, mobility and localisation (PIML). A high level of personalization can be determined thanks to the diversity of offered and customized services that each user can compose as building blocks by choosing not only IP cameras but also additional services. This comports an increasing of the users satisfaction. About interactivity, it concerns the ability of the service to follow and respond to user requests in real time. Interactivity relies on mobility, intended as the ability to enable a service provider to meet user's requests in an efficient way by exploiting the fact that the content delivery system must know and exploit user location. Finally, localisation is regarded as the awareness of actual user location to guarantee context-aware content delivery and caching strategy. More specifically, according to the VISION experiment execution, the following PIML will be achieved:

- *personalisation*: the set of cameras each user can request to receive the required video flows varies according to the user profile. For example, law enforcement officers might be allowed to access all public cameras and plenty of private cameras. Another aspect of personalisation for the VISION project regards the simplification of IP camera selection based on suggestion for per-user experience. This is based on the introduction of easy strategies that store IP camera selection habits by each users, and the additional service selection habits (e.g. on the basis of the selected cameras by the same user or users with a similar profile), in order to automatically propose default IP cameras and additional services;
- *interactivity*: the users will interact with VISION in multiple ways. On one hand, some users behave as video service providers, as they will be able to connect IP cameras to the network platform, register the cameras using the Vision app and assign them to a specific security group. On the other side, the user behaving as Vision client, will be able to explore the map of a smart city, choose the video cameras of interest and additional services for each video flow;
- *mobility*: the user will be enabled to receive video flows in real-time, regardless of his/her location, and also in mobility. Mobility is also supported for

IP cameras that can be installed on mobile vehicles. In fact, the system, thanks to some elements available inside the FLAME platform, is able to recognize changes in the access point used by IP cameras and users, and automatically reconfigure routing to maintain the multipoint-to-multipoint connection at the application level;

- *localisation*: in a scenario of Impaired Person Support, localisation is a key feature, as long as it allows the parent / relatives / friends of the impaired person who, for any reason, gets lost or needs help in a smart city, to localize him/her using the Vision app. In addition, another aspect of localisation included in the VISION project regards the possibility of automatically changing selected cameras (or proposing alternative cameras) according to the context around each user, his/her current position and his/her behavior. For example, if a user is travelling a road, and thanks to a track of his/her position the system has clue that he/she is moving along a given route, the VISION system will change the IP camera selection automatically to allow him/her to monitor the streets where the user is arriving to.

# Chapter 6

## Conclusion

This thesis aimed to analyze aspects related to the network softwarization in the context of 5G ecosystems. In particular, three main activity were addressed in this field.

The first regarded the management and orchestration of network slices, and two different works were presented in this thesis: one about the handover management in the RAN portion of a network slice, with the implementation of a network service that is able to detect handover packets between physical and virtual base stations. The captured messages are analyzed and communicated to the Network Orchestrator using publisher/subscribe approach. The implementation of this solution and the numerical results showed the potentiality of the proposed framework, allowing the Orchestrator to allocate resources to prepare the network to manage the handover, or to block this last one, sending an alert trigger to the Mobility Management Entity block of the RAN, if the resources are limited or the network behind the RAN of the new area presents congestion or faults. The other work regards the use of UAVs to extend network slice to provide computing and network facilities to IoT devices for area monitoring applications. Cases with and without collaboration between UAVs were analyzed. In particular, the study was conducted in four steps: first, an analytical model was proposed; then, a simulator was implemented in Matlab to evaluate the behaviour of a fleet of UAV. Next, the Reinforcement Learning technique was introduces considering with and collaboration cases. The numerical results, for each step, compared with the state of the art, had always highlighted the advantages of the proposed solution, allowing to identify the optimal values of the key parameters of the problem addressed.

The second activity involved the use of network slice for vertical applications. More in deep, a work was proposed about the implementation of the Tactile Support Engine (TSE) inside a Tactile Internet network slice. It is the compo-

ment that implementing Artificial Intelligence techniques, supports the network to respect application requirements in case of problems. A use case regarding a remote control of a game inspired to the famous video game "Subway Princess Runner" was set up to show how the proposed TSE works and the achieved performance. Another work regarding Vehicular Networks and the use of MEC servers installed along the roadway. This solution aims at providing facilities to the vehicles during the processing of the data collected from the environment in a context of a smart city. The first results showed the capability of the proposed framework, with improvement of the job management performed by vehicles. In this context, a future work will be the implementation of collaboration between vehicles, selecting one vehicle as leader with the aim of processing data received by the normal vehicles and, in case of overload, implement an offloading policy toward MEC server installed along the road.

In the context of the European Projects, the following framework were proposed: DiMoViS in the Triangle Project, Tourist Eyes and 5Gamer in 5GINFIRE Project and Vision in Flame Project. Regarding DiMoViS, it was possible to implement a distributed video surveillance system in a 5G environment, discovering all the criticalities that a service like this can have and what are the requirements that the network must meet in order to guarantee optimal QoS and QoE to the end user. Even 5Gamer, a revival of the famous Pong with the addition of Artificial Intelligence that trains according to the gaming habits of the physical player, has made possible to explore the field of video games, with all the necessary requirements to be able to guarantee the user to play at best, without noticing any delays or other problems in the network. TouristEyes and Vision are two experiments that have had a double value. On the one hand, the possibility of building increasingly advanced video distribution platforms in a 5G network context, introducing Artificial Intelligence to analyze video streams in real time and carry out people recognition. On the other hand, it was possible to develop two services aimed at people with disabilities by exploiting the potential offered by the 5G network. With Tourist Eyes, it was proposed a platform to allow blind people to be able to move in city contexts that are unfamiliar to them, without any support (such as a guide dog, either because not available or due to allergies of the subject), not only in outdoor contexts but also indoors, where GPS technology doesn't work. In Vision, the presence of the LPR service allows, through the analysis of the video coming from internal and external cameras, to recognize lying people, in order to give them help by sending alerts to relatives and/or law enforcement agencies.

# List of publications

1. A. Lombardo, C. Rametta and C. Grasso, "A Network-Assisted Platform for Multipoint Remote Learning," 28th International Tyrrhenian Workshop (TIWDC), Palermo, Italy, 2017, pp 183-196.
2. G. Baldoni, C. Grasso, A. Lombardo, C. Rametta, A. Scala and S. Sotile, "Set-top box virtualization as a personal cloud server for 5G users," 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, 2018, pp. 1-3.
3. C. Grasso and G. Schembra, "Design of a UAV-Based Videosurveillance System with Tactile Internet Constraints in a 5G Ecosystem," 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 2018, pp. 449-455.
4. F. D'Urso, C. Grasso, C. Santoro, F. F. Santoro and G. Schembra, "The Tactile Internet for the flight control of UAV flocks," 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 2018, pp. 470-475.
5. C. Grasso, R. Raftopoulos and G. Schembra, "The Triangle Platform for End-to-End Performance Analysis of a 5G Video Transmission Network Slice," 15th International Symposium on Wireless Communication Systems (ISWCS), Lisbon, Portugal, 2018.
6. L. Galluccio, C. Grasso, S. Milardo, G. Schembra and E. Sciacca, "An Experimental Testbed for Managing BAN Services at the Network Edge," 14th International Conference on Network and Service Management (CNSM), Rome, Italy, 2018.
7. C. Grasso and G. Schembra, "A Fleet of MEC UAVs to Extend a 5G Network Slice for Video Monitoring with Low-Latency Constraints," Published in

## LIST OF PUBLICATIONS

---

- MDPI Special Issue on Softwarization at the Network Edge for the Tactile Internet, January 2019, Volume 8, Issue 1,3.
8. G. Faraci, C. Grasso and G. Schembra, "Reinforcement Learning for Management of a 5G Network Slice Extension with UAV's," 2019 IEEE INFOCOM Workshops: SMILING 2019: Sustainable networking through Machine Learning and Internet of Things, Paris, 2019.
  9. C. Grasso and G. Schembra, "5G-Hander: A Network Service for Handover Detection in 5G Networks," Special issue article on Internet Technology Letters, Wiley Online Library.
  10. L. Galluccio, C. Grasso, M. Grasso, R. Raftopoulos and G. Schembra, "Measuring QoS and QoE for a Softwarized Video Surveillance System in a 5G Network," 2019 IEEE International Symposium on Measurements and Networking, Catania, 2019.
  11. G. Faraci, C. Grasso and G. Schembra, "Fog in the Clouds: UAVs to Provide Edge Computing to IoT Devices," AMC Transactions on Internet Technology, Special issue on Evolution of IoT Networking Architectures.
  12. G. Faraci, C. Grasso and G. Schembra, "Design of a 5G Network Slice Extension with MEC UAVs Managed with Reinforcement Learning," to appear on IEEE JSAC, Special Issue on Advances in Artificial Intelligence and Machine Learning for Networking.
  13. F. Busacca, C. Cirino, G. Faraci, C. Grasso, S. Palazzo and G. Schembra, "Multi-Layer Offloading at the Edge for Vehicular Networks," 2020 IEEE MedComNet, Arona, 2020.
  14. C. Grasso, K. Eswar, P. Nagaradjane, M. Ramesh and G. Schembra, "Designing the Tactile Support Engine to Assist Time-Critical Applications at the Edge of a 5G Network," Submitted to Computer Communications, Special Issue on Network Intelligence.



# References

- [1] Cisco. *White paper: Cisco Annual Internet Report (2018-2023)*. Tech. rep. Mar. 2020. URL: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [2] G. A. Akpakwu et al. “A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges”. In: *IEEE Access* 6 (2018), pp. 3619–3647.
- [3] A. Al-Fuqaha et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376.
- [4] 5G PPP Architecture Working Group. *White paper: View on 5G Architecture*. Tech. rep. Feb. 2020. DOI: 10.5281/zenodo.3265031. URL: <https://zenodo.org/record/3265031#.Xyu3yK9xeMo>.
- [5] A. Banchs et al. “A 5G Mobile Network Architecture to Support Vertical Industries”. In: *IEEE Communications Magazine* 57.12 (2019), pp. 38–44.
- [6] CNIT. *The 5G Italy Book 2019: a Multiperspective View of 5G*. CNIT, 2019.
- [7] ETSI 3GPP. *5G; Study on Scenarios and Requirements for Next Generation Access Technologies*. Tech. rep. May 2017. URL: [https://www.etsi.org/deliver/etsi\\_tr/138900\\_138999/138913/14.02.00\\_60/tr\\_138913v140200p.pdf](https://www.etsi.org/deliver/etsi_tr/138900_138999/138913/14.02.00_60/tr_138913v140200p.pdf).
- [8] Justine Sherry et al. “Making Middleboxes Someone Else’s Problem: Network Processing as a Cloud Service”. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–24. ISBN: 9781450314190. DOI: 10.1145/2342356.2342359. URL: <https://doi.org/10.1145/2342356.2342359>.

## REFERENCES

---

- [9] NFV White Paper. “Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges Call for Action. Issue 1”. Oct. 2012.
- [10] B. Han et al. “Network function virtualization: Challenges and opportunities for innovations”. In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97.
- [11] Hyuncheol Kim et al. “Service platform and monitoring architecture for network function virtualization (NFV)”. In: *Cluster Computing* 19 (Sept. 2016). DOI: 10.1007/s10586-016-0640-3.
- [12] ETSI ISG. *Network Function Virtualisation (NFV); Architectural Framework*. Tech. rep. Oct. 2013. URL: [https://www.etsi.org/deliver/etsi\\_gs/NFV/001\\_099/002/01.01.01\\_60/gs\\_NFV002v010101p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf).
- [13] Margaret Chiosi et al. “Network Functions Virtualisation (NFV) Network Operator Perspectives on Industry Progress”. In: (Oct. 2013). DOI: 10.13140/RG.2.1.4110.2883.
- [14] M. Veeraraghavan et al. “Network Function Virtualization: A Survey”. In: *IEICE Trans. Commun.* 100-B (2017), pp. 1978–1991.
- [15] A. Fischer et al. “Virtual Network Embedding: A Survey”. In: *IEEE Communications Surveys Tutorials* 15.4 (2013), pp. 1888–1906.
- [16] P. v. Anvith et al. “A Survey on Network Functions Virtualization for Telecom Paradigm”. In: *2019 TEQIP III Sponsored International Conference on Microwave Integrated Circuits, Photonics and Wireless Networks (IM-ICPW)*. 2019, pp. 302–306.
- [17] ETSI ISG. *Network Function Virtualisation (NFV); Use Cases*. Tech. rep. May 2017. URL: [https://www.etsi.org/deliver/etsi\\_gr/NFV/001\\_099/001/01.02.01\\_60/gr\\_nfv001v010201p.pdf](https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf).
- [18] J. Gil Herrera and J. F. Botero. “Resource Allocation in NFV: A Comprehensive Survey”. In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 518–532.
- [19] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.
- [20] Zohaib Latif et al. “A comprehensive survey of interface protocols for software defined networks”. In: *Journal of Network and Computer Applications* 156 (2020), p. 102563. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2020.102563>. URL: <http://www.sciencedirect.com/science/article/pii/S1084804520300370>.

## REFERENCES

---

- [21] Sachin Sharma et al. “Automatic bootstrapping of OpenFlow networks”. In: Apr. 2013, pp. 1–6. ISBN: 978-1-4673-4984-0. DOI: 10.1109/LANMAN.2013.6528283.
- [22] Antonio Manzalini et al. “Software-Defined Networks for Future Networks and Services: Main Technical Challenges and Business Implications”. In: <http://sites.ieee.org/sdn4fns/whitepaper/> (Feb. 2014).
- [23] Y. Li and M. Chen. “Software-Defined Network Function Virtualization: A Survey”. In: *IEEE Access* 3 (2015), pp. 2542–2553.
- [24] D. Drutskoy, E. Keller, and J. Rexford. “Scalable Network Virtualization in Software-Defined Networks”. In: *IEEE Internet Computing* 17.2 (2013), pp. 20–27.
- [25] ETSI ISG. *Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework*. Tech. rep. Dec. 2015. URL: [https://www.etsi.org/deliver/etsi\\_gs/NFV-EVE/001\\_099/005/01.01.01\\_60/gs\\_NFV-EVE005v010101p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf).
- [26] F. Z. Yousaf et al. “NFV and SDN—Key Technology Enablers for 5G Networks”. In: *IEEE Journal on Selected Areas in Communications* 35.11 (2017), pp. 2468–2478.
- [27] J. Ordonez-Lucena et al. “Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges”. In: *IEEE Communications Magazine* 55.5 (2017), pp. 80–87.
- [28] L. U. Khan et al. “Network Slicing: Recent Advances, Taxonomy, Requirements, and Open Research Challenges”. In: *IEEE Access* 8 (2020), pp. 36009–36028.
- [29] X. Foukas et al. “Network Slicing in 5G: Survey and Challenges”. In: *IEEE Communications Magazine* 55.5 (2017), pp. 94–100.
- [30] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Tech. rep. Sept. 2011. URL: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>.
- [31] F. Polash, A. Abuhussein, and S. Shiva. “A survey of cloud computing taxonomies: Rationale and overview”. In: *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*. 2014, pp. 459–465.

## REFERENCES

---

- [32] F. Fowley et al. “A Classification and Comparison Framework for Cloud Service Brokerage Architectures”. In: *IEEE Transactions on Cloud Computing* 6.2 (2018), pp. 358–371.
- [33] Y. Chen, X. Li, and F. Chen. “Overview and analysis of cloud computing research and application”. In: *2011 International Conference on E-Business and E-Government (ICEE)*. 2011, pp. 1–4.
- [34] W. Shi, G. Pallis, and Z. Xu. “Edge Computing [Scanning the Issue]”. In: *Proceedings of the IEEE* 107.8 (2019), pp. 1474–1481.
- [35] A. Papageorgiou, B. Cheng, and E. Kovacs. “Real-time data reduction at the network edge of Internet-of-Things systems”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 284–291.
- [36] Ying Gao et al. “Are Cloudlets Necessary”. In: 2015.
- [37] T. Taleb et al. “On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration”. In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1657–1681.
- [38] ETSI ISG. *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. Tech. rep. Jan. 2019. URL: [https://www.etsi.org/deliver/etsi\\_gs/MEC/001\\_099/003/02.01.01\\_60/gs\\_MEC003v020101p.pdf](https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf).
- [39] ETSI ISG. *Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NFV environment*. Tech. rep. Feb. 2018. URL: [https://www.etsi.org/deliver/etsi\\_gr/MEC/001\\_099/017/01.01.01\\_60/gr\\_MEC017v010101p.pdf](https://www.etsi.org/deliver/etsi_gr/MEC/001_099/017/01.01.01_60/gr_MEC017v010101p.pdf).
- [40] “Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks”. In: *Entropy* 20.1 (Dec. 2017), p. 4. ISSN: 1099-4300. DOI: 10.3390/e20010004. URL: <http://dx.doi.org/10.3390/e20010004>.
- [41] Christian Grasso and Giovanni Schembra. “5G-Hander: A network service for handover detection in 5G networks”. In: *Internet Technology Letters* 2.4 (2019), e110. DOI: 10.1002/itl2.110. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.110>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.110>.
- [42] ETSI-NFV-Group-Spec. *Network Functions Virtualisation (NFV); Management and Orchestr.* Tech. rep. ETSI, 2014.

## REFERENCES

---

- [43] ETSI 3GPP. *LTE; Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 Application Protocol (X2AP)*. Tech. rep. Sept. 2014. URL: [https://www.etsi.org/deliver/etsi\\_ts/136400\\_136499/136423/12.03.00\\_60/ts\\_136423v120300p.pdf](https://www.etsi.org/deliver/etsi_ts/136400_136499/136423/12.03.00_60/ts_136423v120300p.pdf).
- [44] ETSI 3GPP. *LTE; Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 Application Protocol (S1AP)*. Tech. rep. Sept. 2014. URL: [https://www.etsi.org/deliver/etsi\\_ts/136400\\_136499/136413/12.03.00\\_60/ts\\_136413v120300p.pdf](https://www.etsi.org/deliver/etsi_ts/136400_136499/136413/12.03.00_60/ts_136413v120300p.pdf).
- [45] PCAP. *Manpage of PCAP*. <https://www.tcpdump.org/manpages/pcap.3pcap.html>. Accessed August 2020.
- [46] IETF-RFC-2960. *Stream Control Transmission Protocol*. <https://tools.ietf.org/html/rfc2960>. Accessed August 2020.
- [47] ETSI 3GPP. *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (3GPP TS 36.300 version 9.4.0 Release 9)*. [https://www.etsi.org/deliver/etsi\\_ts/136300\\_136399/136300/09.04.00\\_60/ts\\_136300v090400p.pdf](https://www.etsi.org/deliver/etsi_ts/136300_136399/136300/09.04.00_60/ts_136300v090400p.pdf). Accessed August 2020.
- [48] ETSI 3GPP. *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer - Measurements (3GPP TS 36.214 version 9.1.0 Release 9)*. [https://www.etsi.org/deliver/etsi\\_ts/136200\\_136299/136214/09.01.00\\_60/ts\\_136214v090100p.pdf](https://www.etsi.org/deliver/etsi_ts/136200_136299/136214/09.01.00_60/ts_136214v090100p.pdf). Accessed August 2020.
- [49] C. Rametta and G. Schembra. “Designing a Softwarized Network Deployed on a Fleet of Drones for Rural Zone Monitoring”. In: *Future Internet* 9.1 (Mar. 2017), p. 8. ISSN: 1999-5903. DOI: 10.3390/fi9010008. URL: <http://dx.doi.org/10.3390/fi9010008>.
- [50] C. Grasso and G. Schembra. “Design of a UAV-Based Videosurveillance System with Tactile Internet Constraints in a 5G Ecosystem”. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 449–455.
- [51] C. Grasso and G. Schembra. “A Fleet of MEC UAVs to Extend a 5G Network Slice for Video Monitoring with Low-Latency Constraints”. In: *Journal of Sensor and Actuator Networks* 8.1 (Jan. 2019), p. 3. ISSN: 2224-2708. DOI: 10.3390/jsan8010003. URL: <http://dx.doi.org/10.3390/jsan8010003>.

## REFERENCES

---

- [52] Giovanni Schembra, Giuseppe Faraci, and Christian Grasso. “Fog in the Clouds: UAVs to Provide Edge Computing to IoT Devices”. In: *ACM Trans. Internet Technol.* 0,ja (). ISSN: 1533-5399. DOI: 10.1145/3382756. URL: <https://doi.org/10.1145/3382756>.
- [53] G. Faraci, C. Grasso, and G. Schembra. “Design of a 5G Network Slice Extension with MEC UAVs Managed with Reinforcement Learning”. In: *IEEE Journal on Selected Areas in Communications* (2020), pp. 1–1.
- [54] I. F. Akyildiz, A. Kak, and S. Nie. “6G and Beyond: The Future of Wireless Communications Systems”. In: *IEEE Access* 8 (2020), pp. 133995–134030.
- [55] D. Nemirovsky et al. “A Machine Learning Approach for Performance Prediction and Scheduling on Heterogeneous CPUs”. In: *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 2017, pp. 121–128.
- [56] Y. Yan, B. Zhang, and J. Guo. “An Adaptive Decision Making Approach Based on Reinforcement Learning for Self-Managed Cloud Applications”. In: *2016 IEEE International Conference on Web Services (ICWS)*. 2016, pp. 720–723.
- [57] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2014-2015.
- [58] A. Lombardo, G. Morabito, and G. Schembra. “Modeling intramedia and intermedia relationships in multimedia network analysis through multiple timescale statistics”. In: *IEEE Transactions on Multimedia* 6.1 (2004), pp. 142–157.
- [59] C. Grasso et al. “Designing the Tactile Support Engine to Assist Time-Critical Applications at the Edge of a 5G Network”. In: *Submitted to Special Issue on Network Intelligence, Computer Communications* (2020).
- [60] G. Fettweis and S. Alamouti. “5G: Personal mobile internet beyond what cellular did to telephony”. In: *IEEE Communications Magazine* 52.2 (2014), pp. 140–145.
- [61] O. Holland et al. “The IEEE 1918.1 “Tactile Internet” Standards Working Group and its Standards”. In: *Proceedings of the IEEE* 107.2 (2019), pp. 256–279.
- [62] C. E. McPhail. “Respond, restore, resolve: Achieving 7-nines availability telecommunications systems in the field”. In: *Bell Labs Technical Journal* 11.3 (2006), pp. 173–189.

## REFERENCES

---

- [63] ITU-T. *The Tactile Internet. ITU-T Technology Watch Report*. [https://www.itu.int/dms\\_pub/itu-t/opb/gen/T-GEN-TWATCH-2014-1-PDF-E.pdf](https://www.itu.int/dms_pub/itu-t/opb/gen/T-GEN-TWATCH-2014-1-PDF-E.pdf). Accessed August 2020.
- [64] M. Simsek et al. “The 5G-Enabled Tactile Internet: Applications, requirements, and architecture”. In: *2016 IEEE Wireless Communications and Networking Conference*. 2016, pp. 1–6.
- [65] Adnan Aijaz et al. *Toward a Tactile Internet Reference Architecture: Vision and Progress of the IEEE P1918.1 Standard*. July 2018.
- [66] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [67] Stanford-University. *Hapkit*. <https://hapkit.stanford.edu/>. Accessed August 2020.
- [68] K. Zheng et al. “Heterogeneous Vehicular Networking: A Survey on Architecture, Challenges, and Solutions”. In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2377–2396.
- [69] C. Campolo et al. “5G Network Slicing for Vehicle-to-Everything Services”. In: *IEEE Wireless Communications* 24.6 (2017), pp. 38–45.
- [70] R. Yu et al. “Optimal Resource Sharing in 5G-Enabled Vehicular Networks: A Matrix Game Approach”. In: *IEEE Transactions on Vehicular Technology* 65.10 (2016), pp. 7844–7856.
- [71] X. Hou et al. “Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures”. In: *IEEE Transactions on Vehicular Technology* 65.6 (2016), pp. 3860–3873.
- [72] S. Raza et al. “A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions”. In: *Wirel. Commun. Mob. Comput.* 2019 (2019), 3159762:1–3159762:19.
- [73] L. Liu et al. “Vehicular Edge Computing and Networking: A Survey”. In: *arXiv: Signal Processing* (2019).
- [74] P. Mach and Z. Becvar. “Mobile Edge Computing: A Survey on Architecture and Computation Offloading”. In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1628–1656.
- [75] Y. Sun et al. “Learning-Based Task Offloading for Vehicular Cloud Computing Systems”. In: *2018 IEEE International Conference on Communications (ICC)* (2018), pp. 1–7.

## REFERENCES

---

- [76] F. Busacca et al. “Multi-Layer Offloading at the Edge for Vehicular Networks”. In: *18th IEEE - Mediterranean Communication and Computer Networking Conference (MedComNet 2020)* (2020).
- [77] M. Mozaffari et al. “Mobile Unmanned Aerial Vehicles (UAVs) for Energy-Efficient Internet of Things Communications”. In: *IEEE Transactions on Wireless Communications* 16.11 (2017), pp. 7574–7589.
- [78] Q. Liu, Z. Su, and Y. Hui. “Computation Offloading Scheme to Improve QoE in Vehicular Networks with Mobile Edge Computing”. In: *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*. 2018, pp. 1–5.
- [79] G. Faraci, C. Grasso, and G. Schembra. “Reinforcement-Learning for Management of a 5G Network Slice Extension with UAVs”. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*. 2019, pp. 732–737.
- [80] L. Galluccio et al. “Measuring QoS and QoE for a Softwarized Video Surveillance System in a 5G Network”. In: *2019 IEEE International Symposium on Measurements Networking (M N)*. 2019, pp. 1–6.
- [81] Triangle Consortium. *Deliverable D3.1 - Progress report on the testing framework Rel 1*. Tech. rep. Nov. 2016. URL: [https://www.triangle-project.eu/wp-content/uploads/2017/02/TRIANGLE\\_Deliverable\\_D3.1-FINAL.pdf](https://www.triangle-project.eu/wp-content/uploads/2017/02/TRIANGLE_Deliverable_D3.1-FINAL.pdf).
- [82] Quamotion. *Quamotion - Mobile App Test Automation*. <http://quamotion.mobi/Mwc/Index>. Accessed August 2020.
- [83] C. Grasso, R. Raftopoulos, and G. Schembra. “The Triangle Platform for End-to-End Performance Analysis of a 5G Video Transmission Network Slice”. In: *15th International Symposium on Wireless Communication Systems (ISWCS)*. 2018.
- [84] 5GINFIRE Consortium. *5GINFIRE Project*. <https://5ginfire.eu/>. Accessed August 2020.
- [85] vEyes. *Poseidon 2.0*. <http://www.veyes.it/poseidon-2-0/>. Accessed August 2020.
- [86] Flame Consortium. *Flame*. <https://www.ict-flame.eu/>. Accessed August 2020.



## REFERENCES

---

- [87] Flame Consortium. *Flame D3.1: FMI Vision, Use Cases and Scenarios*. Tech. rep. Mar. 2017. URL: <https://www.ict-flame.eu/wp-content/uploads/sites/3/2017/10/D3.1-FMI-Vision-Use-Cases-and-Scenarios-v1.1.pdf>.
- [88] Flame Consortium. *From the desktop to user trials*. <https://www.ict-flame.eu/blog-post/from-the-desktop-to-user-trials/>. Accessed August 2020.
- [89] K. Hansge, S. Robitzsch, and N. Stanchev and M. Boniface. *TOSCA templating in FLAME*. <https://gitlab.it-innovation.soton.ac.uk/FLAME/consortium/3rdparties/flame-tosca>. Accessed August 2020.
- [90] Hamid Falaki. *University of Bristol - 5GUK Test Network - FLAME Infrastructure Urban Hacking in 5G*. <https://www.ict-flame.eu/wp-content/uploads/sites/3/2019/11/Urban-Hacking-UoB-Infrastructure-compressed.pdf>. Accessed August 2020.
- [91] *FFMPEG*. <https://ffmpeg.org/>. Accessed August 2020.
- [92] Open Networking Foundation. *OpenFlow Switch Specification*. Tech. rep. Mar. 2015. URL: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.
- [93] Open Networking Foundation. *ONOS - Open Network Operating System*. <https://www.opennetworking.org/onos/>. Accessed August 2020.
- [94] Open Networking Foundation. *ONOS*. <https://wiki.onosproject.org/display/ONOS/ONOS>. Accessed August 2020.
- [95] OpenDaylight Project. *OpenDaylight*. <https://www.opendaylight.org/>. Accessed August 2020.

# Appendix

## Appendix A: OpenFlow

OpenFlow [92] arises from the need to establish a standard protocol for communication between control plane and data plane, guaranteeing the decentralization of network control at the SDN controller level and not at the single device level. The standard allows access to the forwarding plane of the devices in the infrastructure layer by controlling the flow entries. The controller, having an overview of the entire network topology, can modify the flow entries in order to guarantee a more sophisticated and updated packet forwarding management according to the network status.

The idea behind the OpenFlow protocol is to make the routing and forwarding tables inserted into routers and switches, totally programmable by external applications through an SDN controller. This involves the removal of all those control functions typical of these devices, such as discovery, path setup, and so on.

The concept of flow is introduced, which is a sequence of packets that, according to some policies, are labelled and treated in the same way. Policies can range from simpler ones (packets directed to the same recipient, packets from the same source, etc.) to more complex policies that can affect the application layer that generated the packets. This means that despite two streams (for example one that transmits video and one that transmits email) are generated by the same source, and are directed to the same recipient, the network can be configured in such a way as to make them follow two totally different paths.

As shown in Fig. A.1, an OpenFlow Switch is composed by a *Datapath* part, with one or more flow tables, a group table (to perform packet lookups and forwarding) and a meter table, and a *Control Channel* part, with one or more OpenFlow channels to external controllers. Each entry of the flow table contains a series of rules that allow identifying the packets, an action to be taken to forward the packets along the network, and statistics relating to the count of packets

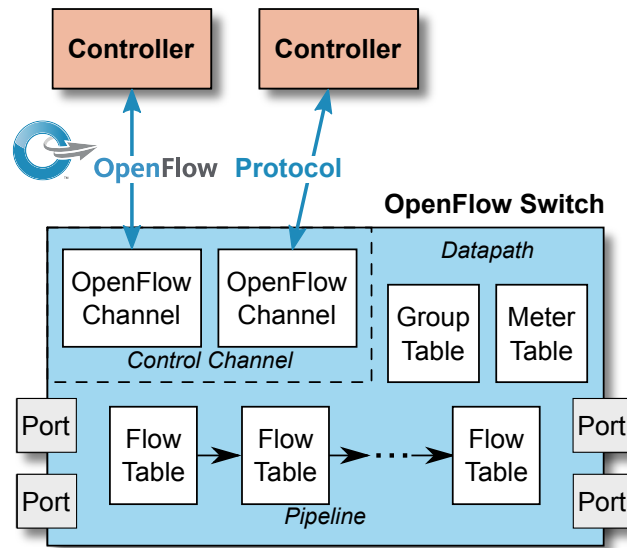


Figure A.1: Structure of an OpenFlow switch [92]

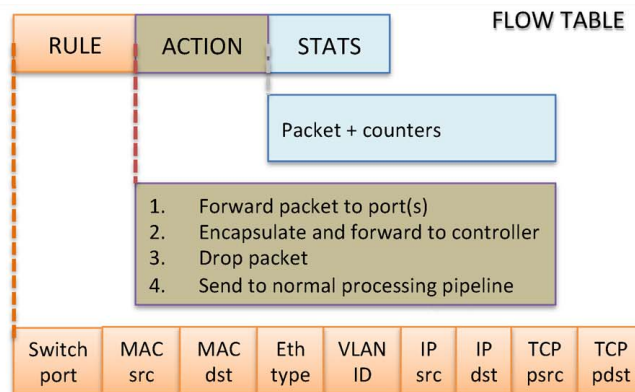


Figure A.2: Structure of the flow table [19]

corresponding to each rule (see Fig. A.2).

For each received packet, the Openflow switch can perform three possible actions:

- forward the packet to one or more ports according to the rules written in the flow table;
- deliver the packet to secure channel to send it to the Controller. This happens when the switch has not found any entry referring to this packet. Rather than discarding it, it is sent to the Controller who will decide whether or not to add a rule related to this packet (and therefore to all those of the same flow) in the flow table;
- eliminate the packet: thanks to programmability, it is possible to set policies

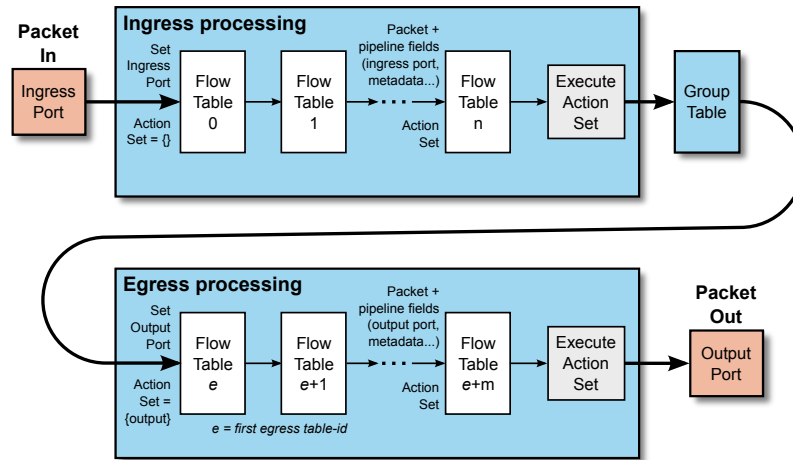


Figure A.3: Pipeline process inside a OpenFlow switch [92]

for managing security in the network, to avoid Denial of Service attacks, or simply to reduce traffic in the case of broadcast packets.

In Fig. A.3, the OpenFlow pipeline is shown [92]. It contains one or more flow tables and defines how ingress packets interact with those flow tables. Each OpenFlow switch has at least one ingress flow table, and can optionally have more flow tables. The flow tables are numbered in the order the packets have to traversed them, starting at 0. There are two stages in the pipeline process: ingress processing and egress processing. Pipeline processing starts with ingress processing at the first flow table 0, where flow entry that matches with the ingress packet is searched. If the outcome of ingress processing is to forward the packet to an output port, the OpenFlow switch may perform egress processing in the context of that output port. This is an optional procedure because some switch could not support egress tables. In this case, the packet is processed by the output port. If a valid egress table is configured as the first egress table, the egress processing start and a match between the packet and one flow entry of the egress table is looked for.

When a flow entry is found, the instruction set specified for that flow entry is executed on the packet. These instructions may explicitly direct the packet to a flow table (using the GotoTable Instruction) with a greater flow entry number. This is because the pipeline processing can only go forward and not backward.

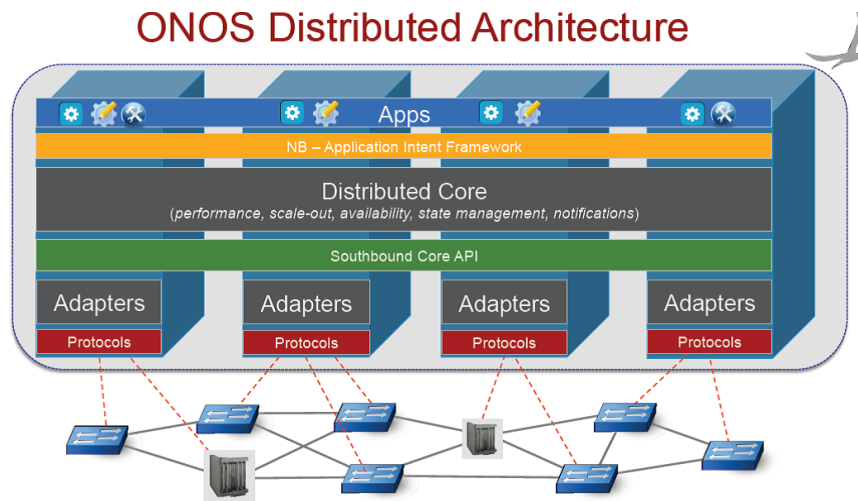


Figure A.4: ONOS architecture [93]

## Appendix B: SDN Controllers

Many implementations of SDN Controller are available, like NOX, POX, Floodlight and Ryu Controller. However, the most widely used today are ONOS and OpenDayLight.

The *ONOS* [93] controller is written in Java. It is the only SDN controller platform that supports the transition from legacy “brown field” networks to SDN “green field” networks. This enables exciting new capabilities, disruptive deployment and operational cost points for network operators. It is designed to allow the creation of highly scalable networks and is widely used both in Local Area Networks and in data center networks, which need to have great connectivity available. ONOS has a three-tier architecture (see Fig. A.4):

- Tier 1 contains modules related to protocols which communicate with the network devices (Southbound interface);
- Tier 2 is the core of ONOS and provides network state without relying on any particular protocol;
- Tier 3 contains applications, i.e. ONOS apps, which use network state information presented by Tier 2;

ONOS presents the following peculiarities [94]:

- *modularity and extensibility*: being the project comprised of a set of sub-projects, each with their source tree that can be built independently, it should be possible to introduce new functionalities as self-contained units.

ONOS allows users to easily customize, read, test and maintain network topologies;

- *scalability*: it is possible to modify the resources available according to the needs of the network. ONOS is designed specifically to horizontally scale for performance and geo-redundancy across small regions, being characterized by both Cluster Scalability and Architectural Scalability;
- *interfaces*: in the Southbound interface, it supports an extensive list of interfaces including OpenFlow, P4, NETCONF, TL1, SNMP, BGP, RESTCONF and PCEP; in the Northbound interface, ONOS offers a very large set of interfaces with gRPC and RESTful APIs. Also, ONOS presents an advanced GUI that is a single-page web-application, providing a visual interface to the ONOS controller (or cluster of controllers);
- *intent-based framework*: ONOS has the implementation of the inbuilt intent-based framework. By abstracting a network service into a set of criteria a flow should meet, the generation of the underlying OpenFlow (or P4) configuration is handled internally, with the client system specifying only what the functional outcome should be.

About resilience, ONOS provides high availability to carry out the most critical and crucial tasks within the network and this allows customers not to be faced with problems such as network downtime or failures of various kinds: one of the objectives of ONOS is have many mechanisms to secure the network and therefore have a high degree of reliability. Fault tolerance is achieved in the system with an odd number of SDN controllers. In the event of Master node failure, a new leader is selected to take control of the network. Moreover, the support of traditional and new-generation devices is guaranteed as, with ONOS, it is possible to add or configure both classic and modern devices within the same SDN network.

The other described SDN controller is *OpenDaylight* (ODL) [95]. In Fig. A.5, it is possible to see the architecture of ODL. As for ONOS, also in this case a three-layer architecture is shown:

- *Southbound Interface and Protocol Plugin* with APIs and plugin used to manage the components of the Data Plane Elements layer;
- *OpenDaylight Core*, composed by three blocks: OpenDaylight Platform, Platform Services (containing pluggable oriented services which perform specific networking tasks and other extensions to enhance the SDN functionality), and Network Services and Applications;

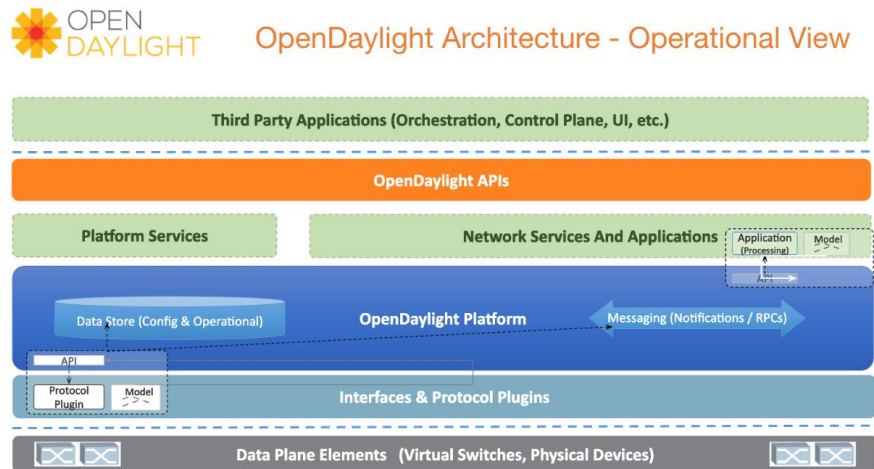


Figure A.5: ODL architecture [95]

- *Northbound Interface* with the APIs used to interact with Third Party Applications.

The characteristics of ODL are:

- *modularity*: built-in mechanisms provided by ODL simplify the connection of code modules to add extra functionalities inside the network;
- *scalability*: ODL uses a model-based approach, which implies a global in-memory view of the network required to perform logic calculations. ODL's latest release further advances the platform's scalability and robustness, with new capabilities supporting multi-site deployments for geographic reach;
- *interfaces*: as already said, it uses Southbound and Northbound interfaces to interact with other components. In particular the Southbound Interface supports an extensive list of protocols including OpenFlow, P4, NETCONF, SNMP, BGP, RESTCONF and PCEP. This means that ODL controller is able to manage more type of devices at the same time;
- *resilience and fault tolerance*: ODL fault tolerance mechanism is similar to ONOS, with an odd number of SDN controllers required to provide fault tolerance in the system. In the event of a master node failure, a new leader would be selected to take control of the network. The mechanism of choosing a leader is slightly different in these controllers: while ONOS focuses on eventually consistent, ODL focuses on high availability.