Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (38^{th}cycle)

Toward a Cloud Continuum for Efficient 6G Infrastructures

MUR DM 351

By

Jacopo Marino

******

Supervisor(s):
Prof. Fulvio Risso, Supervisor
Prof. Carla Fabiana Chiasserini, Co-Supervisor

Doctoral Examination Committee:
Prof. Giuseppe Di Modica, Referee, Università di Bologna
Prof. Lorenzo De Carli, Referee, University of Calgary
Dr. Christian Pinto, IBM Research Europe
Prof. Antonio Puliafito, Università di Messina
Prof. Guido Marchetto, Politecnico di Torino

Politecnico di Torino
2026

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Jacopo Marino
2026

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

This thesis is dedicated to all those who have been part of this journey.

To Garance, for the constant support and encouragement.

To my friends from Cuneo: Francesco B., Francesco V., Sara, Michele G., Miki, Francesca, Eva, Maicol, Simone, Lorenza, Michele S., Giorgia, Giacomo, and Eleonora.

To the friends made at Politecnico di Torino: Gioele, Dario, Silvia, Emma, Nelly, Riccardo, Stefano, and Laura.

To the international friends who brought me different perspectives of life: Felipe, Sebastián, Jean, Roberta, Ludovica, and Ryan.

To Daniele, for the collaboration and shared dedication to the research. To my supervisor, for the guidance and support throughout this work. To Guillem, for the guidance during the Valencian part of the journey.

To Valencia, for its warmth, its streets, and its sea, a city that made me feel at home far from home. To Torino, the city that welcomed me at eighteen and showed me how wide the world can be.

To my parents, for their unwavering support and for always believing in me, through every challenge and every achievement. To my grandparents, Teresa, Giuseppe, Marcella, and Silvano.

Ad maiora.

Acknowledgements

This research was conducted as part of Jacopo Marino Ph.D. programme, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative.

Abstract

Edge computing, 5G/6G networks, and containerization have created the conditions for a new paradigm: the edge-to-cloud continuum, in which heterogeneous infrastructures spanning far-edge devices, telco facilities, and cloud data centers are treated as a unified pool of resources. Despite this vision, deployments remain fragmented. Within each fragment, resources are managed through container orchestrators, with Kubernetes as the de facto standard; yet each cluster has an independent control plane, exposing only a partial view of available resources, and applications must be assigned to specific infrastructures at deployment, preventing transparent migration, dynamic load redistribution, and cross-domain sharing.

This thesis lays the foundations of a computing continuum across distributed cloud infrastructures. Part I builds the architectural foundations, while Part II validate them through distributed robotics. The parts share a goal but address distinct questions, and can largely be read independently.

On the infrastructure side, the thesis first examines the physical substrate available to telco operators. Drawing on publicly available telco data, it shows that operators already possess tens of thousands of central offices, telephone exchanges, and 5G cell sites suitable for hosting edge computing resources. A survey of existing multi-cluster technologies identifies Liqo as the most comprehensive production-ready foundation, while highlighting the open challenges that current solutions fail to address.

Building on this analysis, the thesis presents the Flexible, scaLable, secUre, and decentralIseD Operating System (FLUIDOS), which realizes the continuum through three transparency properties: deployment, communication, and resource availability. Central to FLUIDOS is the REsource Advertisement and Reservation (REAR) protocol, enabling standardized, efficient, and secure resource discovery and allocation across heterogeneous domains. REAR supports a rich data model cov-

ering heterogeneous resource types, with authentication grounded in decentralized identifiers and verifiable credentials. Complementing these foundations, the thesis proposes infrastructure-level data spaces that leverage multi-cluster capabilities to enable controlled cross-domain data sharing while preserving data sovereignty and regulatory compliance, with negligible overhead.

The continuum principles are then applied to distributed robotic systems, where latency, reliability, and energy efficiency constraints provide a demanding validation context. Three contributions are presented. The Scalable Middleware 2 (SM2) decouples local inter-process communication from network transport, preventing the performance degradation observed in DDS-based ROS2 configurations under adverse conditions, and reduces CPU usage by 3× and memory consumption by 4× compared to state-of-the-art alternatives. Four switching architectures for migrating stateless ROS2 computation between Kubernetes clusters are evaluated, showing that a Lifecycle Node-based approach achieves switching times at least 85% faster than redeployment-based alternatives. Finally, a Multiplexer component is proposed to switch between local and remote navigation streams based on message freshness and reliability, ensuring continuous and safe robot operation even under severe packet loss: at 55% packet loss, the approach achieves a 31% improvement in navigation performance over remote-only control, while a complementary Lifecycle Orchestrator reduces onboard CPU usage by up to 99% during periods of stable remote connectivity.

Taken together, these contributions establish a coherent architectural and experimental foundation for the computing continuum, spanning its physical motivations, enabling protocols, data governance mechanisms, and application to real-world robotic scenarios, demonstrating how cloud continuum architectures can serve as a concrete enabler for the efficient deployment of 6G network services.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 13, 80, 106

ALSG Application-Level Security Gateway. 92

API Application Programming Interface. 10, 20, 22–25, 29, 32–36, 52, 56, 65–67, 90, 92, 100, 104, 116, 137

AR Augmented Reality. 13

BMS Battery Management System. 79, 170

CMSG Control Message. 125

CNCF Cloud Native Computing Foundation. 35

CNI Container Network Interface. 26, 31, 37, 138, 139

CO Central Office. 9–14, 17–19

CPU Central Processing Unit. 45, 48, 51, 55, 60, 79, 97, 105, 106, 114, 115, 118, 120, 128, 129, 132, 138, 155, 168–170, 172–174

CRD Custom Resource Definition. 23, 29

CRDT Conflict-free Replicated Data Type. 34, 38, 104

CSV Comma Separated Values. 54

DDS Data Distribution Service. 114, 116–121, 126, 128, 131, 137–139, 145, 150, 173

DHT Distributed Hash Table. 72, 75, 106

DID Decentralised Identifier. 46, 57

DLT Distributed Ledger Technology. 47, 58, 74

DNS Domain Name System. 92

DSL Digital Subscriber Line. 13

ETSI European Telecommunications Standards Institute. 10

FLUIDOS Flexible, scaLable, secUre, and decentralIseD Operating System. 4–8, 39–44, 55, 57, 59, 64, 70–80, 93, 95, 105, 107, 110

FPGA Field-Programmable Gate Array. 111

FPS Frames Per Second. 164, 170

FQDN Fully-Qualified Domain Name. 139

FttH Fiber to the Home. 17

GDPR General Data Protection Regulation. 51, 73, 81, 82

GPU Graphics Processing Unit. 6, 51, 111, 134, 147, 163, 164, 170

gRPC gRPC Remote Procedure Call. 42

GSMA Global System for Mobile Communications Association. 10

HPC High-Performance Computing. 13

HTTP HyperText Transfer Protocol. 21, 42, 53, 74

IDL Interface Definition Language. 113

IdM dentity Management. 74

IDS International Data Spaces. 84, 85, 100, 101

IDS-RAM International Data Spaces Reference Architecture Model. 85

IDSA International Data Spaces Association. 7, 54, 84, 86, 100, 102, 106

IoT Internet of Things. 23, 44, 61, 64, 65

IP Internet Protocol. 26, 27, 31, 32, 37, 51, 56, 69, 93, 125, 139

IPC Inter-Process Communication. 116, 118–120, 122, 131, 134

IPCEI-CIS Important Project of Common European Interest - Next Generation Cloud Infrastructure and Services. 14

ISG Industry Specification Group. 10

JSON JavaScript Object Notation. 54

LAN Local Area Network. 72, 75, 78, 80

LaSC Listener and Switch Controller. 140, 141

LCN Lifecycle Node. 137, 140, 142, 143, 145, 146

LiDAR Light Detection and Ranging. 169

LO Lifecycle Orchestrator. 115, 148, 152, 154, 157, 158, 163, 168, 171, 172, 174

MCU MicroController Unit. 64

MEC Mobile Edge Computing. 10, 11, 17, 18, 111, 135, 147

ML Machine Learning. 110, 111

MMC Message Metrics Collector. 152, 155, 156, 162

MQTT Message Queuing Telemetry Transport. 21, 53, 74

MRSVP Mobile Resource reSerVation Protocol. 48

MUX Multiplexer. 114, 148, 151–156, 158, 159, 162, 163, 166–172, 174

MWC Mobile World Congress. 11, 14, 104

NAT Network Address Translation. 27, 93

NetPol Network Policy. 93, 94, 137, 138, 142, 145, 146, 174, 175

NFV Network Function Virtualisation. 10

NRA Nœud de Raccordement d'Abonnés (Subscriber Connection Node). 18

NRO Nœud de Raccordement Optique (Optical Connection Nodes). 17, 18

OLT Optical Line Terminal. 12

ONU Optical Network Unit. 12

OS Operating System. 52, 70

P2P Peer-to-Peer. 49, 57, 60, 65, 73

PoC Proof-of-Concept. 20

PON Passive Optical Network. 12

PoP Point of Presence. 9–11, 18

POSIX Portable Operating System Interface. 122

PSM Privacy and Security Manager. 57

QoE Quality of Experience. 72

QoS Quality of Service. 12, 48, 67, 113, 117, 118, 128

QUIC Quick UDP Internet Connections. 125, 131, 173

RAM Random Access Memory. 45, 48, 55, 97, 98, 106, 138, 170

REAR REsource Advertisement and Reservation. 5, 7, 8, 39, 44–47, 49–52, 54, 55, 57–59, 71–74, 77, 78, 80, 81, 92, 105, 107, 110

REST REpresentational State Transfer. 33

RMW ROS Middleware. 113, 126, 128, 132, 148, 173

RNAP Resource Negotiation and Pricing Protocol. 48

ROS Robot Operating System. 70, 112, 113, 116, 117, 122, 124, 134, 137, 138, 149

ROS2 Robot Operating System 2. 112–114, 116–122, 124, 126, 128, 131–142, 145, 146, 148–150, 161, 173–175

RPC Remote Procedure Call. 122

RSVP Resource reSerVation Protocol. 47

RSVP-TE Resource reSerVation Protocol-Traffic Engineering. 48

RTPS Real-Time Publish-Subscribe. 116, 117

RTT Round Trip Time. 10

RXP Resource and Service Exchange Point. 63, 67

SaaS Software-as-a-Service. 52

SC Switch Controller. 140, 141, 143

SLA Service Level Agreement. 48, 50

SM2 Scalable Middleware 2. 114, 118, 119, 121–126, 128–132, 134, 173, 174

SNAP Service Negotiation and Acquisition Protocol. 48

SNG SM2 Network Gateway. 119, 125, 131, 132, 173, 174

SoC System on a Chip. 147

SPOF Single Point of Failure. 124

SSH Secure Shell. 52

SSI Self-Sovereign Identity. 74

SwNP Switching with Network Policy. 137, 140, 142–145

SwR Switching with Redeployment. 136, 142–145

SwRNP Switching with Redeployment and Network Policies. 136, 142, 143, 145, 146

TCP Transmission Control Protocol. 171

TDM-PON Time Division Multiplexing Passive Optical Network. 12

TEE Trusted Execution Environment. 47, 73, 89, 93, 107

TSN Time-Sensitive Networking. 120

TSV Tab-Separated Values. 54

TTP Time To Purchase. 56

UDP User Datagram Protocol. 76, 138, 171

UK United Kingdom. 9, 14, 15, 104

VC Verifiable Credential. 47, 57, 74

VM Virtual Machine. 30, 39, 44, 52, 55, 72, 105, 135, 138, 150

VP Verifiable Presentation. 57, 58

VPN Virtual Private Network. 27, 31–33, 37

VR Virtual Reality. 13

W3C World Wide Web Consortium. 50

WAN Wide Area Network. 30, 63, 72, 75, 80

YOLO You Only Look Once. 170, 172

Part I

Cloud Continuum

Chapter 1

Introduction

Over the last decade, containerization has emerged as a dominant paradigm for packaging and deploying applications in a lightweight and interoperable manner, independently of the underlying infrastructure [1, 2]. This abstraction has enabled the rise of cloud-native computing, where applications are no longer conceived as monolithic services bound to single servers, but rather as collections of loosely coupled microservices managed at the scale of entire data centres. In this context, Kubernetes has established itself as the de facto open-source standard for container orchestration, effectively bridging semantic gaps across heterogeneous infrastructure providers and fostering portability and operational consistency[1].

In parallel, the emergence of edge and fog computing paradigms [3–5] has extended the cloud-native model beyond centralized data centres, pushing computation closer to data sources to address requirements such as low latency, bandwidth efficiency, data locality, and privacy. This evolution has given rise to the vision of an edge-to-cloud computing continuum, in which applications seamlessly span geographically distributed and technologically heterogeneous environments [6, 7].

Despite the availability of common orchestration interfaces and the apparent maturity of the computing continuum, current industrial and research practices still treat infrastructures as a collection of interconnected yet fundamentally isolated silos. Each Kubernetes cluster, whether deployed in the cloud, at the edge, or on-premises, remains governed by its own control plane, exposing only a partial and fragmented view of the globally available resources. This fragmentation is

[1] https://www.cncf.io/reports/cncf-annual-survey-2021/

not accidental, but rather the result of well-known limitations: centralized control planes suffer from scalability and stability issues when stretched across wide-area networks [8–10], while hybrid- and multi-cloud strategies, regulatory constraints, organizational boundaries, and physical isolation policies further incentivize cluster proliferation and separation.

As a consequence, today's computing continuum remains largely static. Applications must be explicitly assigned to specific infrastructures at deployment time, preventing transparent workload migration, dynamic load redistribution, or resource compensation across domains [11–13]. Even when applications are architected as multi-tier systems, e.g., data acquisition at the far edge, aggregation at the telco edge, and analytics in the cloud, their deployment logic remains tightly coupled to infrastructure details. Developers and operators must reason about where each component runs, how services are exposed across cluster boundaries, and how resources are provisioned and scaled in each domain. This lack of abstraction significantly limits the dynamism, resilience, and usability of the computing continuum.

This thesis builds upon the concept of liquid computing [14], which advocates moving beyond rigid cluster boundaries toward a borderless computing environment. In this vision, heterogeneous infrastructures, spanning cloud data centres, edge clusters, on-premises resources, and potentially even individual devices, are abstracted into a single virtual pool of resources, often referred to as a big cluster. Within this virtual space, applications are no longer constrained to predefined silos, but can dynamically flow across the continuum, selecting execution locations based on their requirements (e.g., latency, computational power, data locality, or regulatory constraints) and the current state of the infrastructure.

At the core of this work there is the Flexible, scaLable, secUre, and decentralIseD Operating System (FLUIDOS)[2], a European research initiative that introduces a decentralized meta-operating system to enable dynamic and flexible resource management across heterogeneous computing environments. FLUIDOS instantiates the liquid computing vision by leveraging Kubernetes as its technological substrate, while extending it with decentralized, multi-cluster, and multi-domain abstractions that preserve compatibility with existing cloud-native tools and workflows. Rather than merely interconnecting independent clusters, FLUIDOS seeks to redefine the

[2]https://fluidos.eu

computing continuum around a set of fundamental transparency properties that current approaches fail to provide.

Beyond defining the computing continuum, FLUIDOS provides concrete mechanisms to realize it in practice. In particular, resource and service reflection ensure that control-plane information is propagated across peered domains, enabling the distributed execution of workloads while preserving local autonomy. In addition, the storage and data continuum extends this model to stateful applications, addressing data locality, data gravity, and regulatory compliance through proximity-aware and sovereignty-preserving data management [15, 16].

Complementing these architectural foundations, this thesis also explores the REsource Advertisement and Reservation (REAR) protocol, designed to enable standardized, efficient, and secure resource discovery and allocation across the computing continuum. REAR allows continuum participants to advertise and reserve heterogeneous resources, ranging from compute and storage to sensors, actuators, and data, thus supporting dynamic participation, decentralized governance, and fine-grained resource negotiation across heterogeneous domains.

1.1 Industry landscape and motivations

While the computing continuum promises seamless integration of heterogeneous resources across edge and cloud environments, many real-world applications remain constrained by traditional siloed computing models. These models often fail to satisfy the increasingly diverse and stringent requirements of modern workloads, particularly when computation, data, and users are geographically distributed.

Real-time applications require immediate data processing to support decision-making within a few milliseconds [17]. To meet such latency constraints, these applications must operate at the far edge, close to where data is generated. At the same time, they require high availability: if a hosting cluster becomes unreachable, a new healthy instance must be rapidly deployed on a nearby site. Current siloed infrastructures, however, lack mechanisms for transparent service relocation and real-time cross-site orchestration.

Data-intensive applications rely on the processing, filtering, and storage of large volumes of data [18]. In these scenarios, access to specialized hardware accelerators,

such as Graphics Processing Units (GPUs), is essential for advanced analytics, statistical processing, and machine learning inference. In practice, these resources are unevenly distributed across cloud, on-premises, and edge environments, and static multi-cloud deployments prevent applications from dynamically exploiting available accelerators across domains.

Cross-domain data-sensitive applications handle sensitive data that must comply with strict privacy, security, and regulatory requirements. These applications often combine localized processing with broader data sharing and analytics workflows, creating tension between data mobility and regulatory compliance. This challenge is closely related to the concept of *data spaces* [15, 16], which aims to enable federated ecosystems for trusted data sharing while preserving data sovereignty. In current deployments, however, data governance constraints frequently reinforce infrastructure silos, limiting cross-domain workload execution.

Highly mobile and decentralized applications are characterized by the mobility of users and devices, and consequently by the need for applications to follow users as they move across locations [19]. Similar to handover procedures in telecommunications networks, these applications require seamless migration of services across edge and metro nodes. Existing edge-to-cloud infrastructures typically lack automation for mobility-aware orchestration, forcing manual or application-specific solutions.

Within the FLUIDOS project, addressing these limitations is a central objective. By enabling decentralized orchestration, transparent resource sharing, and dynamic workload placement across heterogeneous domains, FLUIDOS aims to bridge the gap between industry-driven requirements and the execution capabilities offered by a truly liquid computing continuum.

1.2 Summary of contributions

Part I of this dissertation is structured into five main chapters, each addressing a research area within the computing continuum. Specifically, the contributions of each section can be summarised as follows:

- **Chapter 2** examines the edge-to-cloud continuum from two complementary perspectives: the standardisation and deployment challenges identified by the research community and standards bodies, and the physical infrastructure

that telco operators can offer as the substrate for this continuum. Drawing on concrete data from telco operators, the chapter quantifies the scale and operational complexity of distributed edge infrastructure, arguing that the sheer number of potential edge sites makes the cloud continuum not merely desirable but a practical necessity.

- **Chapter 3** surveys production-ready and research-oriented multi-cluster orchestration platforms. It analyses KubeEdge, Karmada, and Liqo as the most relevant open-source candidates, compares their capabilities and limitations, and identifies Liqo as the most comprehensive foundation for realising the cloud continuum. Research-oriented contributions are also discussed, highlighting design principles that informed the subsequent architectural choices.

- **Chapter 4** presents the FLUIDOS approach to building a true computing continuum, introducing three transparency properties (deployment, communication, and resource availability) and the architectural mechanisms to realise them. Central to this chapter is the REAR protocol, which enables standardised, efficient, and secure resource advertisement and reservation across heterogeneous domains, supported by a rich data model and authentication grounded in decentralised identifiers and verifiable credentials.

- **Chapter 5** addresses the challenge of controlled cross-domain data sharing in multi-domain continuum environments. The chapter proposes infrastructure-level data spaces that leverage Liqo's multi-cluster capabilities to allow data consumers to deploy processing logic directly within a data producer's cluster. The approach enforces data sovereignty and regulatory compliance with negligible overhead, and is shown to integrate with application-level data space standards such as the International Data Spaces Association (IDSA) architecture and the Gaia-X ecosystem.

- **Chapter 6** summarises the contributions of Part I, discussing how the presented work collectively addresses the fragmentation of modern computing infrastructures. It reflects on the open challenges that remain, including resource discovery at scale, interoperability with non-Kubernetes environments, and runtime enforcement of data sovereignty, motivating the transition to Part II.

It is worth noting that, while Part I and Part II are both motivated by the overarching goal of realizing a cloud continuum, the relationship between them is one of shared vision rather than direct technical dependency. Part II does not exercise the REAR protocol, the data-space architecture, or the FLUIDOS control plane developed in Part I. Rather, it investigates a complementary and equally demanding question, namely, how continuum-style workload mobility and resource transparency must be re-engineered when the workload itself (distributed robotics) imposes strict latency, reliability, and energy constraints that the general-purpose mechanisms of Part I do not explicitly target. The two parts can therefore be read as two threads of the same research programme, connected by a common problem statement and design philosophy, but addressing distinct layers of the continuum stack.

1.3 Previously published works

This part includes previously published and co-authored works. In particular, Chapter 3 and Chapter 4 are adapted from the work presented in [20–22] and in the public deliverable of FLUIDOS [23]. Chapter 5 is adapted from [16].

Chapter 2

6G and Cloud Continuum

The convergence of next-generation mobile networks and distributed computing paradigms is reshaping the architectural foundations of telecommunications. As the industry advances toward 6G, the demand for ultra-low latency, data sovereignty, and intelligent service placement is driving the evolution of the so-called edge-to-cloud continuum, a model in which computational resources are seamlessly distributed across a hierarchy of locations spanning from user premises to centralised cloud data centres. This vision, however, cannot be realised in the abstract: it depends on the availability of physical infrastructure capable of hosting compute resources at scale, distributed across national territories and close to end users. Telco operators, with their thousands of Central Offices (COs), telephone exchanges, and cell sites, possess precisely this kind of infrastructure, and the question of how to leverage it for edge computing is now at the centre of both academic research and industrial strategy.

This chapter examines the edge-to-cloud continuum from two complementary perspectives. It first reviews the standardisation and the deployment challenges that the research community and standards bodies have identified as fundamental to distributed edge computing, with particular attention to the requirements introduced by 6G. It then turns to the physical infrastructure that telco operators can offer as the substrate for this continuum, drawing on concrete data from the United Kingdom (UK) and France to quantify the scale of the opportunity and, critically, the operational complexity that this scale entails. The chapter argues that the sheer number of potential edge sites, ranging from a few hundred national Points of

Presence (PoPs) to tens of thousands of telephone exchanges, is what makes the cloud continuum not merely desirable but necessary as an architectural paradigm.

Managing such distributed edge infrastructure is a complex, multi-layered challenge that has been central to the Mobile Edge Computing (MEC) vision since its earliest formulation. The European Telecommunications Standards Institute (ETSI) Introductory White Paper [24] establishes that MEC introduces multiple distinct management layers covering the hosting infrastructure, the application platform, and the software applications running on top of it, and that because virtualisation and cloud technologies enable flexible deployment scenarios, different organisations may find themselves responsible for management at each of these levels. This need becomes more pronounced at scale: ETSI White Paper #36 [25] confirms that interactions between the edge cloud middleware, the applications, and the underlying networks all require coordination, and that standards are specifically needed to enable common infrastructure capabilities, smart application placement, and service continuity across distributed nodes. Beyond single-operator environments, the Global System for Mobile Communications Association (GSMA) is working to federate multiple operators' edge computing infrastructures, enabling application providers to deploy and manage their software across multiple domains through common Application Programming Interfaces (APIs), with ETSI Industry Specification Group (ISG) MEC directly addressing those federation requirements in support of multi-operator, multi-network and multi-vendor environments.

How individual operators translate these standardisation efforts into concrete deployment strategies varies considerably, and the choices they make about where to place compute resources within their network hierarchy have profound implications for the scale and nature of the operational challenge. TIM's[1] network, for example, is organised in a three-tier hierarchical architecture comprising Local Aggregators at COs, Feeders at regional sites, and Metro PoPs at national sites [26]. A subset of national PoPs already hosts Telco Cloud islands based on Network Function Virtualisation (NFV) Infrastructure, designed to run virtualised and cloud-native network functions. Notably, the operator observes that redistributing functionalities from national to regional sites yields only a marginal Round Trip Time (RTT) improvement of approximately 1ms, which is insufficient to justify deeper edge deployments for current applications. However, the need of ultra-low latency use

[1] https://www.tim.it

cases such as extended reality, autonomous driving, industrial automation, and eHealth, along with data sovereignty requirements, are identified as the key drivers that will push computation toward far-edge locations. TIM's target architecture therefore envisions a continuum of edge nodes spanning from national peering points down to on-premises micro-edge devices, with inter-operator federation as a critical enabler [27].

Telefónica[2] similarly observes that the boundaries between cloud and telecommunications networks are beginning to blur, as new generations of services demand ever-lower latency and greater proximity to the end user. MEC enables the placement of cloud resources within a telecoms operator's network, deploying computing and storage capabilities closer to the customer to allow real-time processing, guaranteed bandwidth, and increased privacy and security. In this evolving landscape, edge computing, as a natural evolution of the network, opens the door to collaboration with traditional cloud providers to create true hybrid cloud solutions and services [28].

A fundamentally different approach emerges when operators choose to distribute compute not across a limited number of national PoPs, but across the much larger set of COs and telephone exchanges that constitute the access network. FiberCop[3], in collaboration with Microsoft, Dell Technologies, and 6WIND, has completed a field trial of an Edge Cloud solution for telco applications, presented at Mobile World Congress (MWC) 2026 [29]. At the core of the trial is a virtualised Broadband Network Gateway by 6WIND, integrated within FiberCop's access network on an Edge Cloud node running on Microsoft Azure Local with Dell infrastructure. The solution employs Selective Local Breakout technology, which routes only high-performance-demanding traffic locally while maintaining centralised management for everything else. Tests demonstrated latencies typically 1-5ms, enabling real-time control of robotic systems, distributed AI applications, automated logistics, Digital Twins, and mission-critical industrial applications. Crucially, the trial showed that the architecture can be deployed even in small-sized COs, and with its 10,500 COs distributed across the Italian territory potentially suitable for hosting Edge Cloud nodes, FiberCop positions itself as an operator capable of transforming its fibre network into an enabling platform for the country's digitalisation. The contrast between concentrating compute in a few tens of national PoPs and distributing it across thousands of exchange sites illustrates that there is no single established approach

[2]https://www.telefonica.com/en/
[3]https://www.fibercop.com/en/

to edge placement, and that the choice of deployment granularity determines the magnitude of the orchestration and management challenge that the cloud continuum must address.

The push toward distributing computational resources closer to end users has also reached the optical access network layer, where several architectural approaches have been explored for embedding compute within the Passive Optical Network (PON) infrastructure itself. Ahmad et al. [30] observe that the growing demand for low-latency applications such as augmented and virtual reality has triggered increasing integration of fog computing with PONs. In the literature, candidate locations for fog facilities include the CO, the Remote Node, the Optical Network Unit (ONU) premises, or the PON equipment itself. Their work focuses on this last approach, where ONUs are equipped with additional computing and storage units capable of serving both local users and foreign task requests from other ONUs in the same PON system. Results on a 32-ONU system show that the approach effectively balances energy efficiency with Quality of Service (QoS) requirements.

Newaz et al. [31] propose the fog co-located Optical Line Terminal (OLT) architecture, where fog servers are placed alongside the OLT at the CO of a Time Division Multiplexing Passive Optical Network (TDM-PON) system. The architecture introduces a resource manager entity at the CO, responsible for managing computational and storage resources across fog servers and micro servers in the access segment. Simulation results show that the fog co-located OLT architecture significantly reduces upload response time and improves throughput compared to conventional cloud service architectures where all processing occurs at remote data centres, confirming the benefit of embedding fog computing capabilities within the telco CO infrastructure.

GENIO [32] is a platform that integrates edge computing within existing PON infrastructures deployed by telecom operators. It enhances telco COs with computational and storage resources, enabling operators to leverage their existing PONs as a distributed edge computing infrastructure. The architecture extends two key PON elements: OLTs, natively present in COs, are evolved into servers with computational and storage capabilities, while ONUs at user premises are extended with smart embedded systems. This results in a three-layer architecture spanning far-edge at user sites, edge at COs, and cloud, with the cloud layer providing centralised orchestration of workers deployed at the edge or far-edge layers. These works

collectively reinforce the CO as the most natural and recurrent location for edge compute placement within optical access networks, a pattern that acquires particular significance when considered alongside the sheer number of such facilities available in national telco infrastructures.

The trend toward repurposing legacy telco facilities for edge computing and colocation is already visible at industrial scale. In the United States, as demand for copper and Digital Subscriber Line (DSL) services has declined with the rise of fibre and fixed wireless broadband, thousands of COs have become largely underutilised, with some facilities originally built for over 100,000 subscribers now serving only a few thousand. Operators such as AT&T[4], Lumen Technologies[5], Frontier Communications[6], and Ziply Fiber[7] have begun repurposing these buildings as edge colocation centres, leveraging their existing power, backup power, cooling, and security infrastructure, as well as their proximity to densely populated areas [33]. The emergence of Artificial Intelligence (AI) workloads is changing this economic calculation: while model training remains predominantly centralized in large cloud and High-Performance Computing (HPC) facilities, inference is increasingly being pushed toward edge locations to meet the response-time requirements of applications such as video analytics, Augmented Reality (AR), and Virtual Reality (VR). This shift toward distributed, edge-hosted inference, separate from where training occurs, further favours repurposing over selling, as low-latency processing close to end users becomes increasingly critical [34, 35]. Beyond cloud gaming, mainstream applications increasingly sensitive to end-to-end latency include augmented and virtual reality (where motion-to-photon delays above $\sim 20ms$ cause perceptible lag and motion sickness), real-time video conferencing and live event streaming, connected and autonomous vehicle coordination (e.g., cooperative perception and platooning), industrial closed-loop control in smart manufacturing, and, increasingly, interactive AI assistants relying on edge or near-edge inference to bound response time. Ziply Fiber provides a concrete example: formed through the acquisition of Frontier's operations in 2020, the company has repurposed more than 200 of its old CO buildings into colocation data centres across Washington, Oregon, Idaho, and Montana [36]. These hardened telco facilities already feature redundant power systems, backup generators, cooling infrastructure, and physical security, making them well suited

[4]https://www.att.com
[5]https://www.lumen.com/en-us/home.html
[6]https://www.frontier.com
[7]https://ziplyfiber.com

for hosting enterprise compute infrastructure. Lumen has similarly repurposed some of its COs for enterprise colocation, confirming a broader industry trend in which telco operators leverage their geographically distributed legacy infrastructure as a foundation for edge and colocation services within the cloud continuum.

This convergence toward distributed edge deployment is now also taking shape at the multi-operator level in Europe. At MWC 2026, Deutsche Telekom[8], Orange, Telefónica, TIM, and Vodafone[9] announced the first live demonstration of the European Edge Continuum, a federated edge infrastructure spanning the five largest operators in Europe [37]. The five operators have successfully federated their edge environments, enabling seamless deployment of applications across their combined European footprint, with the federation now operational in lab and pre-production environments. The initiative was enhanced by platform components developed in the context of the Important Project of Common European Interest - Next Generation Cloud Infrastructure and Services (IPCEI-CIS), funded by the European Union through Next Generation EU. The European Edge Continuum enables customers and developers to deploy applications automatically and securely across nodes from different operators through a single-entry point, laying the foundation for dynamic workload allocation, intelligent distribution of applications across federated nodes, and service continuity as users move across networks [38]. To appreciate the operational implications of such federated ambitions, it is necessary to examine the actual scale and geographic distribution of telco facilities in concrete national contexts.

2.1 Telco infrastructure at scale

The UK and France were selected as case studies primarily for data-availability reasons: both countries have telecommunications regulators (Ofcom and ARCEP, respectively) that publish detailed, periodically updated infrastructure data (including exchange counts, cell-site sharing statistics, and 5G rollout obligations) at a level of granularity rarely available elsewhere in Europe. Both also host large incumbent operators (BT and the four major French operators) with multi-decade-old fixed and mobile networks, making them representative of the dense, legacy-rich telco footprint found across much of Western Europe, rather than greenfield deployments. The

[8] https://www.telekom.com/en
[9] https://www.vodafone.com

two countries additionally differ in regulatory approach (privatised single-incumbent in the UK vs. ARCEP-mandated multi-operator site-sharing in France), which strengthens the generality of the argument: the conclusion that a large, underutilised physical site base exists for edge co-location holds under two different regulatory and market structures, not just one.

2.1.1 BT's network topology in UK

BT Group[10], recognised as the world's oldest communications company, serves more than 30 million customers in the UK and internationally. While its network is the largest and covers the entire country, it is economically impractical for competing operators to replicate such scale. These operators often deploy limited backbones, commonly a fibre network connecting major cities such as London, Bristol, Birmingham, Manchester, and Leeds, while depending on external connectivity for off-network areas.

BT's core network includes around 5,500 telephone exchanges and approximately 350 regional and trunk sites. In contrast, smaller operators typically operate between 5 and 40 exchanges and a few hundred points of presence, often focused on major cities. The access network connects end users to the nearest core network entry point, usually a local telephone exchange [39]. This extensive geographic distribution of exchange buildings, each equipped with power, cooling, and fibre connectivity, represents a significant asset that could serve as a foundation for edge computing deployment across the national territory. At the same time, 5,500 exchanges represent a fundamentally different management challenge than 350 regional sites: the former implies a distributed computing fabric of potentially tens of thousands of servers spread across thousands of locations, while the latter corresponds to a more traditional, albeit geographically distributed, data centre model.

[10]https://www.bt.com

2.1.2 ARCEP data on mobile and fixed infrastructure in France

Mobile infrastructure as the primary candidate layer for edge nodes

The French telecommunications landscape provides detailed quantitative evidence of the infrastructure available for edge computing deployment. As of 31 December 2024, there were 61,506 mobile cell sites (masts, rooftops, and towers) in Metropolitan France [40]. Each of these sites already possesses the fundamental prerequisites for compute hosting: a physical structure, power supply, and backhaul connectivity.

Of particular significance for multi-operator edge deployments, 30,535 of these sites, close to 49.6%, were already being shared between multiple operators as of the same date, and 9,741 were shared simultaneously by all four national operators (Bouygues Telecom[11], Free Mobile[12], Orange[13], and SFR[14]). Sites shared by all four operators are especially attractive for neutral-host edge computing models, where compute infrastructure can serve multiple tenants from a single physical location, reducing both capital expenditure and the environmental footprint of deployment.

5G sites as the priority tier for ultra-low-latency edge computing

Within the broader pool of mobile cell sites, the subset of 5G-enabled sites represents the highest-priority tier for edge server deployment, given the sub-10ms latency requirements that 5G use cases such as Industry 4.0, autonomous vehicles, and augmented reality demand. As of 31 December 2024, the four operators had activated a combined total of more than 60,000 5G sites, with each operator individually operating between 11,805 and 20,594 sites. The per-operator breakdown is reported in Table 2.1.

The 3.5GHz band is particularly relevant, as it is the core 5G band attributed in France in November 2020, offering the best balance between coverage, capacity, and latency for edge computing workloads. Operators are bound by regulatory obligations to reach 8,000 sites in the 3.5 GHz band by end of 2024 and 10,500 by end of 2025, with the requirement that all sites provide a 5G-type service by 2030. This mandated rollout trajectory provides a reliable and legally binding forecast of

[11] https://www.bouyguestelecom.fr
[12] https://mobile.free.fr
[13] https://www.orange.fr
[14] https://www.sfr.fr

Table 2.1 French Operators Total 5G Sites.

Operator	**Total 5G sites**	**Of which 3.5 GHz**
Bouygues Telecom	13,998	8,700
Free Mobile	20,594	8,061
Orange	11,805	11,388
SFR	13,966	9,401

the physical infrastructure base that will be available for edge computing co-location in the coming years.

Fixed network infrastructure and NRO-based CO edge locations

On the fixed network side, the fibre rollout provides a complementary layer of candidate locations for edge servers, particularly at the CO tier, which the industry broadly recognises as the first and most economically viable level of MEC deployment. At the end of 2024, 91% of premises (including both residential and business locations) in Metropolitan France were eligible for Fiber to the Home (FttH), with 4 million premises still to be upgraded. This near-complete deployment means that the Nœud de Raccordement Optique (Optical Connection Nodes) (NRO) infrastructure, the optical connection nodes that serve as the active headend of FttH networks, is now present across virtually the entire national territory. NROs are, by definition, locations where optical fibre converges, where active equipment is housed, and where operators already manage and maintain physical infrastructure: precisely the conditions required for edge server co-location.

The obligation of network completeness imposed by ARCEP[15] on infrastructure operators ensures that these NROs must cover all premises within their deployment zone, giving the NRO layer a universal geographic reach that few other infrastructure layers can match. Furthermore, under the New Deal Mobile targeted coverage scheme, 3,481 additional cell sites were put into service by end of 2024, further extending the geographic reach of the access-edge layer into rural and low-density areas that were previously underserved, zones where edge computing could play a critical role in enabling latency-sensitive services without dependence on distant centralised cloud infrastructure.

[15]https://en.arcep.fr

The approximately 15,000 Orange PoPs (Nœuds de Raccordement d'Abonnés (Subscriber Connection Nodes) (NRAs) and NROs) represent the most strategically significant tier for initial MEC server deployment, as they already combine the necessary physical prerequisites, namely fibre backhaul, power, space, and national geographic distribution, with proximity to both fixed and mobile subscribers.

2.2 The operational challenge and the need for the Cloud Continuum

The evidence assembled throughout this chapter converges on a central tension: telco operators possess a vast and geographically distributed physical infrastructure that is, in principle, ideally suited to host edge computing resources, but the operational reality of deploying and maintaining compute servers across thousands of distributed sites presents challenges that are qualitatively different from those of centralised cloud computing. Unlike cloud data centres, which benefit from economies of scale in staffing, standardised hardware environments, and streamlined logistics, telco COs and cell sites were designed primarily for telecommunications equipment, not for general-purpose infrastructure.

The colocation of edge servers in telco exchanges introduces significant complexities across several dimensions. The maintenance of distributed servers across hundreds or thousands of sites requires either a large and geographically dispersed technical workforce or advanced remote management capabilities that can handle hardware failures, software patching, and security updates without on-site intervention. The heterogeneity of legacy facilities, which vary considerably in terms of available space, power capacity, cooling adequacy, and physical access constraints, makes it difficult to deploy standardised compute configurations at scale. The lifecycle management of edge servers, including firmware updates, operating system upgrades, and application redeployments, becomes exponentially more complex as the number of sites grows, particularly when each site may host workloads with different latency, reliability, and security requirements.

These challenges scale directly with the deployment granularity that the operator chooses. An approach that concentrates compute in a few tens of national PoPs, as TIM's current telco cloud deployment illustrates [26], keeps the management

perimeter comparable to that of a traditional distributed data centre architecture. An approach that pushes compute into thousands of COs, as FiberCop's 10,500-site vision and BT's 5,500-exchange network suggest, or into tens of thousands of cell sites, as the ARCEP data for France demonstrates, transforms the problem into one of managing a massively distributed computing fabric where the ratio of sites to available operations staff makes manual intervention at each location impractical. When this is further multiplied by multi-operator federation, as the European Edge Continuum initiative envisions across five major operators, the combined infrastructure could encompass tens of thousands of heterogeneous locations, each requiring provisioning, monitoring, updating, and securing.

It is precisely this scaling challenge that motivates the development of the edge-to-cloud continuum as an architectural paradigm. Rather than treating edge sites as isolated, independently managed islands of compute, the continuum model envisions a unified management and orchestration framework that spans from far-edge devices through CO edge nodes to centralised cloud data centres. By abstracting the underlying infrastructure heterogeneity and providing automated lifecycle management, intelligent workload placement, and federated governance mechanisms, the edge-to-cloud continuum aims to make the operational complexity of distributed edge deployment tractable at commercial scale. This is the enabling condition for telco operators to transform their vast physical footprint into a viable distributed computing platform, and it is the architectural foundation upon which the integration of 6G services with edge computing capabilities will ultimately depend.

Chapter 3

Technologies for the Cloud Continuum

The idea of providing unified computing resources within a single continuum has been explored for several years, with numerous proposals emerging from both the scientific community and enterprise-driven open-source initiatives. The underlying objective shared by these efforts is to offer a unified view of multiple distributed clusters, either operated by a single administrative entity or deployed within a specific geographic area. Existing scientific work addresses this problem from different perspectives, ranging from full-fledged systems to Proof-of-Concept (PoC) implementations. These proposals vary in their design choices, with some closely adhering to the de facto standard container orchestrator, Kubernetes, while others introduce custom abstractions and dedicated APIs[1].

[1]This work was conducted in collaboration with Fulvio Risso, Marco Iorio, Stefano Galantino, Fulvio Valenza, Riccardo Sisto, Alessandro Cannarella, Vlad Coroama, Nasir Asadov, Stefano Braghin, Liubov Nedoshivina, Ambrish Rawat, Mohamed Suliman, Andy Edmonds, Antonio Fernando Skármeta Gómez, Alejandro M. Zarca, José Manuel Bernabé, José Francisco Pérez, Domenico Siracusa, Emna Amry, George Kornaros, Marcello Coppola, Elisa Albanese, Roberta Terruggia, Eduard Marin, Silvio Cretti, Amjad Yousef Majid, and Margherita Facca. It was supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative.

3.1 Production-ready technologies

This chapter presents three of the most promising open-source projects that are well suited as foundational building blocks for realizing the cloud continuum vision. In addition, more experimental solutions are discussed later in the chapter. Although such projects may not represent optimal candidates for directly underpinning a cloud continuum infrastructure, they nonetheless offer valuable insights and design ideas that could be integrated or further developed. Additional related technologies are also introduced where relevant.

Finally, a comparative analysis is provided to highlight the respective advantages and limitations of the presented solutions, enabling the reader to better understand their trade-offs in the context of cloud continuum infrastructures.

3.1.1 KubeEdge

KubeEdge[2] is an open-source framework that extends Kubernetes to support the orchestration of containerized applications on edge devices. It provides infrastructure-level support for networking, application deployment, and metadata synchronization between cloud and edge environments. In practice, KubeEdge enables developers to implement applications using standard communication protocols such as HyperText Transfer Protocol (HTTP) or Message Queuing Telemetry Transport (MQTT), package them as containers, and deploy them either at the edge or in the cloud according to application requirements.

Figure 3.1 illustrates the KubeEdge architecture, the cloud side acts as the master cluster. This cluster can be deployed either on-premises or on managed cloud platforms provided by vendors such as Amazon or Google, and is responsible for establishing remote connections with edge devices as well as scheduling and deploying pods on them. Communication between cloud and edge relies on a set of core components deployed on both sides, among which CloudCore and EdgeCore play a central role.

- **CloudCore**: This component runs on the cloud side and is responsible for managing the overall KubeEdge system. It acts as an integration layer between

[2]https://kubeedge.io

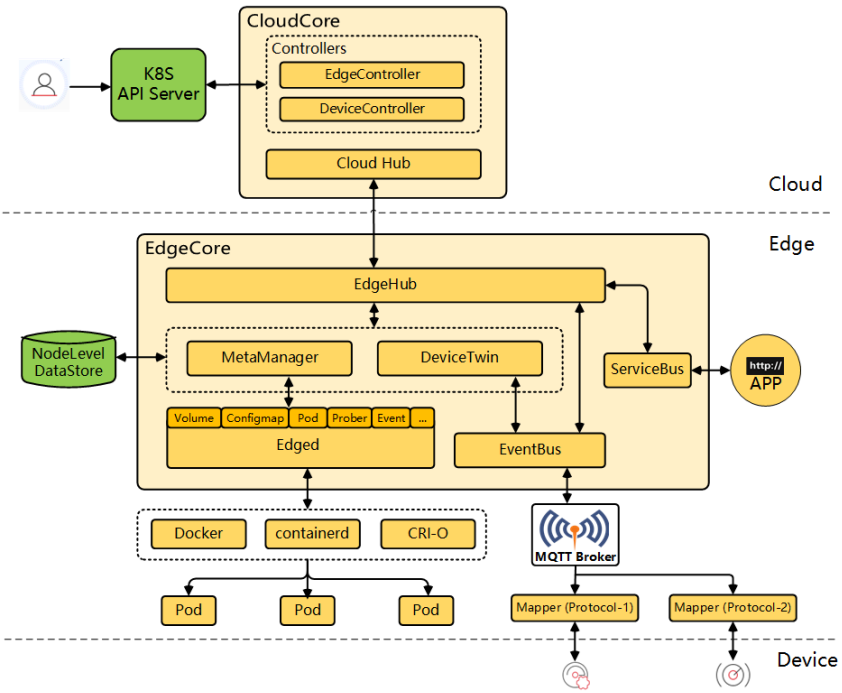


Fig. 3.1 KubeEdge architecture

the Kubernetes API server and the KubeEdge edge components. CloudCore communicates with edge nodes to coordinate the deployment, management, and monitoring of containerized workloads and connected devices.

- **EdgeCore**: This component runs on the edge device and manages the lifecycle and execution of containerized applications locally. It communicates with the cloud side to receive deployment instructions and to report application status. In addition, EdgeCore handles the detection and registration of devices and sensors connected to the edge node, exposing their data and functionality to applications running at the edge.

To integrate edge devices as nodes of a Kubernetes cluster, CloudCore must be deployed on the cloud side. CloudCore is composed of three main modules: CloudHub, EdgeController, and DeviceController. CloudHub acts as a communication broker between the cloud-side controllers and the edge components. EdgeController bridges the Kubernetes API server and EdgeCore, enabling workload orchestration, while DeviceController manages device metadata and status information in a dynamic manner.

On the edge side, a container runtime such as Docker or containerd must be installed alongside EdgeCore. EdgeCore itself is composed of multiple lightweight modules, including EdgeD and EdgeHub, which are designed to operate with a reduced memory footprint. EdgeD is responsible for managing the lifecycle of pods on the edge node, whereas EdgeHub provides the communication channel between the edge and the cloud.

In summary, KubeEdge enables Kubernetes-based orchestration across cloud and edge environments through the deployment of complementary components on both sides. Although it is designed to operate on resource-constrained edge nodes, KubeEdge does not aim to provide a unified networking, storage, or service fabric spanning the entire virtual infrastructure. As a result, it is particularly well suited for applications oriented to Internet of Things (IoT) that execute at the edge, collect local data, and forward it to cloud-based services for further processing and storage.

3.1.2 Karmada

Karmada (Kubernetes Armada)[3] is a Kubernetes management system designed to enable cloud-native applications to run across multiple clusters and cloud environments without requiring application-level modifications. It builds on Kubernetes-native APIs and introduces advanced scheduling mechanisms to support multi-cluster deployments, as illustrated in Figure 3.2. The primary objective of Karmada is to automate application management in hybrid and multi-cloud scenarios, providing centralized control, high availability, failure recovery, and traffic scheduling across heterogeneous infrastructures.

Karmada is compatible with the Kubernetes native API and allows a smooth transition from single-cluster to multi-cluster deployments while preserving integration with the existing Kubernetes toolchain. However, most of its advanced functionalities rely on the introduction of Custom Resource Definitions (CRDs). As a consequence, users and DevOps operators are required to adopt new abstractions and interaction patterns, since traditional Kubernetes primitives alone are insufficient to fully control the virtualized continuum exposed by Karmada.

The system provides a set of built-in deployment and scheduling policies that support cluster affinity, workload distribution and rebalancing across clusters, and multi-

[3] https://karmada.io

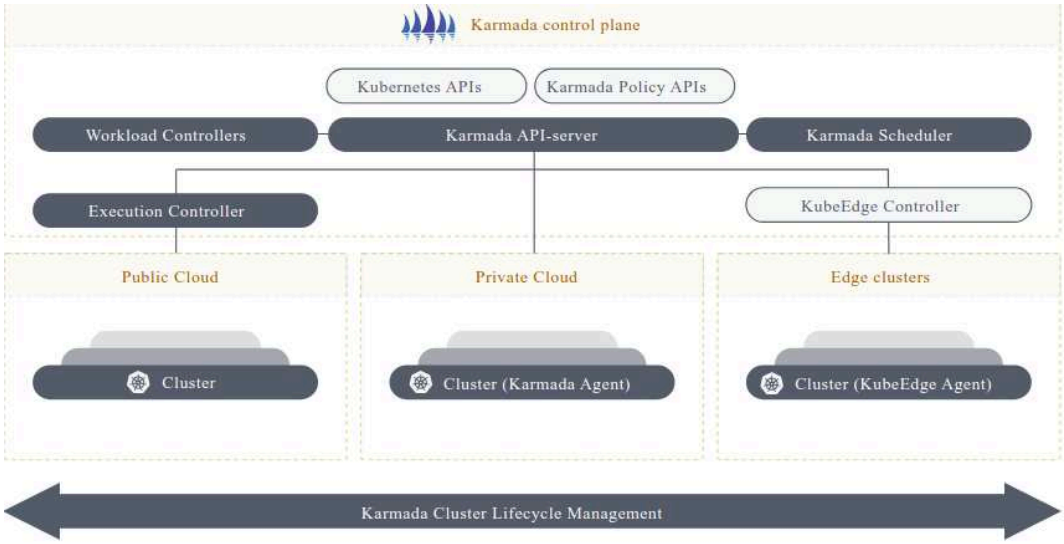


Fig. 3.2 Karmada overview

dimensional high availability. Through these mechanisms, Karmada enables centralized management of Kubernetes clusters deployed in public clouds, on-premises infrastructures, and edge environments.

To manage multi-cluster resources, Karmada introduces several core abstractions, including Resource Templates, Propagation Policies, and Override Policies. Its architecture is centred around the Karmada Control Plane, which consists of the Karmada API Server, Controller Manager, Scheduler, and an etcd datastore. This control plane is responsible for coordinating workload placement, enforcing policies, and maintaining a global view of the managed clusters.

Karmada supports safe isolation between clusters, multiple connection modes, and heterogeneous cloud environments. It enables cluster distribution through various scheduling strategies and supports differential configuration of workloads via the OverridePolicy mechanism. Furthermore, Karmada includes rescheduling capabilities through components such as the Descheduler and the Scheduler Estimator, which can trigger workload redistribution in response to changes in the state of member clusters.

Additional features include support for cluster failover, cluster taints, and a global, uniform view of resources across clusters. Karmada also provides unified authentication through a single API endpoint and enforces access control policies that are consistent with those of the member clusters.

Despite these capabilities, Karmada follows a master–worker model for multi-cluster management, which limits its ability to support a fully federated architecture in which all clusters operate at the same hierarchical level. In this model, a single control-plane cluster is responsible for coordinating the others, introducing a central point of control. Moreover, Karmada does not natively provide a network fabric spanning multiple clusters. Inter-cluster pod-to-pod communication relies on external solutions such as Submariner, which must be deployed independently on each cluster to enable seamless connectivity.

3.1.3 Liqo

Liqo[4] [14] is an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies across heterogeneous on-premises, cloud, and edge infrastructures. It allows clusters to share resources and services originating from different administrative domains, which are aggregated into a unified computing continuum called a *virtual cluster*.

In this model, multiple independent Kubernetes clusters cooperate to provide a unified execution environment while remaining fully autonomous in their management and governance. Each contributing cluster can decide which resources to share and which to keep private. As illustrated in Figure 3.3, clusters (C_x) contribute workloads and services (pods P_x and services S_x) to the virtual cluster. However, some resources may remain local to their originating cluster and therefore remain inaccessible from offloaded workloads. or example, pod P_6 and service S_1 remain accessible only to non-offloaded pods of cluster C_2, and are therefore not exposed to the rest of the computing continuum.

Within the virtual cluster, each remote cluster is abstracted through a *virtual node*. This abstraction represents the aggregated resources of the remote cluster and is fully compliant with standard Kubernetes APIs. Pods scheduled onto such nodes are transparently offloaded and executed in the corresponding remote cluster. Once deployed, these pods become reachable from any other pod in the virtual cluster, regardless of whether it is running locally or in a remote cluster, enabling full pod-to-pod connectivity across the continuum.

[4]https://liqo.io

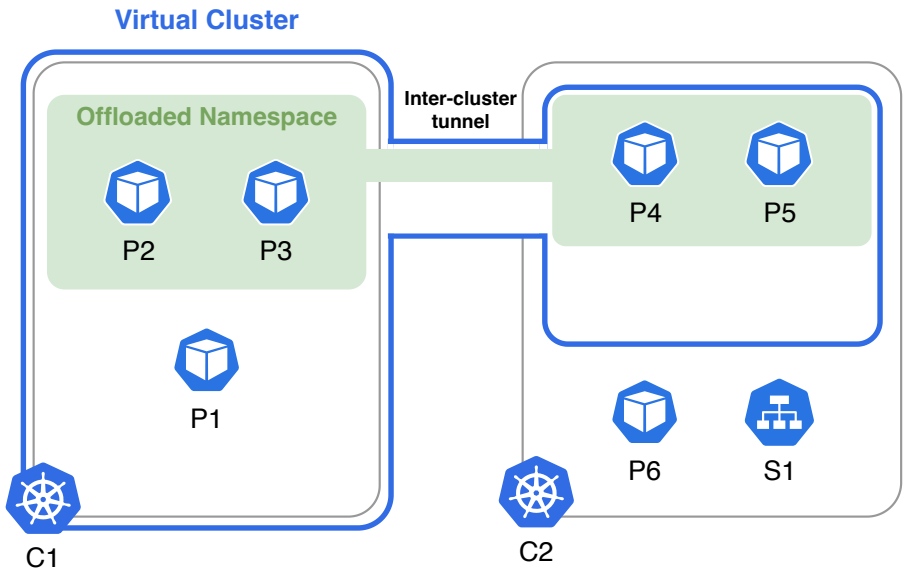


Fig. 3.3 Liqo architecture overview.

Liqo also supports *service offloading*. Services created within the virtual cluster (e.g., Kubernetes *ClusterIP* services) can be accessed by any pod belonging to the continuum, independently of the cluster where the corresponding backend pods are running. This allows applications to span multiple clusters while maintaining the same service abstraction used in single-cluster Kubernetes environments.

Although Liqo primarily targets cloud-oriented data centres and server-based infrastructures, it can also be deployed on individual devices when combined with lightweight Kubernetes distributions. In such cases, it enables multi-cluster federation across constrained environments, although it is not specifically designed to support advanced features such as real-time scheduling or strict timing guarantees.

To enable seamless multi-cluster operation, Liqo provides four core capabilities: peering, offloading, network fabric, and storage fabric.

- **Peering:** Peering establishes a unidirectional relationship between two Kubernetes clusters, which may simultaneously act as providers and consumers in multiple peerings. The peering procedure includes authentication, parameter negotiation, virtual node creation, and network configuration. During this phase, clusters exchange relevant networking parameters such as *PodCIDR* and *ServiceCIDR*. Since clusters may employ heterogeneous networking configurations and Container Network Interface (CNI) plugins, overlapping Internet

Protocol (IP) ranges may occur. Whenever possible, Liqo prefers communication without Network Address Translation (NAT) when address spaces do not overlap. Otherwise, network address translation rules are automatically configured in the Liqo gateway, the component responsible for managing the inter-cluster tunnel. Once this configuration is completed, a secure Virtual Private Network (VPN) tunnel is established between the clusters.

- **Offloading:** Offloading allows workloads to be executed on remote clusters through the virtual node abstraction. Pods scheduled onto a virtual node are transparently executed in the corresponding remote cluster. Liqo supports both stateless and stateful workloads by creating twin namespaces for offloaded pods, ensuring workload isolation and consistency across clusters. Selected control-plane information is propagated and synchronized between clusters, enabling proper execution and resiliency of remote workloads.

- **Network fabric:** The network fabric provides transparent pod-to-pod connectivity across clusters, ensuring that workloads belonging to the virtual cluster can communicate regardless of their physical location. This mechanism allows applications to operate seamlessly even when components are distributed across multiple clusters. The networking layer supports clusters with heterogeneous networking configurations and overlapping IP spaces through the mechanisms negotiated during the peering phase. Recent Liqo versions (from version 0.10 onward) also introduce initial support for security policies that allow administrators to segment cross-cluster traffic and restrict communication only to authorized workloads.

- **Storage fabric:** The storage fabric simplifies multi-cluster storage management by deferring volume binding until the first consumer is scheduled. By exploiting the concept of data gravity, Liqo preferentially schedules pods onto clusters that already host the required storage pools, improving data locality and minimizing unnecessary data transfers across clusters.

Liqo supports different multi-cluster deployment models. Clusters may be organized in a master-worker configuration, where a designated cluster coordinates the others, or operate in a fully decentralized federation where all clusters interact as peers without a centralized control plane. Unlike many other multi-cluster solutions,

Table 3.1 Most promising projects to build the computing continuum.

	KubeEdge	**Karmada**	**Liqo**
Kubernetes Compatibility	Partial (Edge-focused)	Full (via CRDs)	Full (vanilla APIs)
Managed Entity	Node	Cluster/Node	Cluster
Multi-cluster Networking	External	External	Built-in
Resource Footprint	Low	High	High
Maturity	Production-ready	Production-ready	Production-ready
GitHub Stars	7401	5355	1418
GitHub Contributors	323	303	60
First Public Release	Sept 2018	Nov 2020	Nov 2019

Liqo includes a built-in network fabric that ensures transparent communication between pods even when workloads are offloaded across clusters.

A key advantage of Liqo is that each participating organization retains full control over its infrastructure. Resource sharing is voluntary and reversible: a cluster can terminate the peering relationship at any time, thereby revoking access to its shared resources and tearing down any workloads that were offloaded onto it. This autonomy enables organizations to securely collaborate while maintaining governance over their local infrastructure.

3.1.4 Comparison summary

This section provides a compact comparison of the most promising solutions presented in the previous sections. The comparison is summarized in Table 3.1, which highlights the key characteristics of each solution and facilitates a direct assessment of their similarities and differences. GitHub statistics are reported as indicative measures of community adoption and development activity, rather than as direct indicators of technical superiority[5].

KubeEdge primarily targets the far edge of the network, focusing on resource-constrained and often isolated devices. As such, while it plays a crucial role in

[5]Data about GitHub stars and contributors has been retrieved in March 2026.

integrating tiny edge devices into Kubernetes-based infrastructures, its design is less aligned with cloud-native continuum scenarios spanning edge and cloud data centers. From this perspective, Karmada and Liqo represent more suitable candidates for managing distributed cloud-native infrastructures.

Karmada provides centralized multi-cluster management through a specialized API server and advanced scheduling mechanisms. However, its reliance on custom APIs and CRDs limits compatibility with vanilla Kubernetes primitives. Moreover, Karmada enforces a master–worker model, which restricts the range of supported topologies, and it does not natively provide a network fabric for inter-cluster pod connectivity.

In contrast, Liqo is almost fully compatible with vanilla Kubernetes and supports both hierarchical and fully federated multi-cluster topologies. By allowing clusters to operate at the same hierarchical level, Liqo enables more flexible and decentralized continuum configurations. Additionally, Liqo includes a built-in network fabric, offering transparent multi-cluster and pod-to-pod connectivity without relying on external networking solutions.

While KubeEdge and Karmada benefit from larger contributor communities, as shown in Table 3.1, the choice of Liqo as the foundation for this thesis is driven by architectural fit rather than by community size: full compatibility with vanilla Kubernetes, a built-in network fabric, and native support for decentralized, non-hierarchical topologies: these properties align directly with the cloud continuum vision pursued in this work.

Overall, the comparison indicates that Liqo provides the most complete set of continuum-oriented capabilities among the production-ready solutions considered here, particularly due to its support for flexible multi-cluster topologies and its integrated networking support. These observations are used in the remainder of the chapter and inform the architectural direction presented later in this thesis.

3.2 Research-oriented proposals

This section analyses solutions proposed by the research community, including both conceptual research contributions (e.g., works that do not provide a corresponding software prototype) and experimental implementations such as proofs-of-concept

or research-oriented prototypes. The solutions and tools presented in this section should be interpreted as representative of the current state of the art in the literature, rather than as production-ready systems. Nonetheless, they provide valuable insights and design principles that can inform and inspire future research and development activities in the context of the computing continuum.

In contrast to the previously discussed open-source platforms, these solutions primarily focus on architectural concepts, abstractions, and mechanisms, often evaluated under controlled or limited experimental settings.

3.2.1 Cloudlets

Mobile cloud computing aims to overcome the limited computational resources of mobile devices by offloading resource-intensive tasks to remote cloud infrastructures. However, for latency-sensitive and real-time applications, this approach can be ineffective due to the physical distance between users and centralized cloud data centres, which introduces significant Wide Area Network (WAN) latency. To mitigate this limitation, the concept of cloudlets was introduced [41, 42].

Cloudlets are trusted, resource-rich computing nodes deployed in close proximity to mobile users, for example near or co-located with wireless access points. By leveraging this proximity, mobile devices can rapidly instantiate customized Virtual Machines (VMs) on cloudlets and execute applications in a thin-client fashion, significantly reducing end-to-end latency compared to traditional cloud offloading.

Performance can be further improved through dynamic partitioning of applications into multiple components. In this model, latency-critical components are executed on nearby cloudlets, while less time-sensitive components can be offloaded to more distant cloud infrastructures. This flexible allocation enables cloudlet resources to be prioritized for real-time processing, while still benefiting from the scalability of remote clouds. The cloudlet infrastructure itself is dynamic, allowing devices to join or leave cloudlets at runtime.

Two main cloudlet deployment models are commonly considered. In the *ad-hoc cloudlet* model, cloudlet nodes are dynamically discovered within a local-area network. Each node runs a Node Agent capable of spawning Execution Environments to host application components. A Cloudlet Agent is responsible for managing the cloudlet as a whole, recalculating deployment decisions and migrating components

whenever nodes join or leave the system. In contrast, the *elastic cloudlet* model relies on a virtualized infrastructure in which cloudlet nodes are instantiated as virtual machines. In this case, the Cloudlet Agent can dynamically scale the cloudlet by spawning additional nodes when resource demand increases or shutting them down when resources are underutilized.

Evaluation studies demonstrate that a component-based application design, where applications are decomposed into multiple functional modules, such as microservices, can significantly improve performance in latency-sensitive scenarios, including augmented reality. These studies also show that centralized cloud infrastructures are generally unsuitable for hosting real-time constrained components. Moreover, the hierarchical decision-making model adopted by cloudlet architectures enables globally optimized resource allocation, avoiding conflicting local decisions that may arise in purely distributed approaches. However, these works also highlight the need for developer-facing tools that support the creation of component-based applications and the specification of quality constraints without introducing excessive complexity. Ideally, such tools should assist developers in automatically or semi-automatically partitioning applications into components suitable for distributed execution.

3.2.2 FLEDGE

FLEDGE [43] is a lightweight container orchestration system designed for resource-constrained edge devices while maintaining compatibility with Kubernetes. The system integrates a modified Virtual Kubelet and a VPN to enable direct connectivity with Kubernetes clusters. On edge devices, FLEDGE components are deployed as a single entity referred to as the FLEDGE agent. Acting simultaneously as a Kubelet and as a container networking component, FLEDGE eliminates the need for a traditional CNI layer, which is typically positioned between the Kubelet and the network plugin.

Given the limited number of pods that can be deployed on low-resource edge devices, FLEDGE adopts a simple yet effective pod networking mechanism that assigns IP addresses on a first-come, first-served basis. This networking component is also responsible for configuring network namespaces correctly, independently of the underlying container runtime. By labelling edge nodes appropriately, the deployment of standard Kubernetes network plugins is avoided. Experimental

evaluations indicate that FLEDGE requires approximately 60MiB of memory and 78MiB of storage to operate on a Raspberry Pi 3, including all dependencies. These requirements are significantly lower than those of standard Kubernetes and K3s, and are comparable to those of KubeEdge.

FLEDGE leverages Kubernetes IP address ranges to configure container networking, thereby isolating pod networking within individual nodes without affecting the rest of the cluster. From the perspective of the Kubernetes control plane, this approach is transparent, and conventional container networking plugins continue to operate normally on non-edge nodes. However, monitoring resource availability on edge devices remains challenging. Since the operating system and orchestration components can consume a substantial fraction of available resources, decisions based solely on pod-allocated resources may be insufficient to determine whether additional workloads can be scheduled.

To address connectivity across heterogeneous networks and to enhance security, FLEDGE establishes an OpenVPN-based tunnel and builds both cluster and container networking on top of this. While this approach simplifies connectivity, it introduces several drawbacks. First, the effectiveness of packet encryption depends on the chosen cryptographic algorithms, and encryption may be disabled to improve performance. Second, the use of a VPN incurs additional computational and networking overhead, which can negatively affect scalability. Finally, physical access to an edge device may allow unauthorized access to cluster services through the VPN, making additional physical and operating-system-level security measures necessary.

Both the OpenVPN and FLEDGE containers require privileged execution to manage network interfaces, namespaces, and control groups. As FLEDGE executes Kubernetes deployments, the agents must run with root privileges, which introduces potential security risks. In particular, compromised agents could provide unauthorized access to container images and workloads, making them an attractive attack vector in scenarios involving proprietary or sensitive software.

Within the FLEDGE architecture, the Virtual Kubelet plays a limited role, primarily enabling communication with Kubernetes control planes. Its physical location is not critical, as API calls can be forwarded through a custom broker mechanism. Two deployment strategies are supported:

- **Cloud-based Virtual Kubelet:** The Virtual Kubelet is executed as a pod in the cloud, separate from FLEDGE agents, with Kubernetes API calls forwarded to edge devices via REpresentational State Transfer (REST) interfaces. This configuration reduces resource consumption on edge nodes and improves overall robustness.
- **Edge-based Virtual Kubelet:** The Virtual Kubelet is integrated directly into the FLEDGE agent and executed as a container or native process. While this approach reduces architectural complexity, it increases resource consumption on edge devices and decreases resilience to network failures.

Preliminary evaluations show that FLEDGE deployments on ARM-based devices are significantly more resource-efficient than their x86 counterparts. This difference is particularly evident when using Docker, where x86 deployments require approximately three times more storage than ARM-based ones. On ARM platforms, FLEDGE deployments using Docker and Containerd consume up to 50% and 65% less memory, respectively. Among the evaluated runtimes, Containerd emerges as the most suitable choice for FLEDGE. Overall, an ARM-based FLEDGE deployment using Containerd requires approximately 80MiB of storage and 50MiB of memory, including the VPN client container. These results indicate that FLEDGE provides Kubernetes-compatible orchestration with a significantly reduced resource footprint, making it a viable solution for highly constrained edge devices.

3.2.3 Decentralized Kubernetes federation control plane

In their position paper [44], the authors propose a vision for a fully distributed and decentralized control plane for Kubernetes federation. The primary objective is to enable federation at scale, potentially supporting thousands of Kubernetes clusters to address the requirements of next-generation edge–cloud deployments. A key motivation of this work is to improve robustness in the presence of network partitions and disconnected operation, such as split-brain scenarios, which are common in edge environments. The proposed approach preserves cluster autonomy while enabling collaborative handling of failures and coordination across clusters, and is explicitly designed to scale to large, geographically distributed deployments.

The proposed federated control plane focuses on two core operations: the configuration of federated resources and the propagation of decisions and associated

state to the clusters affected by those resources. Federated resources must be continuously defined, updated, and removed, while placement decisions and configuration data must be disseminated consistently across participating clusters. To achieve this without relying on centralized components, the authors propose replacing the traditional central KubeFed controller with a set of distributed federation controllers, one deployed in each cluster. These controllers locally apply templating, placement constraints, and value substitutions on federated resources, thereby maintaining cluster autonomy while improving resilience and scalability.

Distributed federation is enabled through a shared federated database based on Conflict-free Replicated Data Types (CRDTs) [45]. CRDTs provide mathematically provable guarantees of eventual consistency and conflict-free convergence, making them suitable for decentralized environments. The shared data structures are replicated across clusters using epidemic dissemination mechanisms, such as gossip protocols [46]. To support this architecture, the Kubernetes API server is conceptually extended to distinguish operations on federated objects from those on cluster-local resources. Rather than communicating directly, federation controllers coordinate indirectly through the shared CRDT-based database, ensuring that all clusters maintain a consistent view of the desired global state and the current system status.

Given the dynamic and uncertain nature of resource availability in edge environments, the federation controllers continuously update shared data structures with information about the number of pods belonging to a given federated deployment that are currently running in each cluster. When ideal scheduling decisions cannot be realized due to local resource constraints, clusters can collaboratively redistribute workloads by reacting to changes in the shared state. This mechanism enables adaptive and cooperative resource management without requiring global synchronization or centralized decision-making.

Overall, the proposed architecture demonstrates how a distributed federated database can be leveraged to enable Kubernetes federation without relying on a centralized controller or a globally shared etcd instance. While the work remains conceptual and does not provide a production-ready implementation, it introduces important design principles for scalable, resilient, and decentralized control planes that are highly relevant for future computing continuum and edge–cloud federation architectures.

3.3 Past and discontinued projects

For the sake of completeness, this section reviews solutions that have seen limited adoption, have been deprecated, or have gradually lost support from their respective communities. Although these projects are no longer actively developed or considered state of the art, they provide valuable insights into early design choices, limitations, and challenges encountered in multi-cluster and distributed orchestration systems. Analyzing these past efforts contributes to a more comprehensive understanding of the evolution of computing continuum technologies.

3.3.1 KubeFed

KubeFed (Kubernetes Cluster Federation)[6] was an early attempt to provide multi-cluster management for Kubernetes through a unified set of APIs hosted on a control-plane cluster. Its primary goal was to enable the synchronization and propagation of Kubernetes resources across multiple clusters, supporting use cases such as multi-region deployments and disaster recovery. KubeFed introduced core abstractions such as templates, placement rules, and overrides to define how resources should be distributed and customized across federated clusters. These mechanisms provided a foundation for higher-level behaviours, including policy-based placement and dynamic scheduling.

Despite its conceptual relevance, KubeFed underwent multiple incompatible specification revisions (v1 and v2), which fragmented development efforts and hindered adoption. As a result, the project was gradually abandoned by the community, which later converged on Karmada as its de facto successor. At the time of this transition, Liqo was not considered a direct successor, as it was not yet part of the Cloud Native Computing Foundation (CNCF) ecosystem.

3.3.2 Admiralty

Admiralty[7] is a Kubernetes controller-based system designed to distribute workloads across multiple clusters. It enables the execution of workloads on remote clusters

[6] https://github.com/kubernetes-retired/kubefed
[7] https://admiralty.io

by representing them locally through proxy pods, which are scheduled onto virtual-kubelet nodes corresponding to target clusters. In Admiralty, source clusters delegate workload execution to target clusters, while maintaining synchronization between proxy pods and their remote counterparts. This mechanism allows pod dependencies and related resources to follow delegated workloads transparently.

Admiralty can be considered an early alternative to more recent multi-cluster federation frameworks. While not a direct predecessor, it shares several conceptual similarities with Liqo, particularly in its use of virtual nodes and workload delegation across clusters. Despite its conceptual relevance, Admiralty remains a niche solution rather than a widely adopted general-purpose federation layer, with a number of more actively maintained multi-cluster frameworks having gained greater traction in the meantime.

3.4 Other related solutions

For the sake of completeness, this section briefly reviews additional solutions that are not directly intended to realize a computing continuum as defined in this thesis, but nonetheless exhibit conceptual overlaps or explore related design spaces. The discussion emphasizes their scope and their limitations with respect to continuum requirements, rather than implementation details.

Several solutions primarily focus on *device-level* edge enablement. EVE-OS[8] provides an operating-system-level foundation for distributed edge deployments, supporting containers, Kubernetes, and virtual machines across heterogeneous hardware. Its emphasis on lifecycle management and security is particularly relevant for devices deployed in the field; however, it does not provide native multi-cluster federation nor unified fabrics across sites. Similarly, BalenaOS[9] targets embedded and IoT deployments with robust networking and remote management features, but it remains centred on fleet and device management rather than cross-cluster orchestration.

Other projects explore *alternative runtimes* for managing workloads on a single node. Aurae[10] proposes a memory-safe, API-driven runtime daemon (written in Rust) to manage containers, virtual machines, and isolated execution environments with

[8] https://lfedge.org/projects/eve/
[9] https://www.balena.io/os
[10] https://github.com/aurae-runtime/aurae

strong security primitives. While promising for enterprise-grade device management, it is still evolving and does not address multi-cluster federation or continuum-wide abstractions.

A different direction is represented by *decentralized infrastructure* platforms such as ThreeFold[11], which proposes a peer-to-peer model for exposing compute, storage, and networking resources contributed by independent participants. Although its decentralization principles are relevant to the broader discussion on continuum governance and autonomy, the model is not Kubernetes-native and does not provide seamless multi-cluster orchestration aligned with the continuum approach considered in this work.

With respect to *multi-cluster connectivity*, Submariner[12] enables pod-to-pod and service-to-service communication across Kubernetes clusters through an overlay network and optional service discovery mechanisms. In practice, Submariner is often used as a complementary component in multi-cluster systems (e.g., to provide the network fabric required by higher-level orchestration frameworks), but it addresses networking only and does not provide orchestration, scheduling, or storage federation on its own. Cilium Cluster Mesh[13] offers similar objectives through a Cilium-based mesh, providing high performance and observability. However, it requires homogeneous cluster configurations (Cilium as the CNI and coordinated IP addressing), limiting applicability in heterogeneous environments. Notably, it can be integrated with Liqo by delegating multi-cluster networking to an external fabric.

Finally, a set of tools focuses on *multi-cluster management* rather than unified execution across clusters. VPN-based solutions such as EdgeVPN[14] can provide site-to-site or mesh connectivity between clusters, but remain limited to networking concerns. Rancher Fleet[15] provides GitOps-based deployment and monitoring across large numbers of clusters, improving consistency and auditability, but it treats clusters as independent targets rather than a single logical infrastructure. Open Cluster Management[16] similarly supports scalable governance and policy distribution via a hub-and-spoke architecture, including some degree of disconnected operation, yet

[11] https://threefold.io
[12] https://submariner.io
[13] https://cilium.io/use-cases/cluster-mesh/
[14] https://edgevpn.io
[15] https://fleet.rancher.io
[16] https://open-cluster-management.io

it does not establish unified network, service, or storage fabrics and therefore only partially matches continuum goals.

3.5 Concluding remarks

This chapter presented a comprehensive overview of technologies and research efforts relevant to the realization of a computing continuum, spanning production-ready platforms, research-oriented proposals, past and discontinued projects, and complementary solutions addressing specific aspects of distributed infrastructures. By structuring the discussion according to maturity and scope, the chapter provided a critical perspective on the current technological landscape and its evolution.

The analysis of mature open-source platforms showed that only a limited number of solutions are sufficiently stable, actively maintained, and feature-complete to serve as practical foundations for a computing continuum. Among these, KubeEdge, Karmada, and Liqo emerged as the most relevant candidates. KubeEdge primarily targets the far edge, enabling the integration of highly resource-constrained devices into Kubernetes-based ecosystems. Karmada extends Kubernetes toward multi-cluster orchestration through centralized control and advanced scheduling mechanisms, but remains constrained by a hierarchical control-plane model and the absence of a native network fabric. Liqo, in contrast, supports both hierarchical and fully federated multi-cluster topologies, maintains strong compatibility with vanilla Kubernetes, and provides integrated networking and storage abstractions, making it the most comprehensive solution among the analysed production-ready platforms.

Research-oriented proposals explored architectural directions that go beyond current implementations, addressing challenges such as decentralized coordination, dynamic workload partitioning, and resilient operation under network partitions. Although these solutions are not production-ready, they introduce key concepts, such as CRDT-based coordination, collaborative scheduling, and component-level offloading, that significantly inform the design space and highlight promising directions for future continuum architectures.

The discussion of past and discontinued projects illustrated how early design choices, including rigid centralization, limited extensibility, or ecosystem fragmentation, hindered long-term adoption. Similarly, the review of other related solutions

demonstrated that many existing systems focus on individual concerns, such as networking, device management, runtime isolation, or GitOps-based deployment, without providing a unified abstraction that spans compute, storage, and networking across distributed clusters.

Overall, the chapter highlights that realizing a computing continuum requires more than multi-cluster management or connectivity alone. It demands unified abstractions that treat distributed clusters as first-class entities, support flexible and non-hierarchical topologies, and enable transparent execution across heterogeneous environments while preserving autonomy and resilience. Despite substantial progress, current solutions still exhibit open challenges, including limited resource visibility and negotiation across clusters, security across administrative domains, and the lack of decentralized yet coordinated control mechanisms.

These observations motivate the need for architectural approaches that build upon existing platforms while addressing their limitations, and provide the foundation for the design choices introduced in the following chapters.

Taken together, the solutions reviewed in this chapter share three limitations that motivate the contributions presented in Chapter 4. First, none provides a standardized, security-aware protocol for advertising and reserving heterogeneous resource types (Kubernetes clusters, VMs, sensors, services, datasets) across independently administered domains (Liqo's peering is built-in but does not generalize beyond Kubernetes-native resources). Second, resource discovery in all three solutions assumes some degree of prior administrative coordination between clusters, rather than supporting fully decentralized, trust-minimized discovery among previously unknown parties. Third, none of the reviewed solutions addresses data governance or regulatory compliance as a first-class concern in cross-domain resource sharing. Chapter 4 introduces the REAR protocol and the FLUIDOS architecture specifically to close these three gaps.

Chapter 4

Building the Cloud Continuum

As discussed in the previous chapters, current computing infrastructures remain largely fragmented into isolated clusters, each governed by its own control plane. While modern applications already span multiple locations, from the far edge to the cloud, their deployment logic remains tightly coupled to specific infrastructures (Chapter 1), and existing multi-cluster technologies only partially address this limitation (Chapter 3). This chapter presents the FLUIDOS approach to building a true computing continuum, introducing three distinctive transparency properties and the architectural mechanisms to realize them. These contributions were previously introduced in [23, 21, 22] and are presented here in a unified form[1].

4.1 The continuum pillars

The computing continuum may initially appear to be an existing reality. For example, a single Kubernetes cluster spanning multiple infrastructure layers could seem sufficient to support many continuum-related use cases. However, such an approach still relies on a centralized control plane managing all physical nodes, which limits

[1]This work was conducted in collaboration with Stefano Galantino, Elisa Albanese, Nasir Asadov, Stefano Braghin, Francesco Cappa, Andrea Colli-Vignarelli, Amjad Yousef Majid, Eduard Marin, Lorenzo Moro, Liubov Nedoshivina, Fulvio Risso, Domenico Siracusa, Antonio Fernando Skármeta Gómez, and Luca Zuanazzi. It was supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative. Finally, I would like to thank Matteo Cicilloni for his support.

the ability to incorporate localized intelligence and autonomous decision-making across different parts of the infrastructure.

As a result, current solutions only partially realize the vision of the computing continuum. This section therefore identifies three key characteristics that are still missing in existing approaches and that are explicitly addressed by the FLUIDOS architecture.

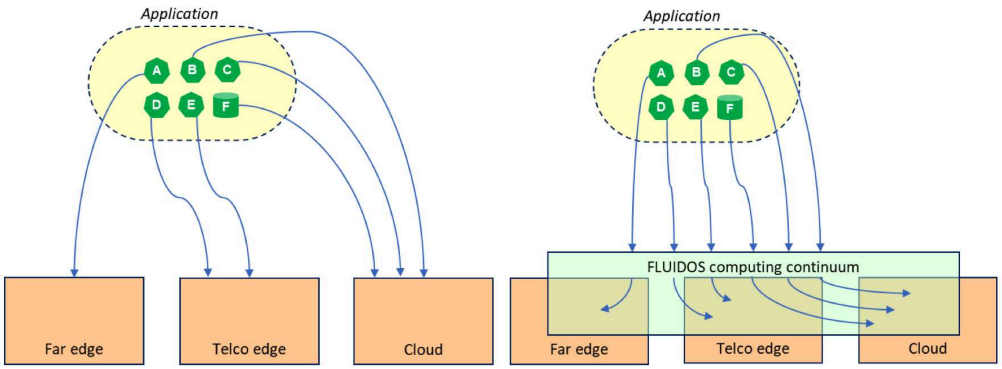
4.1.1 Deployment transparency

When microservices are deployed in the current silo-based computing environment, each component must be explicitly configured to run in a specific location, such as an edge data center or the cloud. The placement of each component is therefore predetermined and fixed a priori, and changing its location requires initiating a new deployment phase (Figure 4.1a). Consequently, performing dynamic optimizations at runtime becomes complex or even impossible, as it would require an overarching orchestrator capable of redeploying components to new locations, something that is not supported by current technologies.

In contrast, Liquid Computing introduces an intent-based interface that automatically places each microservice in the most suitable location based on application requirements, infrastructure conditions, and security or privacy constraints (Figure 4.1b). This approach also enables dynamic adjustments during operation when needed. As a result, developers and DevOps teams interact with a unified deployment and control interface, while FLUIDOS transparently ensures that services are executed in the most appropriate location.

4.1.2 Communication transparency

The communication between microservices differs depending on whether they belong to the same communication domain (e.g., the same Kubernetes cluster) or to different clusters. In Kubernetes, communications within a cluster are typically allowed by default, whereas communications from external components are restricted. Moreover, services that accept internal communications rely on primitives such as the *ClusterIP* service, while services exposed outside the cluster require different mechanisms, such as *NodePort* or *LoadBalancer* services.



(a) Current silos-based computing continuum. (b) FLUIDOS computing continuum.

Fig. 4.1 Computing continuum: deploying apps with traditional approach vs. FLUIDOS.

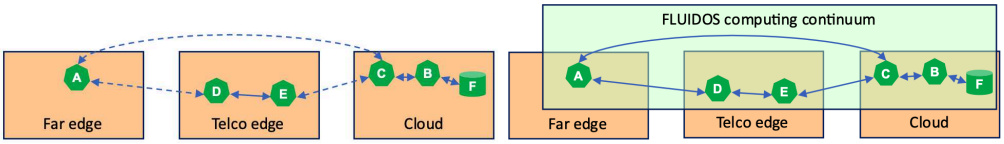
As a consequence, microservices must be explicitly configured to interact with each other based on their deployment location (Figure 4.2a). This requirement increases configuration complexity and complicates redeployment, as communication settings must be adapted whenever services move across clusters or domains.

Some technologies can partially mitigate this limitation. For example, microservices may communicate through message brokers based on publish/subscribe paradigms, such as *Apache Kafka*. However, this approach requires all services to rely on the same communication primitives, which is not always feasible. In particular, pub/sub systems may introduce additional latency and may not be suitable for applications built on other protocols, such as HTTP or gRPC Remote Procedure Call (gRPC).

FLUIDOS addresses this challenge by enabling a virtual cluster spanning multiple physical clusters, where applications operate within a unified virtual environment. In this model, communications between microservices are mediated by the FLUIDOS virtual network fabric, which guarantees seamless connectivity regardless of the physical location of each service. As a result, services communicate as if they were deployed within the same cluster, eliminating the need for complex and error-prone configurations related to cross-cluster communication (Figure 4.2b).

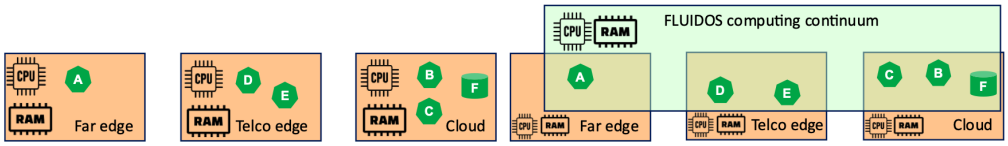
4.1.3 Resource availability transparency

With current technologies, each microservice can use only the resources available within its own cluster. This limitation applies both during normal operation (e.g.,



(a) Current silos-based computing continuum. (b) FLUIDOS computing continuum.

Fig. 4.2 Computing continuum: service-to-service communications with traditional approach vs. FLUIDOS.



(a) Current silos-based computing continuum. (b) FLUIDOS computing continuum.

Fig. 4.3 Computing continuum: available resources with traditional approach vs. FLUIDOS.

when a service is started) and when updates such as automatic scaling are performed. As a consequence, a service may experience disruptions due to a lack of resources in its local cluster, even when resources are available elsewhere in the computing continuum but cannot be leveraged because they belong to a different cluster (Figure 4.3a).

This limitation is generally not critical in large cloud data centers, where resource scarcity is unlikely. Even when a cluster initially uses only a subset of the available infrastructure, the issue can often be mitigated by resizing the cluster, for example by dynamically adding or removing worker nodes. However, the problem becomes more pronounced in edge environments, where the available resource pool is typically limited to a small number of servers. In such scenarios, obtaining additional capacity requires leveraging resources located in nearby domains.

Liquid computing addresses this challenge by enabling the creation of a virtual computing space that spans multiple physical domains. Services deployed within this virtual space can access resources across the entire virtual infrastructure, regardless of their physical location. In FLUIDOS, this allows services to scale seamlessly based on resource availability across all participating domains. For example, different instances of the same service may run simultaneously at the telco edge and in a cloud data center, effectively blurring traditional cluster boundaries (Figure 4.3b).

4.2 REAR protocol

Within the context of the European project FLUIDOS, we argued that the effective utilization of resources within a continuum is contingent upon their recognition and exposure as a unified pool. The dynamism of the continuum can therefore be fostered by allowing devices to advertise/purchase (possibly) any type of resource, ranging from traditional computing resources (e.g., VMs, slices of Kubernetes clusters), to sensors, actuators, and volumes of data. As a result, devices can dynamically join the continuum, advertising the locally available resources and purchasing remote resources when needed, having access to the entire catalogue of possibilities.

This section proposes REAR [21], a novel protocol designed to address the challenges associated with resource advertisement and reservation in the computing continuum. REAR aims to provide a flexible and scalable framework that enables entities within the continuum to advertise their resources and facilitate the reservation of those assets by consumers or applications.

REAR addresses three key objectives. (*i*) *Standardization*, defining common interfaces and messages for resource advertisement and reservation to promote interoperability and compatibility across heterogeneous computing environments. (*ii*) *Efficiency*, to optimize resource allocation and utilization by allowing devices to enrich the resource description including internal energy metrics, latency considerations, and cost models. (*iii*) *Security and Trust*, incorporating mechanisms for authentication, authorization, and secure communication to ensure the integrity and confidentiality of resource transactions.

Throughout this section, we will delve into the design principles, architectural components, and operational workflows of our proposed protocol. Additionally, we will discuss the potential applications and benefits of adopting this protocol in various domains, including edge computing, the IoT, and cloud computing. By presenting this protocol, we aim to contribute to the ongoing efforts aimed at realizing the full potential of the computing continuum by enabling efficient resource management and utilization.

4.2.1 Motivations

Resource discovery plays a pivotal role in distributed continuum infrastructures to prevent underutilized resources and inefficient allocation policies. In the following, we outline the optimizations enabled by the continuum's resource aggregation and highlight the requirements for REAR.

Support for intent-based allocation

Intelligent, data-driven applications can leverage the continuum to find the optimal placement and satisfy user-specified service level requirements, often referred to as *intents*. Intent-driven orchestration is becoming a popular approach in several scenarios beyond workload management [47], for instance, to express user-level requirements for network configurations [48] or to model network security requirements [49]. Nevertheless, the integration within real-world scenarios is even more challenging due to the complexity of modern system architectures, whether programmable network switches or Kubernetes.

Such enhanced capabilities require approaches to gather information not only on existing (local) resources but also on those offered by other participants in the continuum. This enables the optimal utilization of available resources accounting for requirements beyond traditional computing (i.e., Central Processing Unit (CPU), Random Access Memory (RAM)) such as economics, latency, and security/compliance for the deployment and the lifecycle management of applications. The REAR protocol extends the boundaries of the local compute resources to implement intent-based allocation policies. Moreover, such dynamicity allows users to define the desired application architecture using intents, delegating to the infrastructure the task of retrieving and connecting services to the corresponding data sources.

Support for carbon-aware allocation

Increasing energy demand propelled by recent technological trends including the cloud computing [50, 51] underscore a critical dilemma. While the societal and environmental benefits of computing make its expansion both inevitable and desirable, the sustainability of this growth is questionable due to diminishing returns on

hardware efficiency gains [52]. This scenario necessitates the exploration of new paradigms for carbon-aware computing.

A promising decarbonization strategy for a computing continuum advocates for workload scheduling that intelligently shifts computational jobs in space and time to capitalize on locations and periods of lower-carbon electricity availability [53–55]. This location-based approach ensures a genuine reduction in the carbon footprint of a computing continuum, bypassing the pitfalls of market-based strategies by focusing on the true carbon intensity of the grid mix at the time and place of usage. In addition, hardware-embodied emissions can be included in the picture as a way to provide a much more sustainable computing continuum [56, 57]. These are the emissions from the hardware manufacturing phase and they complement the operational emissions described above to provide a more holistic overview of the carbon footprint. Recent advancements in this field [58] demonstrate that it is both possible and essential to include embodied emissions in carbon-aware optimization efforts.

In light of these considerations, the motivation for integrating carbon-aware allocation mechanisms within REAR becomes even more compelling. By embedding such strategies into the fabric of the computing continuum, we not only aim to enhance operational efficiency and collaboration across heterogeneous computing resources but also to pioneer a sustainable approach in such a fluidified cloud-edge continuum.

Support for security features

The REAR protocol is seen as a fundamental component of the computing continuum, facilitating the dynamic (dis)aggregation of assets (resources, data, or capabilities) across various domains. This renders the traditional notion of a static security perimeter obsolete, adhering to the zero-trust approach, where neither asset, provider, or consumer is inherently trusted. Rather, continuous verification, authentication, authorization, and monitoring of all assets is required [59]. In this context, security is not only an integral component of REAR but is also communicated through it.

In particular, our vision entails mutual authentication and authorization for each party involved in the advertisement and reservation process, enabling authorized access to the offered assets. Given the dynamic nature of such a multi-party scenario, recent advancements in the field suggest a shift towards Decentralised Identifier

(DID)[2] [60] and Verifiable Credentials (VCs)[3] [61] as a lightweight alternative to traditional centralized authentication. Some research endeavors also explore integrating this concept with Distributed Ledger Technology (DLT), which can serve as a registration authority and authorization scheme for distributed services [62].

At the same time, the REAR protocol assumes a crucial role in conveying the security attributes delivered by the advertised assets. Let us consider a scenario where a cluster offers a pool of computing resources for others to expand. In this instance, the advertised pool might provide dynamic network policy control, restricting communication solely to services within the pool and refining the network perimeter in real time. Additionally, the advertised features may encompass hardware security capabilities, such as hardware-backed Trusted Execution Environments (TEEs) on compute nodes, to achieve a higher degree of workload confidentiality and integrity (which is particularly relevant for sensitive workloads), and to give tenants the ability to attest the environment (or enclaves) where their workloads run. Furthermore, assets could be equipped with proactive and reactive protection services, such as threat detection and mitigation systems or cyber deception tools, thus providing insights into attackers' tactics and techniques.

In the current design, advertisements are intended to be visible only to nodes that have already completed the enrolment process described in Section 4.2.6 (i.e., that hold a valid VC), rather than being broadcast openly; however, this restricts discovery to already-enrolled parties and does not yet address more fine-grained access control over which enrolled parties may see which advertisements (e.g., limiting visibility to a pre-approved partner list). Extending REAR with advertisement-level access policies, rather than uniform visibility to all enrolled nodes, is identified as an open item for future work.

4.2.2 Related work

Reservation protocols play a key role in multi-user applications, networking, and distributed systems, managing access to resources and ensuring efficient and fair resource allocation. The Resource reSerVation Protocol (RSVP) represents a foundational approach in computer networks, designed to manage resource reservations in

[2] https://www.w3.org/TR/did-core/
[3] https://www.w3.org/TR/vc-data-model/

QoS enabled networks for efficient data traffic delivery [63]. Mobile Resource reSerVation Protocol (MRSVP) [64] has extended the functionality to the context of which mobile devices perform reservations based on their current and future locations and Resource Negotiation and Pricing Protocol (RNAP) [65], which integrates economic factors into the reservation process. Additionally, Resource reSerVation Protocol-Traffic Engineering (RSVP-TE) introduces traffic engineering capabilities, enabling explicit path establishment for data traffic to optimize network utilization [66]. In our perspective, these protocols primarily address network-centric parameters, overlooking the nuanced multi-dimensionality of computing resources (e.g., CPU, RAM, etc.) and the heterogeneity inherent in modern computing platforms (aka the as-a-Service model).

In the realm of distributed systems, the Service Negotiation and Acquisition Protocol (SNAP) [67] and subsequent Service Level Agreement (SLA) negotiation mechanisms [68] present methodologies for establishing QoS agreements, emphasizing the importance of flexible, bilateral negotiation frameworks. Such advancements illustrate the effort to refine SLA negotiation, catering to the dynamic needs of clients and servers within distributed architectures. Furthermore, authors in [69] also describe a brokering architecture that can make advance resource reservations for SLAs, and also the need to consider security and privacy for managing the distributed services [62].

The advent of 5G technology introduces new dimensions to this landscape, offering telecommunication operators unprecedented opportunities to leverage their network and computing infrastructures [70, 71]. With 5G, the requirements for Edge infrastructure intensify[4] [72], necessitating protocols that support seamless application deployment across diverse Telco providers and facilitate the federation of Operator Platforms[5] [73]. Despite the potential, current proposals face limitations, including (*i*) a lack of resource price discovery mechanisms, (*ii*) limited support for dynamic environments, and (*iii*) a focus on containerized applications that may not fully capture the generality of offered resources/services.

This evolving landscape underscores the imperative for innovative reservation protocols that can address the multifaceted challenges of modern computing environments. Such protocols must not only accommodate the complex interplay of network

[4]https://www.gsma.com/futurenetworks/resources/gsma-operator-platform-telco-edge-requirements-2022/

[5]https://www.gsma.com/futurenetworks/resources/east-westbound-interface-apis/

and computing resources but also adapt to the rapidly changing dynamics introduced by advancements like 5G, thereby ensuring efficient, fair, and economically viable resource allocation in distributed systems.

4.2.3 Architecture

REAR has been designed around the concept of *Node*, i.e., a unique computing environment, under the control of a single administrative entity. The node can be composed of one or more machines and modeled with a common, extensible set of primitives that hide the underlying details while maintaining the possibility to export the most significant distinctive features (e.g., the availability of specific services, and peculiar hardware capabilities). In addition, nodes belonging to the same administrative entities can be logically grouped in *Domains*, allowing for enhanced aggregation policies when advertising available resources. In this respect, a given set of resources can be restricted to be advertised only within the same domain, i.e., only the nodes belonging to the same domain can purchase them. Alternatively, the resources can be made available to all participants in the continuum.

Such a hierarchical infrastructure allows for two different interactions among nodes involved, generically referred to as *consumers* and *providers*, depending on the respective role: (*i*) a *Horizontal* interaction enables the creation of a resource exchange process among peers, which can share their resources and services, or part of them, based upon a set of policies. Horizontal interactions are carried out according to a Peer-to-Peer (P2P) paradigm, hence without the need for any centralized entity that controls and supervises the entire process. (*ii*) a *Vertical* interaction introduces new concepts such as aggregation and hierarchical scaling into the picture. Third-party brokering entities can provide endpoints that customers can browse and query to obtain aggregated views of the resources available in one (or more) domain. This, in turn, would allow the creation of the "unique pool of resources" enabled by the continuum. It is worth mentioning that such interaction can be recursively implemented to replicate multi-level hierarchical aggregations.

In the REAR protocol, each node has two different components: a *resource importer* and a *resource exporter*. The former is responsible for the discovery of available resources. Since the number (and type) of resources in the continuum infrastructure can be potentially huge, the component is also in charge of filtering

the available options based on the data models presented in Section 4.2.4. The latter advertises the node's available resources to the other members of the continuum. This approach is still compliant with the hierarchical model, as the resource exporter can perform the advertisement of the resources of the nodes for which it acts as a broker. In addition, a dedicated *contract manager* component keeps track of the purchased resources in the form of a contract and exposes an endpoint that enables further logic to be implemented to verify the SLA mentioned in the contract to ensure that it is, in fact, being upheld.

4.2.4 Data model

The REAR data model outlines the various types of resources advertised in the continuum. This model includes the definition of two terms: *Flavor* and *FlavorType*. The *Flavor* encompasses the set of information shared among all possible resources, while the *FlavorType* is a pointer to another dedicated structure that specifies the unique characteristics of each resource[6]. The complexity of the data model requires a formal, semantic definition of the various components and their relationships. To enable such reasoning we created two ontologies[7] according to the standard defined by the World Wide Web Consortium (W3C), characterizing the relationships between Kubernetes entities, the REAR data model, its relationship with Kubernetes, and the various properties of the *FlavorTypes* described later.

Flavor

The *Flavor* data model provides a structured way to represent and manage different kinds of computing resources and it includes all the information that is independent from the chosen *FlavorType*, hence facilitating interoperability and standardization across various systems and platforms, enabling efficient resource advertisement and reservation within the computing continuum. The *Flavor* data model includes the following information:

- *FlavorID*: A unique identifier for the flavor.

[6]https://github.com/fluidos-project/REAR-data-models
[7]https://github.com/fluidos-project/fluidos-ontology

- *ProviderID*: A unique identifier for the provider of the flavor, which can be different from the owner in case this flavor is being advertised by a broker.
- *Location*: Information about the location of the flavor described using the triplet `<latitude,longitude,altitude>`.
- *NetworkPropertyType*: The type of network property ensured by the provider (e.g., 5G, WiFi, Ethernet).
- *Price*: Information about the price of the flavor, including amount, currency, and billing time (e.g., daily, monthly).
- *Owner*: Information about the owner of the flavor, including domain, node ID, IP address.
- *FlavorType*: A reference to a specific FlavorType schema, allowing for defining details specific to each flavor type.

FlavorType

Currently, five *FlavorTypes* are defined in REAR, which provides flavor-specific data models that describe the computational resources to be purchased. New *FlavorTypes* data models can be added to support more use cases.

K8Slice. It identifies a Kubernetes cluster capturing both its hardware characteristics and various policies related to its deployment and usage within Kubernetes environments. When a K8Slice flavor is purchased, the remote resources can be seen as a logical extension of the local resources, to deploy general-purpose workloads (with Liqo.io). A K8Slice includes the following main information:

- *Characteristics*: Defines the hardware characteristics of the Kubernetes flavor, including CPU, GPU, memory, storage, and the number of pods that can be deployed.
- *Properties*: Specifies additional properties of the Kubernetes flavor, including the expected inter-cluster latency[8], the list of security standards supported (e.g., ISO/IEC 27002) and data-privacy regulations (General Data Protection

[8]At the time of writing, inter-cluster latency is assumed to be estimated either from prior knowledge or via network measurement utilities (e.g., ping) executed by the provider towards the consumer.

Regulation (GDPR)), and the carbon footprint expressed in terms of embodied and operational emissions, computed using standard lifecycle-assessment methodologies (e.g., aggregating manufacturing and end-of-life embodied emissions amortized over the resource's expected lifetime, plus operational emissions derived from measured or estimated power draw and the carbon intensity of the local energy grid). At present, this value is expected to be supplied by the provider at flavor-registration time rather than computed automatically by REAR.

- *Policy*: Defines policies related to the Kubernetes flavor, including the fact that multiple instances of the same FlavorType can be aggregated into a single entity, or partitioned to obtain only a subset of the resources.

VM. The VM FlavorType provides an option to acquire computing resources in the form of VMs. The VM FlavorType shares most of the characteristics of the K8Slice previously described, including the possibility to describe the architecture of the node in which the VM is running as well as information on the Operating System (OS) on the VM. Despite being similar, it is important to model both FlavorTypes separately as the usage of resources differs. For instance, a K8Slice needs to communicate with the Kubernetes API server, while a VM requires a Secure Shell (SSH) connection.

Service. The *Service* FlavorType enables providers to advertise services, following the Software-as-a-Service (SaaS) model. Since it is almost impossible to provide a generic description suitable for all possible services, the *Service* FlavorType data model follows the same pattern used for the Flavor and FlavorType data model: the *Service* FlavorType provides a high-level description of the Service, including all the specifications that are independent of the actual service, whereas the *ServiceType* details the service-specific characteristics. Specifically, the *Service* FlavorType describes:

- *Name* and *Description*: Respectively the name and the human-readable description of the service.
- *Tags*: A keyword list that summarizes a service's properties.
- *Plan*: The plan for the service (e.g., Enterprise, Trial).

- *Latency*: The expected latency with the consumer, measured assuming the consumer is located within the same geographic region as the provider, since latency is primarily determined by physical distance and network path between the two.
- *ServiceType*: The reference to the characteristics of the specific ServiceType. For instance, a *PostgreSQL* ServiceType can include the number of transactions per second or the number of tables that the user can create.

Sensor. The Sensor FlavorType allows for Sensors to be advertised and (possibly) shared among multiple consumers in the continuum. The Sensor FlavorType is characterised by:

- *SensorType*: The type of sensor (e.g., light, humidity).
- *SensorModel* and *SensorManifacturer*: Additional information on the sensor.
- *SamplingRate*: The frequency of the measurements.
- *Accuracy*: The expected accuracy of the measurements.
- *MeasurementUnit* and *SamplingRateUnit*: The unit of measure for both the measurements and the sampling rate.
- *AccessType*: How the measurements can be accessed from the consumer (e.g., HTTP, MQTT).

Purchasing access to physical sensors and actuators raises privacy and security considerations beyond those of purely computational resources: a sensor feed may capture personally identifiable information (e.g., camera footage of a workplace), and an actuator grants a remote party the ability to affect the physical world. While REAR's authentication and authorization mechanisms (Section 4.2.6) apply uniformly across all flavor types, the Sensor and Actuator flavors are expected to require additional, type-specific safeguards, such as purpose-limitation clauses in the negotiated contract, audit logging of actuation commands, and, for safety-critical actuators, human-in-the-loop confirmation before a remote command is executed. A full treatment of these safeguards is left to future work; this thesis focuses on establishing

the resource model and negotiation protocol within which such safeguards could be expressed and enforced.

Data. The *Data* FlavorType allows datasets to be shared and advertised between participants of the continuum. This means that REAR enables a form of data sharing as recommended by various EU-funded initiatives such as GAIA-X Data Spaces[9] and International Data Spaces Association (IDSA) Reference architecture [74]. The Data FlavorType is characterized by:

- *Name* and *Description*: Respectively the name and the human-readable description of the dataset.
- *Tags*: keyword list summarizing the dataset properties.
- *License*: The license(s) regulating the utilisation of data.
- *Plan*: Dataset usage plan (e.g., Free, Enterprise, Trial).
- *Format*: The format of the dataset (e.g., Comma Separated Values (CSV), Tab-Separated Values (TSV), JavaScript Object Notation (JSON)).

Treating Data as a first-class, purchasable resource type introduces security and privacy implications beyond those of compute or sensor resources, including unauthorized redistribution after purchase, re-identification risks for purchased datasets, and compliance obligations that persist after the data leaves the provider's domain. These implications are addressed in depth in Chapter 5, which introduces infrastructure-level data spaces specifically to govern data sharing under regulatory and sovereignty constraints; the present chapter establishes the resource model, while Chapter 5 establishes the governance layer that mitigates these risks in practice.

4.2.5 REAR workflow

REAR defines several messages, which can be classified as either *required* or *optional* (an example is depicted in Figure 4.4).

[9] https://gaia-x.eu/what-is-gaia-x/deliverables/

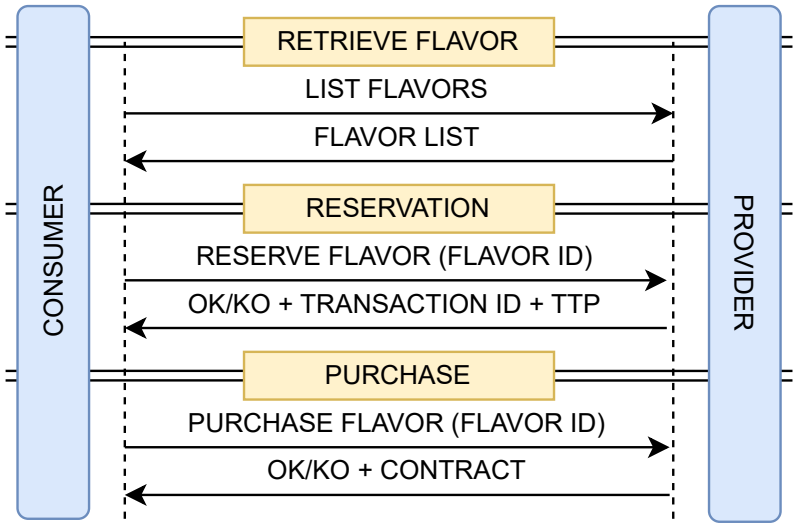


Fig. 4.4 REAR workflow overview, implementable via REST API calls.

List Flavor (required)

The REAR workflow described below assumes that the consumer has already identified a candidate provider; this initial discovery step is handled separately by FLUIDOS's peer discovery mechanism (Section 4.5.2), which allows nodes to discover other nodes connected to the continuum without prior knowledge of their identity. Once a candidate provider has been discovered, the consumer initiates the REAR workflow by sending it a List Flavor message.

This message is sent by the consumer to probe the available flavors offered by a given provider. Since different FlavorTypes can be offered by a single provider, the consumer can filter out possible resources by specifying the desired characteristics in the *List Flavor* message, using the FlavorType data model described earlier. For example, if the consumer wants to purchase VMs, it can retrieve the list of possible VMs offered by a provider by specifying characteristics of the VM FlavorType, such as 2 CPUs and 4GB of RAM. The provider will then reply with a list of VMs that match the requirements, if any.

The *List Flavor* message thus contains the requested FlavorType with the desired characteristics and some form of identification for the consumer with the tuple `<ConsumerID,Region>` to allow the provider to generate a (possibly) customized offer for the consumer.

Reserve Flavor (required)

Once the consumer knows the Flavors and their IDs, the *Flavor* reservation process is performed through the *Reserve Flavor* message, sent by the consumer to inform the provider about its willingness to reserve a specific flavor. Specifically, the consumer/provider interaction can be summarized as follows. After the client has collected the list of available *Flavors* offered by the provider, it notifies the intention of reserving a specific flavor by sending the *Reserve Flavor* message, specifying the ID of the *Flavor* to be reserved. To verify the consumer identity, the *Reserve Flavor* message must also include an authentication token that will be then validated by a Trusted third-party authentication and authorization service. The authentication token included in the message is obtained during the enrolment phase described in Section 4.2.6, preventing a party without a valid credential from reserving resources. Once received, the provider checks if the flavor is still available and if so it replies with a summary of the reservation process including the *TransactionID* and the *Time To Purchase (TTP)*, i.e., the time by which the Flavor must be purchased. This allows reserved Flavors to be released in case either the consumer becomes unreachable, or the subsequent purchase process exceeds a predefined threshold. If the *Flavor* is not available a 404 error message is sent to the consumer.

A successful Reserve message does not strictly require a subsequent Purchase: the reservation holds the resource for a limited time window, after which it is automatically released, allowing the consumer to evaluate the reserved resource (e.g., checking compatibility) before committing to the Purchase step.

Purchase Flavor (required)

The *Purchase Flavor* message is sent by the consumer upon receipt of the provider's response during the reservation phase to complete the purchase of an offered flavor. To do so, the consumer sends the *Purchase Flavor* message including the *TransactionID* (obtained with the *Reserve Flavor*) and the identification token to the provider. If authorized, the consumer will then be prompted to a payment service (either external or managed by the provider) and, if successful, a copy of the *Contract* is returned to the consumer, detailing the purchase and the information required to access the purchased resource (e.g., IP address, API endpoint).

Subscribe / Refresh / Withdraw Flavor (optional)

Given that each offered flavor may not be always available, the consumer can notify the intention to receive continuous updates on a specific set of *Flavors* using the *Subscribe Flavor* message. This internally triggers the creation of a stateful channel between the consumer and the provider, which asynchronously sends back updates for any change in the specified *Flavor* using the *Refresh Flavor* message.

If the *Flavor* is no longer available, the provider can tear down the communication channel related to the specific *Flavor* and notify the consumer that the *Flavor* can no longer be purchased using the *Withdraw Flavor* message.

4.2.6 REAR enrolment and authentication

The REAR protocol foresees a P2P interaction between the *customer* (i.e., the node requesting a given resource) and the *provider* (i.e., the node offering the resources). Both parties must be authenticated to prevent unauthorized access to the resources available in the continuum. At its core, REAR relies on a DIDs system to provide a globally unique identifier to verify the various actors involved in the REAR message exchange and remove the need for a centralized registry. Each entity in the continuum is represented by a set of attributes, constituting what is called a *Verifiable Credential (VC)*.

To participate in the resource exchange process in the continuum, nodes must perform the enrolment towards a Trusted Issuer to obtain a VC. The VC contains all the attributes describing a given node, however, not all of them must be provided when purchasing a given resource. In fact, a given provider might require only a subset of them (e.g., only university and department are required). To this end, a *Verifiable Presentation (VP)* can be requested through the local Privacy and Security Manager (PSM), an entity acting as a Trusted Issuer, including only the subset of requested attributes and preventing information disclosure.

Figure 4.5 details the complete REAR workflow, including also the authentication phase. Upon connecting to a FLUIDOS infrastructure a customer is requested to provide a set of attributes to retrieve the VCs (i.e., the digital proof of identity recognized in FLUIDOS). The VC can then be used to request the related VP containing only a subset of the attributes to be used for future resource negotiation.

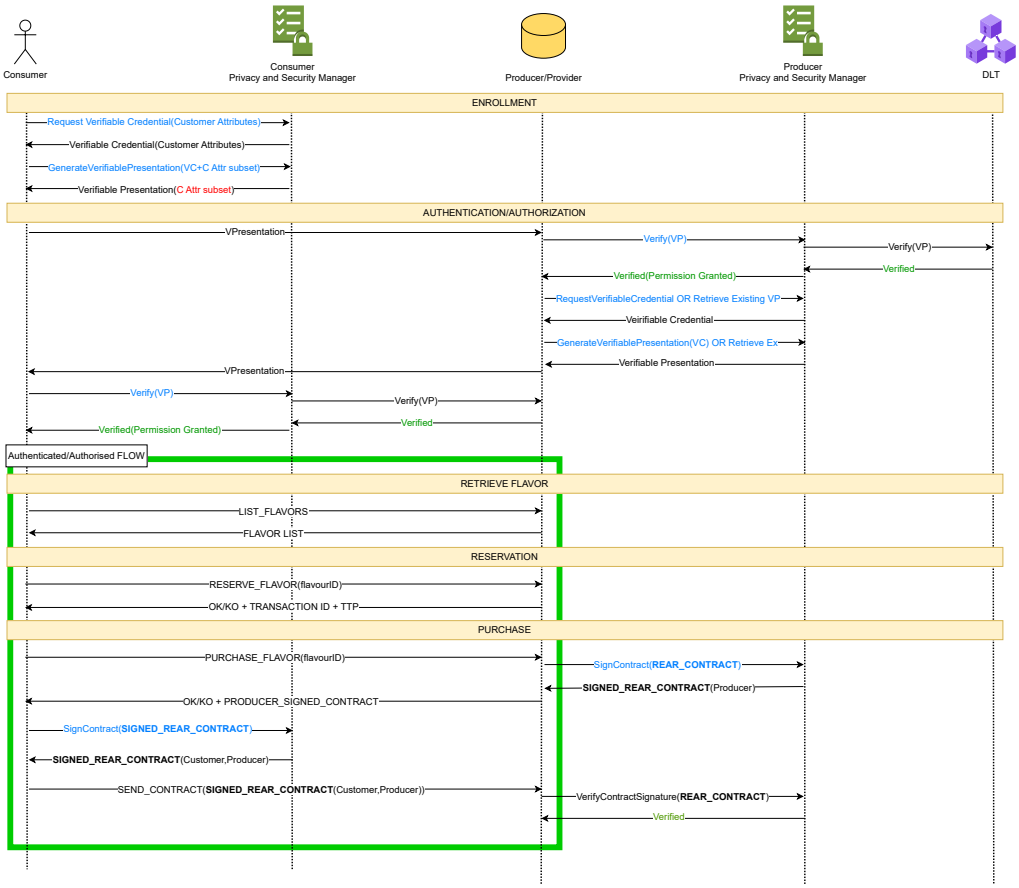


Fig. 4.5 Complete REAR workflow, including authentication.

Before initiating any resource negotiation, the customer and the provider mutually exchange the respective VP to authenticate, following a two-step approach: the customer sends the VP to the provider, which validates it using its own Privacy and Security Manager and a DLT. Upon validating the VP, the provider requests a VP from its Privacy and Security Manager (if not already available in its digital wallet) and sends it to the customer, which, in turn, validates it. Therefore, the Privacy and Security Manager acts as a Policy Decision Point, whereas the REAR protocol acts as a Policy Enforcement Point, generating the contract and granting access to the resources. If this preliminary authentication phase is successful, all the subsequent messages will be authenticated/authorized.

4.3 FLUIDOS

Having introduced REAR as the protocol governing resource discovery, advertisement, and reservation across domains, this section presents FLUIDOS, the broader architecture within which REAR operates. Concretely, REAR provides the negotiation layer that two FLUIDOS nodes use to agree on which resources will be shared and under which terms. FLUIDOS then provides the surrounding architecture (peer discovery, deployment transparency via Liqo, and the liquid computing abstractions described below) that allows the negotiated resources to actually be consumed transparently by applications. In short, REAR answers *which resources can be obtained, from whom, and under what conditions*, while FLUIDOS answers *how those resources are integrated into a usable, transparent computing environment*.

FLUIDOS envisions liquid computing as a continuum of resources and services allowing the seamless and efficient deployment of applications, independently of the underlying infrastructure. We present here the main characteristics of liquid computing, followed by the most significant deployment scenarios.

4.3.1 Main characteristics

The three transparency properties introduced in Section 5.1 describe the user-facing goals of the computing continuum. This section complements them by presenting the architectural characteristics that enable those properties, as originally introduced in [23].

We believe this paradigm shall be composed of the following four distinguishing characteristics.

Intent-driven

A consumer can assign to each workload the desired execution constraints through high-level policies, without knowing about the infrastructural details. Overall, liquid computing brings the cattle service model [75] to a greater scale. Like servers in a data center, with no one caring about where each task is executed, if requirements are fulfilled, this paradigm blurs the cluster borders so that users are relieved from selecting a specific infrastructure for their applications. Yet, different clusters are

associated with different properties (e.g., in terms of geographical location and security characteristics) and, indeed, this is one of the main driving reasons behind cluster sprawling. Thus, it is of utmost importance the adoption of an intent-driven approach, allowing final users to enrich each workload with a set of high-level policies to express the associated constraints (e.g., geographical locality and spreading, costs, capabilities, etc.); automated schedulers shall select the best execution place across the entire border-less infrastructure, depending on the available resources and enforcing in concert the user-specified policies. Yet, we deem at the same time the resource continuum abstraction to enable more contextualized scheduling decisions (given the knowledge about the entire infrastructure), allowing for further optimizations and better scalability compared to the siloed approach.

Decentralized architecture

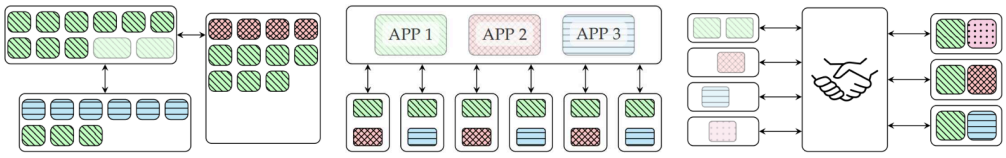
The resource continuum stems from a P2P approach, with no central point of control and management entities, as well as no intrinsically privileged members. Following a decentralized and peer-based model like the Internet, the liquid computing approach fosters the coexistence of multiple actors, including larger cloud providers, smaller, territory-linked enterprises, and even small office/homeowners. Indeed, each entity can autonomously and dynamically decide who to peer with (hence, share resources), similarly to the concept of Autonomous Systems in the Internet inter-domain routing. A dynamic discovery and peering protocol is in charge of the automatic identification of available peers and the negotiation of peering contracts based on the demands and offers of each actor, along with their specific constraints; optionally, the above operations could also be delegated to an intermediate dedicated entity such as a broker. No sensitive information disclosure is mandated (e.g., infrastructural setup), with the entire process possibly involving only the request for a certain amount of abstract resources (e.g., CPU, memory, ...) and the offer of available ones, together with the associated cost. Besides peering establishment, decentralization also concerns the preserved ability of each cluster to evolve independently, thanks to the local orchestration logic, and the support for the creation of arbitrary topologies, with different points of entry for the deployment of different workloads.

Multi-ownership

Each actor maintains full control of his own infrastructure while deciding at any time how many resources and services to share and with whom. Although single clusters are expected to be under the control of a single entity, the entire resource ocean would likely span across different administrative domains. Once a new peering is established, the control plane of the target infrastructure is in charge of configuring the appropriate isolation primitives (e.g., resource quota, network and security policies, ...), based on the underlying orchestration capabilities, to enforce the shared resource slice and prevent noisy neighbors' phenomena. Specifically, we foresee a shared security responsibility model, with the provider responsible for the creation of well-defined sandboxes and the possible provisioning of additional security mechanisms (e.g., secure storage) negotiated at peering time. Requesters, on the other hand, are expected to take measures to fortify their applications (similarly to public cloud computing) and to configure for each sensitive component the appropriate policies to ensure it is scheduled on security-compliant infrastructures only (e.g., private data is processed locally).

Fluid topology

Members can join and leave the virtual continuum at any time, independently from their infrastructure size, from enterprise-grade data centers to IoT and personal devices. Generalizing traditional federation approaches, liquid computing aims at supporting highly dynamic scenarios, with frequent and unexpected (or, in other scenarios, explicitly desired) connections and disconnections. Besides spanning across public and private data centers, as well as edge clusters, the resource continuum encompasses also single devices. This would include IoT, industrial and domestic scenarios, all characterized by a multitude of independent appliances typically dedicated to specific tasks (e.g., machine tools control, home automation, monitoring, etc.), which could greatly benefit from this paradigm, transparently leveraging the shared resources to offload computations and extend their capabilities [12].



(a) Elastic cluster. (b) Super cluster. (c) Brokering cluster.

Fig. 4.6 Three main deployment scenarios for Liquid Computing.

4.3.2 Deployment scenarios

We foresee three high-level deployment scenarios to be mostly enabled and fostered by liquid computing (Figure 4.6).

Elastic cluster

The Liquid Computing paradigm reduces the fragmentation of scattered clusters thanks to the possibility of transparently leveraging the resources available in other locations (Figure 4.6a), which enables it to balance and absorb load spikes (cloud bursting). This applies to edge-computing scenarios, whose pervasiveness is typically achieved at the expense of computational capacity but is visible also in more traditional cloud contexts where multiple clusters are active. For example, elastic clusters can be used to support multi-cloud strategies (no single vendor lock-in) and geographically distributed multi-cluster deployments (e.g., due to cost optimization, latency, redundancy, or legislative concerns). Resource sharing can involve both on-premise and public data centers, hence deploying latency-sensitive applications close to the final users, while benefiting from the virtually infinite computational capacity featured by larger infrastructures for resource-intensive tasks. Thanks to the decentralized approach and the support for peering contracts, resource sharing can occur also when clusters are under the control of different organizations, i.e., they belong to different administrative domains.

Super cluster

We are currently witnessing scenarios in which a company owns hundreds, or even thousands, of small clusters, often located at the edge of the network, such as a large telecom operator. In this context, the Liquid Computing paradigm enables a higher-level, super cluster abstraction, representing the single point of entry that

can transparently deploy and control applications hosted by the entire infrastructure (Figure 4.6b). Although apparently centralized, each level-2 cluster maintains its own local orchestration logic, hence being resilient to network outages and preventing possible synchronization issues arising with relatively high latency WAN links [8–10]. In addition, thanks to the super cluster, the administrative burden is greatly reduced, enabling a borderless orchestration that geographically distributes and controls the tasks from a single point of entry, without the necessity of an explicit interaction with the underlying clusters. This scenario facilitates the automatic migration of applications from one cluster to another, which helps when dealing with disaster recovery, infrastructure interventions, scaling, or placement optimization. In addition, it greatly simplifies the replication of jobs across clusters (hence management, monitoring and troubleshooting) as it leverages existing primitives that operate on nodes of the "super cluster"; for example, an operator can easily replicate the same service on a subset of its edge clusters to serve all the end users present in their close vicinity. Finally, this approach can be combined with the elastic cluster for increased dynamism, thus benefiting from the single point of entry for workloads replication, while enabling at the same time the offloading of "bursty" workloads from the edge to the cloud.

Brokering cluster

Complex applications require complex infrastructural setups, accounting for both resiliency and performance. While larger cloud providers could theoretically offer a sufficiently wide catalog of services to satisfy most demands, relying on a single vendor increases lock-in and potentially leads to cost inefficiencies. In this context, we believe that Liquid Computing could foster the creation of new Resource and Service Exchange Points (RXPs), with third party entities behaving as brokers between consumers and providers (Figure 4.6c). Consumers would then only need to peer (both technologically and contractually) with a single RXP to immediately benefit from the entire set of resources (cloud and edge data centers, ...) and ready-to-use services therein offered. This would reduce the complexity and the operational costs, especially for smaller companies, lacking the bargaining power of larger enterprises. Resource providers, at the same time, would be encouraged to participate, to easily reach a wider turnout of interested customers. Additionally, even small actors, such as the ones operating at the edge of the network, would be enabled to

participate in the edge-cloud market, possibly in a fair competition with far larger giants that may not have enough resources to serve a given geographical area, in a way that looks like the energy market in which millions of tiny producers are aggregated by larger buyers.

4.3.3 Liquid Computing vs. current computing continuum

Given the above consideration, the computing continuum envisioned by FLUIDOS (a.k.a., Liquid Computing) is not simply the capability to deploy services in multiple sites datacenter clusters or devices. It defines a virtual space, spanning across multiple technological domains (e.g., one cluster on-prem, another running on the cloud; or one single device, and a cluster running on edge servers; etc.) and, potentially, across multiple administrative boundaries (e.g., two mobile edge operators sharing their resources), in which the three properties presented above hold: Deployment transparency, Communication transparency, and Resource availability transparency, with the advantages presented above.

Obviously, the creation and the maintenance of the liquid continuum has a cost, e.g., in terms of computing (additional software services running on the nodes) and communication overhead (exchanging messages to synchronize data in the different entities). FLUIDOS has the ambition to determine the best trade-off between the advantages brought by the Liquid Computing (e.g., in terms of better utilization of available resources) and the corresponding costs, which could be used to determine where to put the boundaries of the computing continuum, e.g., at the edge of the infrastructure.

4.3.4 The "Liquid Computing" pillars

In our opinion, Kubernetes represents a key enabler for Liquid Computing, thanks to its widespread diffusion in data centers of any size [76], as well as the support for single devices and IoT computing by means of lightweight distributions, such as K3s. To this end, the recent Microsoft's backed Akri project [77] goes even further, introducing an abstraction layer to dynamically interconnect to this platform the variety of sensors, controllers, and MicroController Unit (MCU) class devices typically present at the very edge of the network. At the same time, Kubernetes can

be easily extended through both custom APIs and logic, allowing to transparently integrate liquid-computing related aspects, as well as to semantically enrich the ecosystem and introduce new services that may be shared with peered clusters. Hence, Kubernetes can become the underlying platform the resource continuum is built upon, and it is conceptually similar to an overarching operating system. Still, the key concepts are more general and can be applied with no difference to other orchestrators, such as OpenStack, or even to a mix thereof. Several of the mechanisms described below build upon the Liqo multi-cluster framework presented in Section 3.1.3. Here, we focus on how these capabilities are extended and composed to realize the Liquid Computing vision.

Dynamic discovery and peering

The first key enabler is the discovery and peering function. It fosters the decentralized governance approach typical of a P2P architecture, preventing the need for central management entities and full administrative control over the entire infrastructure. Additionally, it is responsible for the Liquid Computing dynamism, allowing for new peering relationships to be established and revoked at any time, compared to the manual coordination required by static federation approaches. In this context, we define peering a unidirectional resource and service consumption relationship, with one party (i.e., the consumer) granted the capability to offload tasks and/or consume services in a remote cluster (i.e., the provider), but not vice versa. This allows for maximum flexibility in asymmetric setups, while transparently supporting bidirectional peerings through their combination. Overall, this module deals with four main tasks:

1. **Discovery**: to identify candidate clusters to peer with, including large enterprise domains, as well as possibly local independent appliances (e.g., IoT devices).

2. **Authentication**: given the list of feasible candidates obtained during the previous step, optionally filtered through user-configured criteria, it is responsible for the establishment of a secure communication channel with each selected counterpart. Still, resource offloading is not yet possible at this point, being the granted authorizations related to peering establishment steps only.

3. **Resource negotiation**: involving the exchange of request and offer messages to identify the shortlist of clusters selected for resource offloading. The entire process is policy-driven, with decision modules local to each cluster determining at each step whether to proceed with the negotiation or to abort the process. As a representative example, an offering cluster might implement complex business logic to determine the appropriate prices based on current demands and available resources, accounting for resource brokering and reselling scenarios. Consumers, on the other hand, may filter and rank the received offers by means of appropriate criteria, possibly including compliance with the request constraints, cost, additional attributes, past experience, and more. The negotiation process culminates with the mutual agreement between a consumer and a provider. To this end, we envisage the adoption of smart contracts [78] to formalize the exchange in terms of money and resources, especially in the case of inter-administrative domain peerings.

4. **Peering finalization**: once resource negotiation is completed, the peering relationship needs to be finalized, leading to the exchange of the preparatory parameters required for subsequent computation offloading, as well as the setup of isolation mechanisms and the granting of the suitable permissions in the target cluster.

Hierarchical resource continuum

Once peering relationships have been established towards one or more targets, the local cluster gains logical access to remote resource slices. Yet, these need to be properly exposed for application offloading through a continuum abstraction, while respecting the limited knowledge propagation and the multi-ownership constraints. Moreover, we consider API transparency to be of utmost importance to foster its widespread adoption, thanks to the introduction of no disruption in well-established deployment and administration practices, as well as the immediate support for existing management solutions. Being traditional clusters composed of multiple nodes, each one mapping to a physical or virtual server, we propose to represent peered clusters through local, virtual, big nodes. Local, as attached to the consuming cluster; virtual, since they abstract a set of remote resources possibly unrelated from the underlying hardware; and big, being potentially much larger than classical nodes (in terms of available capabilities), as backed by an entire data center slice. The

node concept perfectly complies with the requirement of sharing limited information, hence abstracting peered clusters only in terms of the aggregated resources currently being shared, with no additional details regarding its actual internal configuration. At the same time, it leads to overall better scalability, given the reduced amount of data synced among different clusters. This approach opens up for two possible cluster models. First, extended clusters, encompassing a combination of traditional physical Kubernetes nodes (i.e., workers), and virtual ones. This could be suitable for the resource optimization and RXP consumer use-cases, to allow borrowing external computational capacity to overcome local limitations. Second, virtual clusters, characterized by the absence of local workers. Combining only virtual nodes, they provide a single point of control abstraction to simplify the deployment of applications on user-defined slices of the underlying infrastructure. Moreover, they represent a key enabler for resource brokering, aggregating the resources offered by multiple providers (each one mapped to a virtual node) for reselling.

The virtual node abstraction leads the underlying orchestration platform (e.g., Kubernetes) to consider the above nodes as valid scheduling targets, hence allowing traditional workloads to be transparently assigned to remote clusters. No differences are perceived by the final users, who simply benefit from the larger number of available resources. This approach brings to a hierarchical representation of the resource continuum.

When a new workload is deployed in the local cluster, the scheduler first selects the optimal node for its execution. Then, if the target is a virtual node, the workload is remapped to the corresponding remote cluster, where a second scheduling round is started to identify the physical server where it will be executed upon. While considering a two-layer scheduling in this example, the approach can easily generalize to multiple levels if needed, depending on the number of virtual node redirections. Hence, allowing scheduling decisions to occur at different abstraction layers, reducing the overall number of feasible candidates to consider at each step and potentially increasing the resulting accuracy. Once more, compliance with standard Kubernetes APIs enables vanilla schedulers to deal out-of-the-box with extended clusters. However, custom scheduling logic might be appropriate in certain scenarios, allowing for further optimizations thanks to the knowledge about the semantics of the peering relationship (e.g., monetary costs, network characteristics, geographical distance, QoS). In both cases, end-users can easily enforce domain-specific constraints through Kubernetes standard high-level policies (i.e., selectors

and affinities) to assign workloads to slices of nodes and ensure replicas spreading. Hence, sticking to an intent-driven approach, while requiring no modifications in standard application deployment workflows.

Resource and service reflection

Each virtual node is responsible for its allocated workloads, whose execution is delegated to the remote cluster. Hence, selected control plane information should be present both in the local cluster (required to fulfill the requirement of the virtual node abstraction) and in the remote cluster (enabling the remote-control plane to carry out its operations). This introduces the resource reflection concept: objects exist both in their native form (i.e., in the local cluster), and in their shadow form, remotely. Indeed, applications most likely require accessory artifacts for proper execution (e.g., configurations, authorization tokens, network endpoints, etc.), which then need to be reflected in the target cluster. The resource reflection logic enforces the transparent realignment between the two digital twins of the same artifact across the different domains, while ensuring the desired information opacity properties (i.e., omitting or masquerading data that should not be propagated) and resolving possible conflicts which may arise in the remote infrastructure (e.g., naming collisions, different underlying technologies, etc.). Overall, it shall support the propagation of both local modifications (*outgoing reflection*, e.g., the change in a user configuration) and of remote status changes (*incoming reflection*, e.g., an application is being restarted due to a crash), hence allowing for proper inspection. Service endpoints represent one of the most important reflected information, enabling an application running on one cluster to be reachable (hence, consumable) from another cluster. This may require the close coordination of the network fabric to disambiguate and transparently translate possible overlapped network addresses used in the communication flows.

The clever reflection of the required information is the key to achieve objectives such as robustness, enabling clusters to evolve also in case of network disconnections, and scalability, reducing the amount of synced data.

Network continuum

According to the virtual nodes approach, different components of the same application may be spread across multiple clusters. Still, the resource continuum, alone,

is not sufficient to ensure their correct execution, as the various microservices most likely need to interact among each other. Orchestration platforms typically implement internal communication by means of private IP addresses, resorting to public ones only for user-facing services. Hence, they are unsuitable for direct (pod-to-pod) cross-cluster interactions and require the introduction of an appropriate network fabric responsible for the transparent communication between microservices, no matter where they are executed. Accounting for the decentralized and dynamic approach fostered by Liquid Computing, with peers possibly joining and leaving at any time, the network fabric cannot rely on ahead-of-time knowledge for its establishment. Indeed, it shall only require the cooperation between the two involved clusters, which negotiate the configuration parameters necessary to set up the secured communication channel and the proper mechanisms to guarantee any-to-any communication across the entire virtual cluster. Being the interconnecting clusters potentially under the control of different administrative domains, it is likely conflicts may arise, e.g., in terms of overlapping IP addresses or underlying networking solutions. The network fabric is expected to transparently handle all these issues, while virtually extending the local cluster network to the entire resource continuum, presenting a unique border-less addressing space. Supposing a central cluster C peered with N others, we foresee two main network topologies for data plane communications. First, a hub and spoke topology, with n direct interconnections between C and all the leaves. Conceptually simple, this solution requires all traffic between applications residing on peripheral clusters to flow through the central hub, potentially resulting in communication inefficiencies. Still, it may be appropriate when applications do not span across multiple remote clusters (e.g., the same application is replicated in multiple edge clusters), in case either the communication pattern or the underlying network match the star topology, as well as when traffic policies should be enforced from a single point of control. Second, an opportunistic mesh topology, providing full connectivity between all clusters hosting applications potentially communicating between one another, to avoid traffic tromboning. It is worth noting that peripheral clusters may in turn play the role of central clusters for different peering sessions, hence leading to completely dynamic and independent topologies, and potentially overlapped virtual clusters.

Storage and data continuum

When an application is spread across multiple clusters, stateful workloads require the access to persistent storage locations, which implies a data continuum across the virtual cluster. To this end, we foster the data gravity approach borrowed from well-established practice in the Big Data world [79]. According to it, data attracts the associated workloads (i.e., introducing additional placement constraints) rather than vice versa, to ensure the best performance in terms of reduced network traffic and latency, as well as to enforce storage locality, which represents a strong requirement to comply with law regulations. This paradigm allows also for the extension of traditional in-cluster stateful workloads replication mechanisms (e.g., databases) across the entire resource continuum, transparently achieving increased disaster recovery support. Information replication and synchronization might be further supported if more dynamism is desired, although requiring the exchange of data between potentially distant storage pools; hence, this is mostly suitable only in case of limited amounts of data.

4.4 FLUIDOS architecture

The FLUIDOS software stack is composed of four main layers, as shown in Figure 4.7):

1. Vanilla operating systems abstract the underlying hardware capabilities. Currently, FLUIDOS is compatible with all major Linux distributions, Robot Operating System (ROS), as well as some embedded OSs (e.g., OpenWRT).

2. Kubernetes supports workload execution and introduces a uniform layer on top of different infrastructures.

3. Liqo, which brings in a multi-cluster abstraction on top of Kubernetes, enables seamless offloading of workloads and transparent communication from one cluster to another. At the same time, it provides the three computing pillars described previously.

4. FLUIDOS, which implements the other required Meta-OS capabilities.

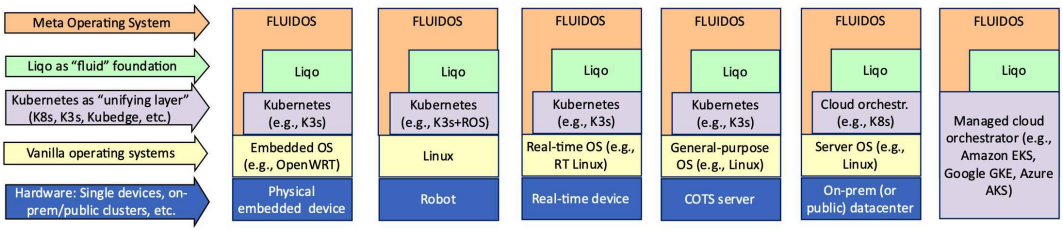


Fig. 4.7 The FLUIDOS software stack.

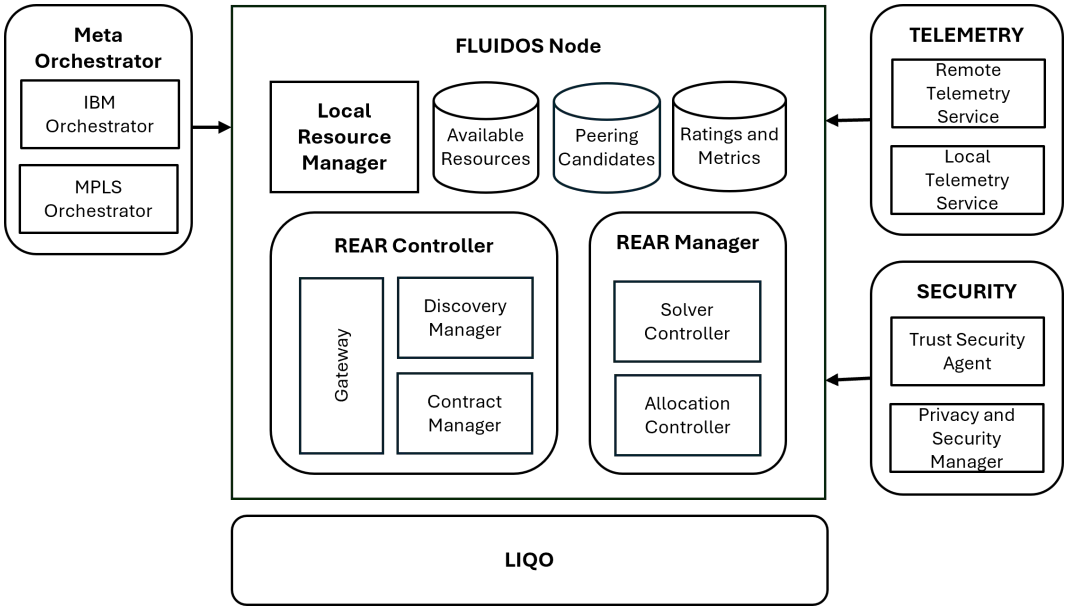


Fig. 4.8 FLUIDOS node core components.

FLUIDOS's main component is the *node* (Figure 4.8), which represents the abstraction of a set of compute resources in the continuum. Specifically, a FLUIDOS node may abstract a Kubernetes cluster with multiple worker nodes or degenerate to a single worker node. FLUIDOS nodes can interact *vertically* and/or *horizontally*. The vertical interaction introduces aggregation and hierarchical scaling into the picture. A FLUIDOS *domain* can be created by aggregating multiple nodes, hence exporting a virtual space that is the union of the resources (and services) of the composing nodes. On the other hand, the horizontal interaction enables the creation of a "stretched" FLUIDOS domain among peers, which can share their resources and services, or part of them, based upon a set of policies.

FLUIDOS nodes communicate with each other using REAR [21], a novel protocol designed to enable entities within the continuum to advertise their resources

and facilitate the reservation of those assets by consumers or applications. Currently, five different classes of assets are supported: Kubernetes slices (i.e., a slice of computing resources of a Kubernetes cluster), VMs, Services (i.e., implementing the so-called Software-as-a-Service), Data, and Sensors. Building upon the negotiation with the REAR protocol, FLUIDOS then enforces the creation and management of the agreed-upon resources in the node. FLUIDOS nodes then rely on the *Network Manager* to discover other FLUIDOS nodes and maintain their metadata. This discovery mechanism ensures that nodes can identify potential resource providers, allowing them to participate effectively in the REAR advertisement and reservation. The *Network Manager* currently supports two technologies: multicast and based on Distributed Hash Table (DHT)[10]. The former targets Local Area Networks (LANs), enabling the discovery of nodes connected to the same physical network, e.g., robots or networks of drones connected to the same 5G cell. In contrast, the latter targets WANs, allowing FLUIDOS to support geographically distributed infrastructures.

All this serves as an enabling technology for the *meta-orchestrator* [80], the FLUIDOS component in charge of dynamically [81] adjusting the shape of the local cluster, acquiring resources from other FLUIDOS nodes, in response to the submitted workload from the user. Specifically, the meta-orchestrator receives requests to deploy an application in the form of *intents*, which specify user requirements for the execution of the application. The intents allow to specify requirements beyond what is traditionally supported by Kubernetes, allowing the specification of performance characteristics (such as latency, throughput or available bandwidth between endpoints), security and compliance, and energy efficiency or carbon emission constraints. Upon receiving application definition and associated intents, the meta-orchestrator will identify a *template* [82] of resource suitable for executing such workload in accordance with the user-defined constraints. The meta-orchestrator will then trigger the REAR protocol to identify the most suitable cluster to satisfy the user intent.

Finally, once resource negotiation is complete, a dedicated telemetry service continuously monitors whether the Quality of Experience (QoE) of the exchanged resources aligns with the agreed-upon transaction. Simultaneously, a security manager enforces additional security measures to ensure workload isolation and compliance with predefined security policies.

[10]https://gitlab.com/pi-lar/neuropil

Note: the entire codebase of FLUIDOS is Open Source and available on GitHub[11].

4.4.1 Research challenges

We now outline the major research challenges steaming from the FLUIDOS Liquid Computing, bridging the gap with known challenges [83].

Decentralization. The computing continuum inherently involves highly decentralized infrastructures. As a result, each entity in the continuum must be able to autonomously decide how to shape the boundaries of its own continuum. FLUIDOS leverages decentralized orchestration strategies by combining the REAR protocol and the meta-orchestrator to enforce a P2P-based interaction for resource management. However, a centralized approach is still supported through the vertical interaction previously discussed.

Data privacy and sovereignty. As data flows across heterogeneous domains, ensuring data privacy and sovereignty becomes paramount. With regulations like GDPR enforcing strict control over data crossing geographic or administrative boundaries, the need for well-defined privacy policies to govern data is becoming increasingly critical. To this end, resources negotiated with the REAR protocol embed information on compliance with certain regulations. Therefore, upon purchasing resources from other entities, clients know whether the processing and the corresponding data are compliant with the requested regulations.

It should be clarified that compliance information embedded in REAR resource advertisements is currently declarative: the provider self-attests the regulations it complies with (e.g., GDPR), and the consumer's decision to purchase relies on this declaration combined with the provider's authenticated identity (Section 4.2.6), rather than on independent, automated verification of actual compliance. Mechanisms for runtime verification or third-party attestation of declared compliance (for instance, leveraging TEEs to attest that data processing occurred under the declared constraints) are identified as future work and discussed further in Section 6.4.

Cross-domain authentication and authorization. The continuum involves interactions between multiple administrative domains, making authentication and

[11] https://github.com/fluidos-project

authorization complex. To this end, FLUIDOS introduces a distributed authentication/authorization approach based on DLT to validate the resources available in the continuum. In addition, FLUIDOS is based on Self-Sovereign Identity (SSI) dentity Management (IdM) where data isolation is achieved by placing control of data through VCs, which allow entities in the continuum to selectively share specific pieces of information without revealing their entire identity or exposing sensitive data. As a result, this approach provides runtime security by protecting against unauthorized access, data breaches, and manipulation of credentials and interactions during runtime. The strong source of trust provided by the DLT is used to further minimize risks in adversarial environments, as the policies requested for authorization are verifiable during runtime and can be checked by all participants.

Extension beyond Kubernetes. While Kubernetes is a dominant solution for edge/cloud container orchestration, it faces limitations at the far edge, where heterogeneous and resource-constrained devices are prevalent. FLUIDOS leverages KubeEdge in these cases, with the capability to share services and support transparent communications also in the above environment. In fact, KubeEdge allows to run containerized applications also on resource-constrained devices, natively supporting MQTT and HTTP-based communications.

4.5 Experimental evaluation

FLUIDOS streamlines the discovery of available resources in the continuum space and the consequent exploitation. In the following, we first evaluate the time required to identify suitable candidate nodes in the continuum (Section 4.5.2), moving then to the evaluation of the REAR protocol, focusing on the time required to list all the available resources in the continuum (Section 4.5.3). Finally, a potential usage scenario for leveraging the newly acquired resources is presented (Section 4.5.4).

This section addresses the following research questions:

RQ1. How quickly can a FLUIDOS node discover peers in the continuum, and how does this scale with the number of nodes and network packet loss (Section 4.5.2)?

RQ2. What is the protocol-level overhead introduced by REAR's three-stage resource discovery workflow (Section 4.5.3)?

RQ3. What latency does application offloading via Liqo introduce relative to local execution (Section 4.5.4)?

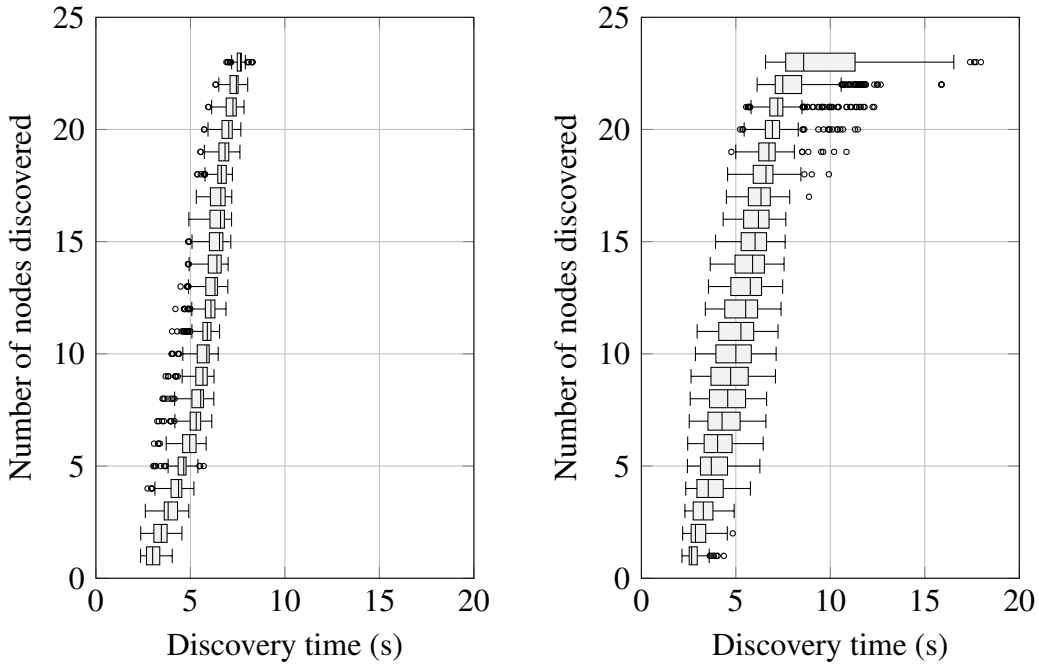
4.5.1 Experimental setup

The measurements reported in this section were collected on a testbed of FLUIDOS nodes deployed as containerized instances on a cluster of commodity servers, all connected to the same local-area network and placed under a single administrative domain. Network conditions (e.g., the 10% packet loss scenario in Section 4.5.2) were emulated using standard Linux traffic-control utilities rather than observed on a real wide-area deployment. This setup is representative of a single-operator, LAN-scale FLUIDOS deployment; it does not capture the WAN-scale, multi-operator conditions (e.g., heterogeneous latency, jitter, and intermittent connectivity across geographically distributed and independently administered sites) that motivate the broader cloud continuum vision discussed in Chapter 2, and the results should be interpreted accordingly. A DHT-based discovery evaluation under WAN-scale, multi-domain conditions is identified as future work in Section 6.4.

4.5.2 Peer discovery time

The peer discovery time is defined as the time required for a FLUIDOS node to identify all available nodes in the continuum. As described in the previous section (Section 4.4), the discovery process is managed by the *Network Manager*, which supports two discovery modes: multicast-based and DHT-based discovery. The multicast-based allows for LANs discovery, enabling the discovery of nodes connected to the same subnet. This scenario is typical in environments such as industrial premises (e.g., robots) or networks of drones connected to a common 5G antenna. In contrast, the DHT-based extends the discovery capabilities to WANs, allowing FLUIDOS to support geographically distributed infrastructures. In the following, we will focus on the multicast-based discovery process.

The discovery time data is collected by measuring the time required for a single node to identify peers in the FLUIDOS continuum. Figure 4.9 illustrates the relationship between discovery time and the number of participating nodes under two conditions: an ideal communication channel (i.e., no packet loss) and a channel with



(a) Discovery time for 20 nodes with no packet loss.

(b) Discovery time for 20 nodes with 10% packet loss.

Fig. 4.9 Network manager discovery times increasing the number of nodes in the continuum.

10% packet loss[12], since multicast is implemented using User Datagram Protocol (UDP). Starting from a fleet of 24 initial nodes, discovery time is measured as the time required by a single node to discover the first N nodes, where N varies between 1 and 23. The order of discovery is not considered relevant. Without packet loss (Figure 4.9a), a linear correlation emerges, with the shortest discovery time recorded as 3 seconds for a two-node system (i.e., discovering and discovered nodes) and approximately 8 seconds to discover all 23 nodes. Overall, discovery time remains consistent. With packet loss (Figure 4.9b), a shorter discovery time is initially observed due to the reduction in the number of packets to process. However, as more nodes need to be discovered, the process is increasingly affected by retransmission delays. Despite this, performance degradation remains minimal, with outliers appearing primarily in larger infrastructures. Additionally, FLUIDOS allows nodes

[12]A packet loss rate of 10% was chosen as representative of a moderately degraded wireless or wide-area network link. Severe enough to meaningfully stress-test the discovery mechanism's resilience, while remaining within a range where the underlying network is still considered operational rather than failed.

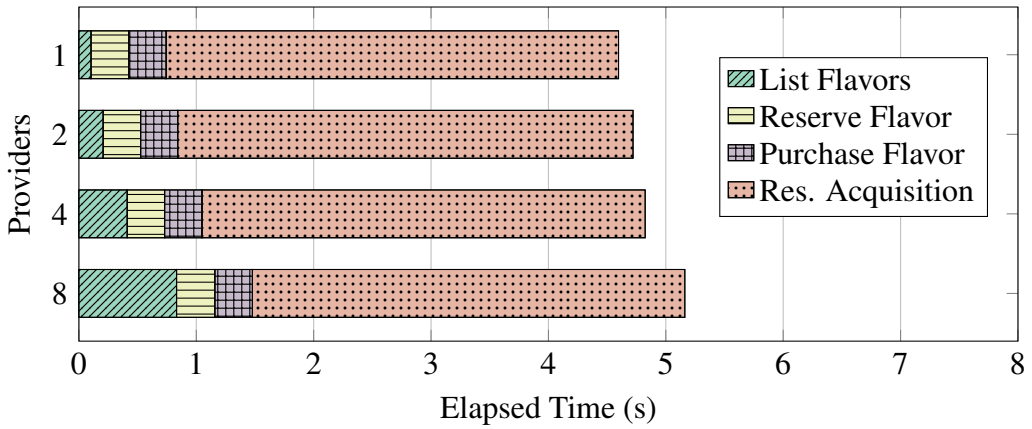


Fig. 4.10 REAR stages' timing vs. time to ready purchased resources.

to act as intermediate brokers, consolidating resources and reducing the apparent cardinality of the infrastructure.

4.5.3 REAR resource discovery

We here report the empirical evaluation of REAR by means of the testbed created for FLUIDOS[13]. In the testbed, we collect metrics from the three different stages of the REAR protocol, namely, *List Flavors*, *Reserve Flavor*, and *Purchase Flavor* to provide a breakdown of the overhead, varying the number of providers offering flavors of type K8Slice. The *Resource Acquisition* phase represents instead the reference time required to extend the pool of locally available resources in Kubernetes using the Liqo framework.

Figure 4.10 details the elapsed time for a generic consumer before the purchased resources are available and ready to use. In the consumer, the list of REAR-enabled providers is currently manually configured, but it will be discovered through third-party brokering services in future releases. As we can see, the *List Flavor* message is influenced by the number of available providers, leading to higher latencies when the consumer has an extensive list of available providers. In the worst-case scenario, the consumer must iterate the entire providers' list before identifying a suitable resource, resulting in a linear time complexity $\theta(n)$ w.r.t. the provider list size n. However, the *Reserve Flavor* and *Purchase Flavor* messages maintain stability through the different scenarios. It is worth noticing that despite any fluctuations, the REAR

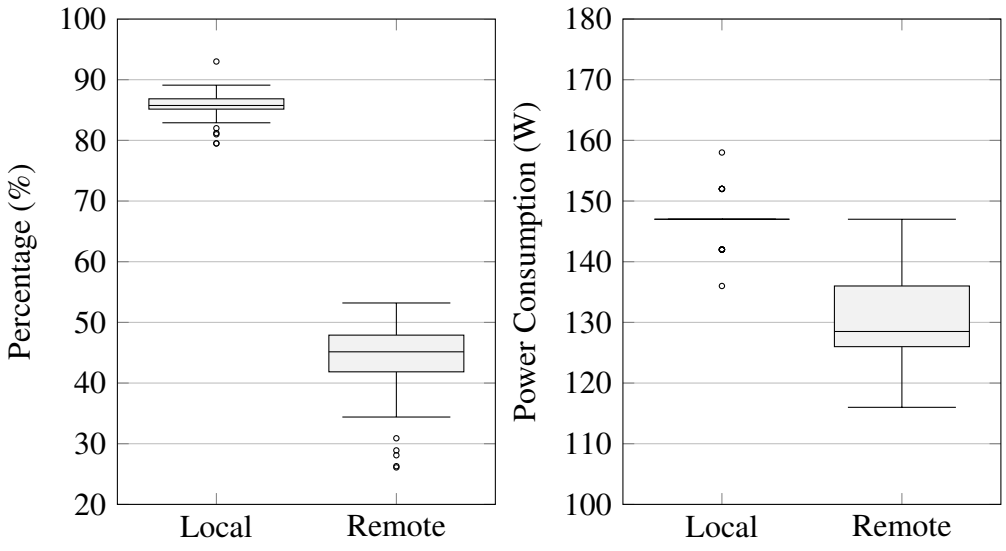
[13]https://github.com/fluidos-project/node

protocol's overhead remains minimal (approximately 30% of the total time in the worst case), and can be further reduced by aggregating resources through third-party brokering and aggregation services to create the "unique pool of resources" enabled by the continuum.

All providers and the consumer in this evaluation run as Kubernetes clusters within the same testbed described above, communicating over the LAN. The number of providers was varied synthetically while keeping network latency between consumer and providers negligible and uniform. The evaluation therefore isolates the protocol-level overhead of REAR's three stages from network-induced variability, but does not yet characterize REAR's behavior when providers expose heterogeneous resource types (e.g., a mix of K8Slice, VM, Sensor, and Service flavors) or are reachable over heterogeneous, higher-latency links, both of which are left for future evaluation. The present evaluation focuses on K8Slice flavors, as these are the resource type most directly exercised by the FLUIDOS/Liqo integration used in the testbed. Extending the experimental evaluation to VM, Sensor, and Service flavors (each of which follows a different acquisition and lifecycle model, as described in Section 4.2.4) is an open item for future validation of REAR's generality across the full resource model. This scenario, a single consumer negotiating with a varying number of providers exposing K8Slice resources, was chosen as the minimal configuration needed to isolate REAR's protocol-level overhead (List/Reserve/Purchase) from confounding factors such as network heterogeneity or resource-type diversity. It is representative of REAR's expected common-case usage, a consumer evaluating multiple candidate providers before committing to one.

4.5.4 Application offloading

Once FLUIDOS nodes become aware of the surrounding environment in the continuum, enhanced policies can be enforced to select the best candidate to peer with (using FLUIDOS terminology), based on the requested intent. This test leverages a FLUIDOS use case, in which a battery-constrained robot might dynamically offload part of the computation to nearby edge nodes or another (inactive/charging) robot while moving, hence reducing the energy drained from the local battery. The offloadable application consists of two components: navigation and obstacle detection. The navigation component is responsible for computing the path that allows the robot



(a) Robot CPU usage.

(b) Power consumption.

Fig. 4.11 Robot dynamics when only local processing is permitted, and when the offloading is enabled.

to autonomously travel from point A to point B. The obstacle detection component analyzes the camera feed to recognize and avoid obstacles during navigation.

The battery data is obtained by reading the output of the robot's Battery Management System (BMS), which measures the power consumption of the entire system, including sensors and actuators (such as wheel motors). Figure 4.11 depicts the resource usage for a single robot in case only local onboard processing is possible, and when the FLUIDOS-enabled Liquid Computing allows for task offloading on nearby nodes. Specifically, the left side represents the CPU usage measured on the robot, whereas the right side shows the resulting power consumption. When the robot is in offloading mode, the CPU load is almost halved: the motion planning is executed remotely, while the logic required to interact with sensors and actuators cannot be moved and hence still runs locally. The reduction in CPU usage also results in a non-negligible decrease in the power consumption for the entire robot ($\sim 15\%$), given that other components (e.g., wheels, etc.) continue to drain power as usual. This significantly increases the duration of the battery and, consequently, the overall operation time.

4.5.5 Discussion of results

The results in this section show that (i) FLUIDOS peer discovery completes within a practical time window even under moderate packet loss, though scaling has only been validated at LAN scale and under single-domain conditions (Section 4.5.2); (ii) REAR's protocol-level overhead remains modest for K8Slice resources under negligible network latency, though heterogeneous resource types and higher-latency links remain to be evaluated (Section 4.5.3); and (iii) Liqo-based application offloading introduces latency overhead that is small relative to typical application response-time budgets (Section 4.5.4). Taken together, these results support the feasibility of the proposed mechanisms under controlled conditions, while highlighting WAN-scale, multi-domain, and heterogeneous-resource evaluation as the natural next steps, as discussed in Section 6.4.

4.6 Concluding remarks

This chapter introduced the computing continuum as a solution to the limitations of traditional siloed infrastructures, highlighting the need for seamless integration across heterogeneous computing layers. The proposed FLUIDOS framework, built upon Kubernetes and Liqo, enables deployment, communication, and resource availability transparency, forming the foundation for a decentralized meta-operating system. Through its intent-based resource negotiation and dynamic orchestration capabilities, FLUIDOS enhances the efficiency and flexibility of distributed computing, allowing workloads to be placed optimally based on application-specific requirements. The experimental evaluation demonstrates that this approach not only improves resource utilization but also reduces operational overhead, paving the way for a more adaptive and scalable infrastructure. However, while this work establishes a solid foundation, several challenges remain open. Future research should focus on optimizing real-time resource discovery and integrating AI-driven scheduling strategies. Furthermore, improving interoperability with other cloud and edge frameworks will be essential to fully realize the vision of a unified and dynamic computing continuum. One challenge that deserves particular attention concerns data governance. As discussed in Section 4.3.4, the storage and data continuum fosters a data-gravity approach in which workloads are attracted toward the data they consume, rather than requiring expensive data transfers across the infrastructure. However, when the computing

continuum spans multiple administrative domains, the question of who controls the data, under what conditions it can be accessed, and how to prevent unauthorized exfiltration becomes critical. Regulations such as GDPR impose strict constraints on data crossing geographic or administrative boundaries, and the multi-ownership nature of the Liquid Computing paradigm amplifies these concerns. The REAR protocol embeds compliance metadata in resource advertisements (Section 4.4.1), but the enforcement of data sovereignty and the creation of trusted environments for cross-domain data sharing require dedicated architectural mechanisms. The following chapter addresses this problem by proposing infrastructure-level data spaces that leverage the multi-cluster capabilities of Liqo to enable controlled data consumption while preserving data sovereignty.

Chapter 5

Data Governance in the Cloud Continuum

Data is a key asset of our modern digital society. The previous chapter introduced a computing continuum that enables workloads to flow across heterogeneous domains, and identified the storage and data continuum as a key architectural component, fostering a data-gravity approach in which processing is moved close to data sources. However, the actors in charge of data production and consumption are often different, posing non-trivial challenges to control who can access what, and (even more important) to guarantee that data is not stolen and/or copied illegally, particularly when sensitive data is concerned. These problems can be analyzed with two concepts: *data sovereignty* and *data gravity*.

Data sovereignty involves, or can be identified with, the control of data flows and the related infrastructure via national jurisdiction [84]. For example, governments are worried about data sovereignty when their information is stored in the cloud and are questioning how to ensure confidentiality and prevent government data from being subject to another jurisdiction if hosted abroad [85]. These concepts can be clearly extended to universities and companies, where data can be hosted outside their facilities. In this context, a mention of the European GDPR has to be made. It imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the European Union[1]. Regarding Europe, it is worth

[1] https://gdpr.eu/what-is-gdpr/

mentioning two additional pieces of legislation, known as the Data Governance Act[2] and the Data Act[3]. The former seeks to increase trust in data sharing, strengthen mechanisms to increase data availability, and overcome technical obstacles to the reuse of data. The latter complements the Data Governance Act by clarifying who can create value from data and under which conditions.

Data gravity is the ability of data to attract applications, services, and other data. Each data chunk has a mass and density, and when these two characteristics become large, it becomes difficult in terms of time, logistics, and cost-effectiveness to move the data from one location to another over the network. It is similar to the scenario of a large physical mass exhibiting gravitational pull to its surrounding objects: the larger the mass, the stronger the attraction [86].

Data spaces allow us to solve those problems by setting up a restricted area in which third parties can consume data and access only what the producer wants to share with them. This chapter aims to demonstrate a cloud-based technology that can dynamically create flexible data spaces upon request, potentially spanning multiple administrative domains, leveraging Liqo. This enables a data producer to offer its data to potential consumers, without giving up on security and data ownership/sovereignty rules, and without affecting the possibility of consumers to read and process arbitrary data. Differently from current data spaces, in which only *communication* (i.e., data transfer) is involved among (data) producer and consumer actors, the proposed solution allows the data consumer cluster to borrow also *processing* resources in the data producer cluster, associating them to its cluster and creating a virtual continuum that spans both. While being more flexible (e.g., data can be processed on the data producer cluster, without requiring long-distance and expensive data movements), it introduces several advantages in terms of data protection. Any service running on the virtual continuum is controlled by the owner of the virtual cluster (hence, data consumer), which therefore has complete control over the lifecycle of its own services, and no control over others that are not owned. On the other side, the data producer does not give up full control of its resources and it may impose additional policies that enable *controlled access* to all the *guest* services running on its domain. If the data producer finds that the data usage deviates from the agreed-upon terms, it retains the right to terminate the partnership. It is

[2]https://digital-strategy.ec.europa.eu/en/policies/data-governance-act

[3]https://digital-strategy.ec.europa.eu/en/library/data-act-proposal-regulation-harmonised-rules-fair-access-and-use-data

noteworthy that the control at this level should be executed in tandem with oversight of the *guest* services. This dual oversight is critical because the data provider needs to ensure that the algorithm being executed within its domain adheres strictly to the terms agreed upon and that no variations from the approved algorithm are being implemented. In the current implementation, detection of usage deviation relies primarily on the data structure validation performed by the Sidecar (Section 5.2.1), which checks that data leaving the consumer's pods conforms to the agreed-upon schema; deviations such as the presence of unexpected fields can be flagged this way. Detecting more subtle misuse, such as a consumer using the data for an undisclosed purpose within the agreed schema, is not currently technically enforced and would require organizational/contractual oversight (e.g., audits) rather than automated detection; this limitation is discussed further in Section 5.2.

Our solution can automatically wrap the application and enforce the defined privacy, security, and access policies, facilitating data producers to offer their data to arbitrary actors, counting on a more solid technical background than simple trust (and legal agreements), hence facilitating the sharing of relevant data among multiple parties. It can dramatically simplify the operations of data producers and consumers (which are often different actors), as well as processing multiple data sources, thanks to a data-gravity approach in which processing is (securely) moved close to data sources[4].

5.1 Data spaces

The International Data Spaces Association (IDSA)[5] aims to create a global standard for International Data Spaces (IDS), as well as to foster technologies and business models that will drive the data economy of the future in Europe and around the globe. The EU-funded Open DEI project [15], which is part of this ecosystem, published a

[4]This work was conducted in collaboration with Leonardo Camiciotti, Francesco Cheinasso, Alessandro Olivero, and Fulvio Risso. It was partly supported by the GEANT Innovation Programme Year 2022, project "Borderless Data Spaces", and the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative. Additionally, I would like to thank Luca Francescato, Marco Iorio, Claudio Pisa, Ferdinando Ricchiuto, and Francesco Torta for their support, comments, and contributions throughout the different stages of this work.

[5]https://internationaldataspaces.org

position paper defining a data space as a decentralized infrastructure for trustworthy data sharing and exchange in data ecosystems based on commonly agreed principles. Users of such data spaces are enabled to access data in a secure, transparent, trusted, easy, and unified fashion. These access and usage rights can only be granted by those persons or organizations who are entitled to dispose of the data.

The IDS standard is adopted by projects like Gaia-X[6], which aims to create an ecosystem whereby data is shared and made available in a trustworthy environment, and employed by Catena-X[7], which can be seen as a data space inside that data ecosystem. Gaia-X defines a data space as a type of data relationship between trusted partners who adhere to the same high-level standards and guidelines in relation to data storage and sharing within one or many vertical ecosystems (i.e. Health, Infrastructure, Tourism, etc.).

5.1.1 Architecture requirements

The International Data Spaces Reference Architecture Model (IDS-RAM)[8] is the beating heart of the IDS: it comprises the standards for secure and sovereign data exchange, certification, and governance across Europe and around the world. Open DEI identified seven architecture requirements that data spaces must fulfill to be considered as such. These requirements include data-sharing empowerment, trustworthiness, publication, economy, interoperability, and data space engineering flexibility, and community. The aim of these measures is to ensure appropriate stakeholder control, security, privacy, data exchange, customization, and community support in data spaces. In Section 5.2, more detailed and lower lever requirements will be explained.

The IDS Connector is the key technical component to achieve secure and trusted exchange of arbitrary data, dealing with the authentication of the entities involved in the data exchange, and the attachment and enforcement of usage policies to data. Moreover, it acts as an application-level gateway (e.g., protocol translator), hence providing a uniform access to any data and by hiding specific protocols used internally towards the actual data source. This enables the exchange of data between

[6]https://gaia-x.eu
[7]https://catena-x.net
[8]https://github.com/International-Data-Spaces-Association/IDS-RAM_4_0

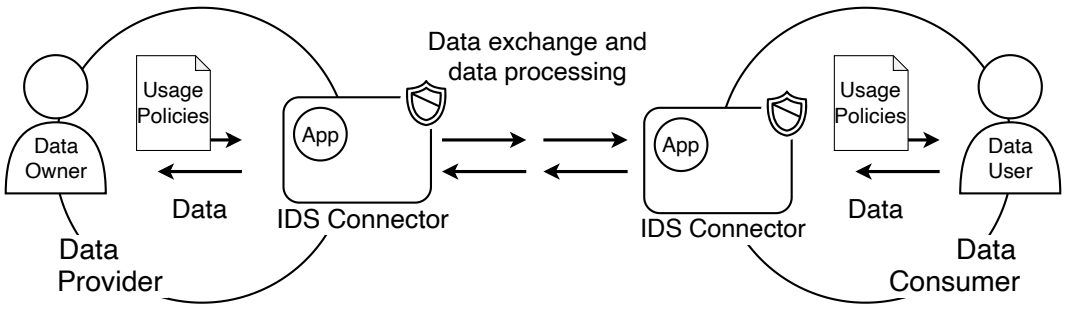


Fig. 5.1 IDSA architecture for data spaces.

clusters while ensuring the enforcement of all necessary security measures and policies.

The connector sends the data directly to the recipient from the device or the database in a trusted, certified data space, so the original data provider always maintains control over the data and sets the conditions for its use, as shown in Figure 5.1[9]. The connector uses technology that puts data inside a sort of *virtual container*, which ensures that it is used only as agreed upon per the terms set by the parties involved (the connector enforces data isolation, restricting data flow according to the agreed-upon policy). IDSA specifies the connector architecture, but then there are several implementations, which are interoperable (i.e. they can communicate with each other seamlessly) thanks to the common standard they adhere to, such as the Eclipse Dataspace Connector[10], TRUsted Engineering (TRUE) Connector[11], and the Dataspace Connector (DSC)[12].

5.2 Architecture

In a traditional interaction between two administratively distinct actors, one acting as data producer and the other as data consumer, each actor controls its own cluster, while (raw) data is transferred between the parties. Each actor has a vested interest in maintaining control over the interactions among data and processing services

[9]https://internationaldataspaces.org

[10]https://projects.eclipse.org/projects/technology.edc

[11]https://www.eng.it/en/case-studies/true-connector-per-facilitare-la-condivisione-di-dati-in-gaiax

[12]https://www.isst.fraunhofer.de/en/business-units/data-business/technologies/Dataspace-Connector.html

in its cluster, with a specific emphasis on the owner of sensitive data, which may give access to sensitive data while, at the same time, preventing unauthorized data exfiltration[13].

To illustrate this concept in a straightforward manner, we consider a scenario depicted in Figure 5.2, with two clusters. A first cluster is owned by a pharmaceutical company (Pharma) that needs to execute its newest algorithms on the patient data, while a second is owned by a hospital (Hospital) that possesses the data to be processed (e.g., medical records). Hospital would like to run the Pharma algorithm on the patient data, but it must prevent that this sensitive data is returned and read by Pharma. Currently, the only solution consists in transferring the data to the Pharma application, and making sure (e.g., through a legal agreement) that data is not stolen. However, no technical methods are available to control that this agreement is actually respected by Pharma.

In our architecture, Pharma seeks to outsource the execution of its workloads to Hospital, while retaining the ability to govern the offloading process using the capabilities provided by Liqo. In other words, Pharma algorithms are running in the Hospital cluster, but those are still in control of the Pharma actor, which handles the lifecycle of this application as it was running locally. On the other hand, Hospital wishes to review offloading requests before accepting or rejecting them. Upon receiving a valid request, Hospital subjects it to a series of additional security measures (e.g., how the offloaded service can communicate with the Pharma cluster; more details in the next Sections), thereby ensuring the secure execution and monitoring of third-party workloads.

The management of these clusters, including their associated complexities, is seamlessly handled by Liqo through standard declarative configurations, thus providing a vanilla Kubernetes experience. Thanks to its capabilities, Liqo simplifies the deployment and management of a robust and secure multi-cluster environment, allowing users to focus solely on the Kubernetes resources required to create the data space playground. However, the proposed solution must adhere to several constraints that establish trust among the involved parties regarding their handling of data and computational resources. These constraints concern network connectivity and privileges restrictions, to prevent pods from bypassing imposed ones. To provide

[13]With data exfiltration we refer to the capability to provide sensitive data to external partners, which has to be consumed by processing services that must produce (and return) only aggregated data, without disclosing all the original details present in the original data.

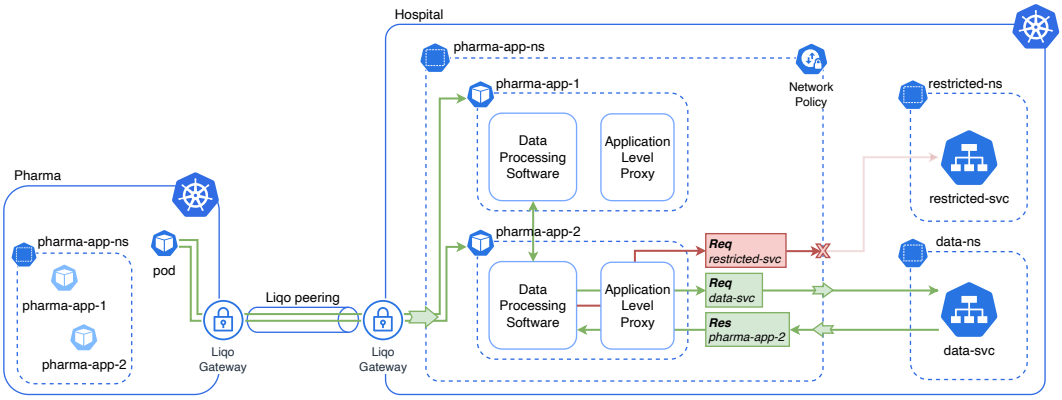


Fig. 5.2 Infrastructure-level data space: deployment example and main communication flows.

a clear understanding of their fulfillment, we present a concise description of these constraints in Section 5.2.1, based on the current capabilities of Liqo at the time of writing.

Building upon the aforementioned scenario, we can further extend it to manage more intricate situations involving multiple data producers and consumers, potentially overlapping in a way where a producer can also act as a consumer when considering other data.

5.2.1 Properties of secure data spaces

Restricted privileges

The offloaded pod must adhere to restrictions set by the hosting cluster and peering can be ceased at any time. The pods need to be started as non-privileged and isolate the host as much as possible in terms of networking and process-level information and data. It is essential to prevent the pods from disabling or bypassing the imposed restrictions, meaning they should not have permission to remove the Sidecar or disable/modify the iptables rules.

Restricted pod-to-pod connectivity

The full pod-to-pod connectivity within the virtual cluster should be limited to avoid offloaded pods to communicate with arbitrary other pods. To restrict pod-to-pod connectivity within a cluster, two components are utilized: Network Policy

and iptables. The Network Policy governs the communication between local and offloaded pods of different entities within the cluster. On the other hand, iptables rules are added to the Liqo Gateway to limit pod interactions by allowing only a select few. While pods of the same entity can interact freely, local and offloaded pods can only communicate with authorized service endpoints, including all endpoints of the reflected service in Liqo's implementation, to ensure the proper functioning of the service.

No network data exfiltration

The offloaded pods must communicate with the external world and within the host cluster in a controlled fashion, adhering to pre-determined data representation models and encryption requirements. A Sidecar (represented as *Application Level Proxy* in Figure 5.2), acting as a proxy, is added to each offloaded pod and it is set by a ConfigMap, ensuring that all the traffic of the pod is forced to go through it. Right now, the only implemented feature is forcing the traffic through the Sidecar, but there is no automatic action enforced to avoid data exfiltration. This is a significant limitation of the current design, not a minor caveat: without automated data-exfiltration prevention, a malicious or compromised consumer pod could in principle copy received data out of the intended boundary through channels the Sidecar does not inspect (Section 5.2.1, Traffic inspection). The current mitigations are limited to schema-based validation of outgoing payloads (Section 5.4) and the organizational trust established during enrolment (Section 4.2.6); neither prevents exfiltration by a consumer that is itself malicious rather than merely careless. Stronger mitigations (such as deep packet inspection, data watermarking/tracing, or confidential-computing-based enforcement within TEEs) are identified as necessary future work and are discussed in Section 6.4. Exclusive communication access is granted only to the offloaded pods, while any communications originating from pods within the data consumer cluster are blocked. Offloaded pods are restricted from communicating with the Internet to prevent bypassing inter-cluster policies.

Traffic inspection

The traffic from offloaded pods to "home" pods need to be inspected to prevent data exfiltration. The Sidecar currently lacks traffic inspection capabilities. However,

it does monitor all requests to and from the application container. Scaling traffic inspection to a large number of concurrent offloaded pods is an open implementation challenge: a production deployment would likely require the Sidecar to apply lightweight, schema-based filtering at line rate (checking outgoing payloads against the expected data structure, as enforced in the Section 5.4 evaluation) rather than deep packet inspection, to keep per-pod overhead bounded as the number of offloaded pods grows.

Service protocols and communications

To prevent the creation of resources that could potentially bypass imposed restrictions, offloaded pods must be restricted from accessing the API server of their original cluster. Currently, it is possible to disable communication to the API server of the original cluster for all offloaded pods, but there is no fine-grained control.

Restricted volume mounting

The offloaded pod must not mount external volumes, to prevent data exfiltration by bypassing controlled communication. Liqo does not currently offer any built-in volume mounting restriction mechanisms.

No code exfiltration

To ensure the security of offloaded pods, they must be protected against image theft and unauthorized image pulling, while also maintaining isolation from the host cluster to prevent inspection, changes, or execution. To mitigate this risk, offloaded pods should be deployed with the *pull always* flag enabled. Additionally, in scenarios where safeguarding against code exfiltration is necessary, it might be advisable to restrict the presence of only the aggregation layer of the algorithm in the remote cluster, while keeping the main algorithm protected. However, Liqo does not currently offer any built-in code protection mechanisms, but it can be solved with orthogonal solutions.

Restricted resource consumption

The cluster that accepts offloaded pods should be able to limit their resource consumption to avoid a negative impact on cluster operations. A cache system, a custom data structure, stores the information of each Liqo peering connection and the defined limits imposed by the data producer. The Validating Webhook is the only component that has access to it, and it is the only one that can make the proper calculations. This system provides constraints that limit aggregate resource consumption at the peering level. It can limit the number of objects that can be created in a remote cluster, as well as the total amount of compute resources that may be consumed by workloads in that cluster.

5.2.2 Workflow

This subsection outlines the process of creating a secure infrastructure-level data space between Pharma and Hospital clusters, enabling data consumption through pod offloading while ensuring the implementation of security measures.

This workflow assumes that the two clusters, Pharma and Hospital, have already been configured and are operational, with Liqo already installed. At this stage, the clusters remain disconnected, and a peering connection must be established between them. Since the Pharma cluster intends to access data in the other cluster, it initiates a Liqo peering towards the Hospital cluster. If the Hospital cluster accepts the connection, it pushes security rules to its Liqo Gateway, granting Pharma access to the Hospital services running in the "data-space" Kubernetes namespace (named *data-ns* in Figure 5.2), which hosts the sensitive data and it is *external* to the Pharma virtual cluster. Within its virtual cluster, Pharma can then read data or offload one or more pods in the Hospital cluster, allowing for data gathering.

When Pharma initiates the offloading process, Hospital detects it and automatically applies a set of security measures to securely host those pods within its cluster. These measures include implementing Kubernetes Network Policies, performing a mutating operation on the offloaded pods, and enforcing firewall rules on the connecting inter-cluster gateway. Offloaded pods are granted access to the protected data, allowing them to manipulate and aggregate the data, thereby adding a new layer to the data producer cluster.

The above security rules ensure that all Pharma pods running on the Pharma cluster can only connect to Pharma pods on the Hospital cluster. Specifically, the set of Network Policies ensures that the offloaded pods can only communicate with selected services while blocking all other requests within the cluster. This is achieved using the standard Kubernetes APIs. The processed data can then be transmitted to Pharma through the Liqo inter-cluster tunnel. The Hospital Liqo Gateway ensures that incoming traffic from the other cluster can only reach its offloaded pods. The process of pod offloading is managed by a Mutating Webhook, which adds an *init* container and a *sidecar* to the main application container, which create the necessary iptables rules to forward all the traffic of offloaded pods to the Sidecar container, which acts as a proxy and monitors all the pod's communications, allowing only the desired ones. This mechanism introduces a strong barrier that safeguards the Hospital cluster against data exfiltration. In fact, this Sidecar intercepts all traffic from offloaded pods and directed outside the Hospital cluster, inspecting the data at the application level through a transparent Application-Level Security Gateway (ALSG) functionality. It enables all Pharma pods running on the Hospital cluster to establish uncensored communications with a list of selected destinations within the Hospital cluster, without ALSG. Finally, it allows service communications from all offloaded Pharma pods to local Kubernetes services in the Hospital cluster, such as Domain Name System (DNS) queries.

It should be noted that the workflow described above automates the technical enforcement of an agreement, not the negotiation of the agreement itself: the underlying inter-organization contract (specifying the purpose, duration, and terms of data sharing) is assumed to be established through a separate, human-mediated legal process before the technical workflow begins. The REAR negotiation (Section 4.2.5) and the data-space connector configuration (Section 5.3) operationalize the already-agreed terms (e.g., translating contractual data-usage restrictions into Sidecar validation rules), rather than replacing the legal decision point. A human-in-the-loop step is therefore implicitly required upstream of the automated workflow, at contract formation time; the present system does not yet support human-in-the-loop intervention during the automated execution phase itself (e.g., to approve individual data transfers), which is left for future work.

5.3 Implementation

As depicted in Figure 5.2, the Liqo Gateway acts as the final destination of the inter-cluster tunnel that has successfully completed the peering phase. Given that the data exfiltration property has to be guaranteed for Hospital, this component (in the Hospital cluster) will deserve a special attention in our architecture. In fact, a new Kubernetes controller (the *gateway controller*) running in the Hospital cluster watches all pods that have been offloaded from the Pharma cluster, leveraging a proper label that is automatically attached by Liqo to all offloaded pods[14]. Following identification, the controller collects their IP addresses and includes them in a whitelist, thereby allowing them to be accessed exclusively by pods belonging to the Pharma cluster (either local or remote), the original offloader, while denying connections from all other clusters. These whitelisted IP addresses are translated into `iptables` rules and pushed on the Liqo gateway of the Hospital. Upon cessation of offloaded pods, their respective IP addresses are removed from the whitelist, and the corresponding `iptables` rules are deleted. Since the Liqo gateway may also act as a NAT between clusters, `iptables` rules are appended outside the secure tunnel, hence are applied after traffic has been decapsulated from the tunnel. Consequently, the rules also lie outside the NAT, thereby making the NAT transparent in our implementation. The IP addresses utilized in the rules are local to the Hospital cluster. The current design provides no specific guarantee that the gateway controller itself has not been compromised; it is trusted as part of the Liqo/Kubernetes control plane on which FLUIDOS relies, with the same trust assumptions as any other cluster-level controller. Extending the gateway controller's deployment with remote attestation (e.g., verifying its binary integrity and runtime environment via a TEE before it is granted access to cross-domain traffic) would meaningfully strengthen this trust assumption and is identified as a promising direction for future work, complementing the TEE-based compliance verification discussed in Section 6.4.

Within the Hospital cluster, another controller, referred to as the *namespace controller*, oversees the reconciliation of offloaded namespaces. Similar to the gateway controller, it identifies offloaded namespaces through labels added by Liqo and appends a Network Policy (NetPol) and a ConfigMap to each identified namespace. Should a namespace be deleted, the associated NetPol and ConfigMap are likewise

[14] It is worth mentioning that the number of offloaded pods can change over time (e.g., new replicas are created), which has to be detected by the above custom controller.

removed. Given that the NetPol is a resource, its deletion ensures the removal of the namespace and all encompassed resources. The NetPol governs the inbound and outbound traffic of the namespace in which it is deployed. It differentiates traffic flow, either from or to namespaces or specific resources, via a specific label. This label, ascribed by the Hospital cluster, allows the Hospital to discern which traffic should be permitted or blocked. If the label corresponds, traffic is allowed; otherwise, it is blocked. Both the sender and receiver must match the label for traffic to flow. Although the label is contained within the NetPol, matching occurs with respect to external resources to the offloaded namespace that intend to exchange traffic with it. The ConfigMap hosts the configuration parameters necessary for directing traffic from each offloaded pod through the Sidecar container.

The final *Mutating Webhook* component performs several operations in the event of pod offloading. An Init container is introduced into the pod prior to the main container when the offloaded pod is scheduled. This container accesses the configuration stored in the namespace ConfigMap and establishes rules that mandate all main container traffic to go through the Sidecar container. Upon the termination of the Init container, the main and Sidecar containers are created. The enforcement by the Init container ensures that traffic from the main container is routed via the Sidecar. As of the current implementation, the Sidecar functions as a traffic monitoring tool, devoid of any metrics, and is built upon the Envoy proxy[15].

It should be noted that the *Gateway Controller* and the *Mutating Webhook* are general-purpose mechanisms for managing offloaded pods across clusters, inherited from Liqo's existing offloading machinery. Their role is not specific to data-space functionality, and they are reused here, with the data-space-specific behavior implemented in the Sidecar and ConfigMap components described above.

5.4 Experimental evaluation

Experimental results have been collected with our custom software running on Liqo v0.8. This version did not include (yet) a proper implementation of the cross-cluster network policies, which had to be carried out by our proof-of-concept implementation.

[15]https://www.envoyproxy.io

The gathered results pertain to the following scenario. Within the Pharma cluster, a pod seeks to access the data supplied by the Hospital cluster. Given the aforementioned workflow has been accomplished, the Pharma cluster has a single pod containing the processing logic, which is offloaded to the Hospital cluster. When the Pharma cluster's pod intends to access data, it must reach out to the offloaded pod to request the desired data. The offloaded pod then communicates with the service exposed by the Hospital within the data space, to which it has been granted access. It collates data, performs an aggregation or analysis, and subsequently sends the processed information back to the requesting pod. All the data sent back to the Pharma pods is inspected and checked by the sidecar, which makes sure no sensitive data is transmitted (e.g., checking the data against a well-defined data structure). In a nutshell, the offloaded pod serves as intermediary, bridging the gap between the Pharma pod and the data provided by the Hospital cluster. This schema-based check is a deliberately lightweight mechanism, and it should be acknowledged as such: it can detect accidental or careless data leakage that does not conform to the expected structure, but it provides no defense against a consumer that deliberately obfuscates or encrypts data before transmission to evade inspection. As with the broader exfiltration limitation discussed in Section 5.2, addressing deliberate evasion would require stronger mechanisms (e.g., content-aware inspection, confidential computing, or restricting egress to pre-approved destinations only) that are left for future work.

This section addresses the following research questions:

RQ1. How much time does establishing a data space between two organizations require (Section 5.4.2)?

RQ2. What resource overhead does the data-space connector (Sidecar, ConfigMap) introduce (Section 5.4.3)? 9 **RQ3.** What latency does data-space-mediated communication add relative to direct, unmediated communication (Section 5.4.4)?

5.4.1 Experimental setup

The Pharma and Hospital clusters used in this evaluation were deployed as two separate Kubernetes clusters hosted on the same physical host as separate VMs. Each measurement in Table 5.1 was repeated 100 times, with reported values representing mean and standard deviation. As with the FLUIDOS evaluations above, this setup

Table 5.1 Pod deployment time.

#Offloaded Pods	Vanilla (s)	Data Space (s)
1	0.095 ± 0.021	0.09 ± 0.023
5	0.240 ± 0.069	0.214 ± 0.077
10	1.214 ± 0.038	1.218 ± 0.038
100	32.794 ± 6.346	31.945 ± 3.368

reflects a controlled, low-latency environment rather than the higher-latency, cross-organization conditions a production data-space deployment would encounter. The relative overhead of the Init/Sidecar mechanism is expected to remain small under such conditions, since the added containers affect only local pod startup rather than network transit time, though this has not yet been empirically verified.

5.4.2 Data space creation time

The standard, or *vanilla*, execution of Liqo requires the establishment of a point-to-point connection between the clusters. Subsequently, a namespace is offloaded, and we will hereinafter refer to it as *offloaded namespace*. Within this offloaded namespace, all deployed resources can benefit from the offloading feature of Liqo. Contrarily, the resources deployed within other namespaces, which are not subject to offloading, are unable to benefit from it.

Our solution adheres to the same two-phase process: peering and namespace offloading. Accordingly, the reported measurements are applicable to both implementations, obviating the need for comparative analysis. The peer-to-peer connection requires a time duration of (3.8098 ± 0.7780)s, and the namespace offloading necessitates (0.3097 ± 0.0738)s. A cursory glance at these values reveals that the peering phase is the predominant consumer of time.

The third phase pertains to the deployment phase where we assessed the deployment of several pods in both the standard Liqo setup and our proposed solution. These findings are encapsulated in Table 5.1. Upon scrutiny, it becomes evident that the results for the two scenarios bear a remarkable similarity. This observation allows us to infer that the inclusion of Init and Sidecar containers in our proposed solution does not significantly influence the time required for deployment.

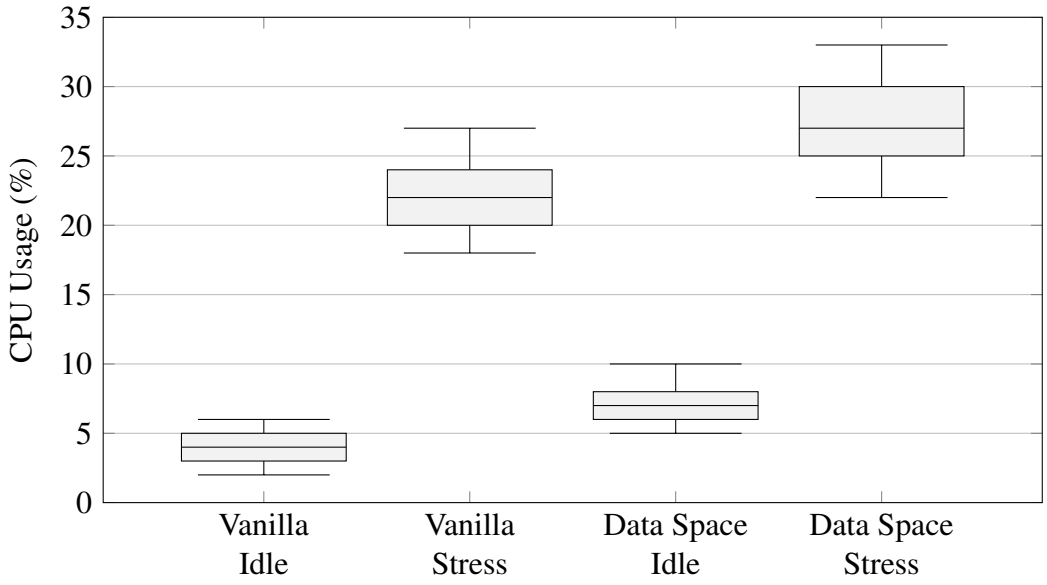


Fig. 5.3 CPU consumption.

5.4.3 Resource consumption

Resource consumption serves as a key metric for evaluating the computational demands placed upon a system during data transfer events. In the context of our study, this evaluation aims to identify any potential overhead on the Liqo Gateway resulting from our additional security rules. To this end, we conducted an analysis based on four distinct scenarios, as depicted in Figure 5.3, which were drawn from both the vanilla implementation of Liqo and our proposal. Initially, data was collected during an idle period before a stress test was initiated. This test design was intended to facilitate a comparative examination of the four scenarios. To simulate data transfer between the two clusters, we used the tool iPerf[16].

Upon examining the graph, we observed that our implementation, denoted as *Data Space* in Figure 5.3, imposes only a negligible amount of overhead, whether in idle or stress conditions, when compared to the vanilla Liqo implementation. This minor increase in CPU consumption is a small price to pay for the advantages of our proposed solution, which allows data to be provisioned via data spaces. This solution avoids introducing any substantial overhead that might be deemed unacceptable if it significantly exceeded that of the vanilla Liqo implementation. We also recorded RAM consumption throughout our experiments. However, our findings indicated that

[16]https://iperf.fr

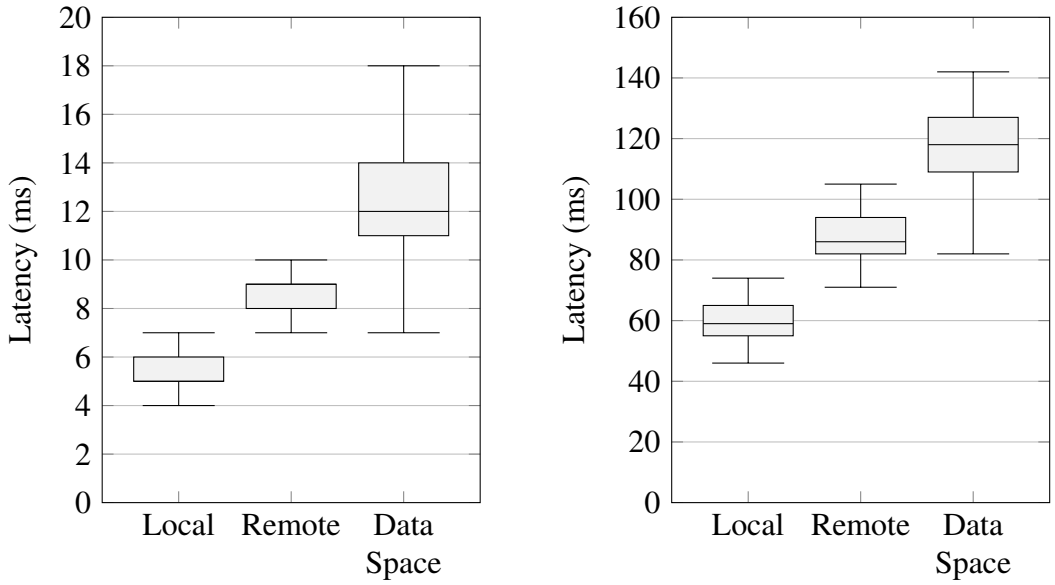
our solution had no discernible impact on RAM usage when compared to the vanilla Liqo. Consequently, in the interest of brevity, we have omitted the corresponding graph.

5.4.4 Latency

Latency, within this context, is defined as the duration between the moment data is requested by a pod and the point at which the requested data becomes available. The premise of our experiment was that the desired data resides within a pod, leading us to explore pod-to-pod communication scenarios. Under this premise, we evaluated three distinct scenarios to ascertain the time needed for a pod to access data. (i) In the first scenario, the communicating pods were located within the same cluster (either Pharma or Hospital). Thus, these pods utilized local connectivity for data exchange. This scenario simulates communication between a data-seeking pod and a data-provider pod within the same cluster. We term this scenario as *Local*. (ii) In the second scenario, the communicating pods were situated across two different clusters (one in Pharma and the other in Hospital), necessitating the use of Liqo for connection. Consequently, data transfer had to pass through the inter-cluster tunnel. This scenario depicts communication between pods in separate clusters, offering a comparison point to a situation where Pharma needs to access a remote service located in the Hospital cluster and download all relevant data for computation within the Pharma cluster. We refer to this scenario as *Remote*. (iii) The third scenario mirrors the second but it introduces one or more offloaded Pharma pods within a data space in the Hospital cluster. In this setup, Pharma pods need to liaise with these offloaded pods, which in turn, communicate with the data-bearing pod in the Hospital. The offloaded pod(s) play a *proxy* role, providing the Hospital with control over data output. This scenario is referred to as *Data Space*.

We gathered latency benchmarks utilizing Apache Benchmark[17], entailing 10K requests with a varying number of parallel connections. For ease of understanding, we report two instances, one involving 10 parallel connections and the other 100, depicted in Figure 5.4a and Figure 5.4b, respectively. As illustrated in these figures, pod-to-pod latency increases when the pods belong to different clusters compared to those within the same cluster. However, the *Data Space* scenario warrants further

[17]https://httpd.apache.org/docs/2.4/programs/ab.html



(a) Latency with 10 parallel connections.

(b) Latency with 100 parallel connections.

Fig. 5.4 Latency experimental evaluation with 10 and 100 parallel connections.

examination. During the testing process, the offloaded Pharma pods functioned merely as proxies, managing all requests to Hospital data, thus inducing a noticeable latency impact. This proxy role gives Hospital control over data flow since the offloaded pods are situated within the data space. When deploying our solution in this manner, latency increases by up to 37% (median value) in comparison to the *Remote* scenario.

This finding directs us back to the initial premise of our work. If we leverage the data gravity capability inherent in our proposed solution, we can manage local and remote latencies more effectively. By assigning data aggregation or analysis tasks to the Pharma offloaded pod, the resultant data has a smaller size compared to the original. In this setup, communication between the offloaded pod and the data-owning Hospital pod remains local to the Hospital cluster, mimicking the *Local* scenario latency behaviour depicted in Figure 5.4a and Figure 5.4b. Conversely, when the Pharma pod retrieves the aggregated data from the offloaded pod, the latency mirrors the *Remote* scenario. Although *Data Space* latency is higher than *Local* and *Remote* latencies, data aggregation mitigates this issue by reducing latency within the data aggregator pod and lowering the total volume of data transferred from Hospital to Pharma, hence the total cost of the deployment in case of clusters

running in public cloud providers. While we cannot lessen the latency directly, we can effect a significant reduction in total data transfer time.

5.4.5 Discussion of results

The results in this section indicate that (i) establishing a data space between two organizations completes within a practical time window (Section 5.4.2); (ii) the Sidecar/ConfigMap connector introduces a modest, bounded resource overhead relative to running the same workload without data-space mediation (Section 5.4.3); and (iii) the added communication latency remains acceptable for the evaluated workflow (Section 5.4.4). As discussed above, these results validate the mechanism's basic feasibility but do not yet address its security robustness against an actively evasive consumer, which remains an open limitation (Section 5.2 and Section 5.4).

5.5 Liqo and the Gaia-X architecture

The data spaces architecture presented in Section 5.1 and generally adopted by Gaia-X defines application-level primitives, but it does not mention any infrastructure-level solution. In our opinion, application-oriented data spaces open up the following problems: (i) it enables secure data exchange, but it does not support any computing component (i.e., pods); hence, in case of a very large quantity of data, all data needs to be transferred from the producer to the consumer (no support for data gravity); (ii) IDSA connector allows access to data from the data consumer, but the latter might prefer having its own API gateway with its aggregation rules; (iii) mutually reachable endpoints are required (e.g., publicly exposed on the Internet) to allow connectors to exchange data.

The architecture presented in this chapter can be easily integrated with application-level data spaces to address the aforementioned problems. For instance, in infrastructure-level data spaces, connectors are just pods that can be executed on either the local or remote cluster, depending on the desired scenario. Hence, considering the architecture in Figure 5.1, Liqo can be used to enhance the pure data exchange capabilities of current data space solutions with a data-gravity approach, enabling the remote deployment of processing instances instead of transferring data. We begin with an overview of the existing workflow involving IDS Connector, followed by an

exploration of integration with Liqo. We assume that each involved entity has both the IDS Connector and Liqo already installed.

IDS connectors can publish the description of their data endpoints at an IDS meta-data broker. This allows potential data consumers to look up available data sources and data in terms of content, structure quality, actuality, and other attributes. Consequently, data consumers can make informed decisions on selecting a data producer based on their specific requirements and needs.

The initial step entails the data consumer contacting the chosen data producer via the data connector to initiate a request for data exchange. To establish a secure communication channel, the data producer's connector and the data consumer's connector engage in authentication and authorization mechanisms. These mechanisms verify the identity and access rights of both parties, which may involve the exchange of authentication tokens, certificates, or other credentials. Once the data producer's connector has validated the data usage conditions and obtained any necessary consent, it grants the data consumer's connector access to the requested data. The data consumer's connector enforces any access control rules defined by the data producer's policies, such as filtering or reduction. Over the established communication channel, the data producer's connector securely transmits the requested data to the data consumer's connector, allowing the latter to receive and process the data according to its specific requirements. Within this workflow, Liqo introduces the potential for two new scenarios.

The first scenario leverages the nature of Liqo itself. Utilizing its direct connection between clusters, Liqo can facilitate peering between the involved entities after the authentication phase. In essence, instead of exchanging data between the connectors through public endpoints, Liqo enables the use of a private tunnel and service reflection. By doing so, the need for public endpoints to transmit data is eliminated, ensuring enhanced security without disrupting the current implementation of the IDS connector.

The second scenario leverages Liqo's ability to deploy pods in remote clusters for exchanging computations rather than data. This approach relieves the requirement to transfer large amounts of data by instead transmitting the processing units responsible for handling that data. Leveraging Liqo's data-gravity approach, bandwidth consumption is reduced, and computations are attracted to the data, thereby optimizing resource utilization and efficiency.

5.6 Concluding remarks

This chapter delineates a novel approach to create infrastructure-level data spaces, harnessing the capabilities of the Liqo project. The proposed solution enables data consumers to establish effective connections with data producers, thereby facilitating controlled data retrieval. Additionally, it promotes the deployment of processing algorithms closer to the data, thereby fostering data gravity.

Our findings substantiate the advantages of this solution, showcasing a negligible overhead against the *vanilla* Liqo. The proposed solution can be employed to safeguard data exchange via data spaces, empowering data providers with the ability to regulate the exchange of data. This capability gives data providers the flexibility to terminate the connection as and when they see fit. Simultaneously, our solution offers the advantage of utilizing offloaded pods for data aggregation, enabling data consumers to tap into the potential of data spaces. This signifies a shift of computational operations closer to the data source, thereby diminishing latency between computation and data. Moreover, it facilitates a reduction in data transfer if the data is aggregated or analyzed *in situ* by offloaded pods before being transmitted to the data consumer. The contraction in data transfer consequently reduces the time required for data retrieval once the aggregation or analysis phase concludes. When contextualized within the framework of initiatives like IDSA and the Gaia-X project, our proposed solution can serve as an instrumental component of these broader ecosystems, acting as a key element to enable data gravity.

In conclusion, our solution provides a versatile, efficient, and secure approach for infrastructure-level data exchange. By enabling data gravity, it aligns with future-oriented trends of data localization and analytics, empowering the architecture proposed by Gaia-X.

Chapter 6

Concluding Remarks

This part of the thesis addressed the challenge of building a true computing continuum over the heterogeneous and geographically distributed infrastructures that characterise modern telecommunications and cloud deployments. Starting from an analysis of the industrial and research landscape, it identified the fundamental limitations of today's siloed approaches and proposed a coherent set of architectural mechanisms to overcome them. The contributions span three interconnected dimensions: the physical and operational context that motivates the continuum, the protocols and frameworks that realise it, and the data governance mechanisms that make it viable in multi-domain environments.

6.1 From fragmented infrastructures to a unified continuum

Chapter 1 established the core motivation: despite the apparent maturity of container orchestration and the widespread adoption of Kubernetes, computing infrastructures remain fundamentally fragmented. Each cluster, whether deployed in the cloud, at the edge, or on-premises, is governed by an independent control plane that exposes only a partial and static view of globally available resources. This fragmentation is not accidental: it reflects well-known limitations of centralised control planes under wide-area network conditions, combined with regulatory, organisational, and security constraints that incentivise cluster proliferation. The result is a computing continuum

that exists in name but not in practice: applications must be explicitly assigned to specific infrastructures at deployment time, and dynamic workload migration, transparent load redistribution, and cross-domain resource compensation remain largely out of reach.

Chapter 2 examined this challenge from the perspective of physical infrastructure at scale. Drawing on concrete data from BT's network in the UK and ARCEP's figures for France, as well as recent industrial initiatives such as FiberCop's edge cloud trial and the pan-European federated edge continuum announced at MWC 2026, the chapter demonstrated that telco operators possess a vast and geographically distributed substrate (comprising tens of thousands of central offices, telephone exchanges, and 5G cell sites) that is, in principle, ideally suited to host edge computing resources. The analysis showed, however, that the sheer scale of this infrastructure, ranging from a few hundred national points of presence to tens of thousands of distributed exchange sites, transforms the management challenge qualitatively. It is precisely this scaling challenge that motivates the edge-to-cloud continuum not merely as a desirable architectural vision but as a practical necessity for making distributed edge deployment tractable at commercial scale.

6.2 Technologies and architectural foundations

Chapter 3 surveyed the current technological landscape, examining production-ready platforms, research-oriented proposals, and past approaches to multi-cluster orchestration. Among mature open-source solutions, KubeEdge, Karmada, and Liqo emerged as the most relevant candidates. KubeEdge targets resource-constrained far-edge devices but does not provide unified networking or storage abstractions across sites. Karmada offers centralised multi-cluster management with advanced scheduling, but its master–worker model limits topological flexibility and it lacks a native network fabric. Liqo, in contrast, supports both hierarchical and fully federated multi-cluster topologies, maintains strong compatibility with vanilla Kubernetes APIs, and provides integrated networking and storage abstractions, making it the most comprehensive foundation for a computing continuum among the solutions analysed. Research-oriented contributions, including CRDT-based decentralised control planes and component-level offloading architectures, introduced important

design principles that informed the direction taken in subsequent chapters, even if they remain short of production readiness.

Chapter 4 presented the FLUIDOS approach to realising the computing continuum, centred on three transparency properties (deployment transparency, communication transparency, and resource availability transparency) that current solutions fail to provide. Deployment transparency removes the need for developers and operators to reason about where each component runs, delegating placement to intent-driven schedulers that act across a unified virtual infrastructure. Communication transparency ensures that microservices interact as if co-located within a single cluster, regardless of their actual physical distribution, through a virtual network fabric that abstracts away cross-cluster boundaries. Resource availability transparency allows services to scale seamlessly across the entire virtual infrastructure, drawing on resources from remote domains as if they were local, thereby eliminating the capacity walls that constrain edge deployments.

Central to the FLUIDOS architecture is the REAR protocol, designed to enable standardised, efficient, and secure resource advertisement and reservation across the continuum. REAR supports a rich data model encompassing Kubernetes cluster slices, VMs, services, sensors, and datasets, and defines a complete workflow for resource discovery, reservation, and purchase. Its authentication and authorisation mechanisms, based on decentralised identifiers and verifiable credentials, provide a zero-trust foundation for cross-domain resource transactions. Experimental evaluation demonstrated that peer discovery scales predictably with the number of nodes in the continuum, that the REAR protocol overhead remains minimal relative to the total resource acquisition time, and that application offloading enabled by FLUIDOS yields measurable reductions in CPU load and power consumption in robotics use cases, extending operational autonomy by approximately 15%.

6.3 Data governance in multi-domain environments

Chapter 5 addressed a challenge that becomes critical as the computing continuum spans multiple administrative domains: ensuring that data can be consumed and processed across domain boundaries without compromising sovereignty, privacy, or regulatory compliance. The chapter proposed infrastructure-level data spaces as a mechanism for controlled cross-domain data sharing, leveraging Liqo's multi-

cluster capabilities to allow a data consumer to deploy processing logic directly within a data producer's cluster. This data-gravity approach avoids the need for large-scale data transfers, instead moving computation to the data. The architecture enforces a set of security properties (restricted pod privileges, limited pod-to-pod connectivity, controlled outbound traffic, and traffic inspection via sidecar proxies) that prevent data exfiltration while preserving the data producer's governance over its own infrastructure.

Experimental evaluation showed that the infrastructure-level data space introduces negligible overhead relative to a vanilla Liqo deployment, both in terms of pod deployment time and CPU and RAM consumption at the inter-cluster gateway. Latency increases by up to 37% in the data space scenario compared to a direct remote connection, a consequence of the proxy role played by offloaded pods. However, this overhead is offset by the reduction in total data transfer when aggregation or analysis is performed in situ, making the approach particularly attractive for large datasets and latency-tolerant analytics workloads. The chapter also demonstrated how infrastructure-level data spaces can be integrated with application-level data space standards such as the IDSA architecture and the Gaia-X ecosystem, extending their capabilities with data-gravity-based computation offloading and private inter-cluster tunnelling.

6.4 Open challenges and future work

The contributions presented in this part collectively establish a solid architectural and experimental foundation for the computing continuum. Nevertheless, several challenges remain open and warrant further investigation.

Resource discovery at scale currently relies on either multicast-based mechanisms suitable for local-area networks or DHT-based approaches for wide-area deployments. As the number of continuum participants grows to encompass tens of thousands of edge sites, efficient and privacy-preserving discovery (one that does not require full topology exposure) remains an active research problem. Integrating AI-driven scheduling strategies into the meta-orchestrator could further improve placement decisions by learning from historical workload patterns and infrastructure dynamics. Rather than relying solely on static policies (e.g., bin-packing heuristics or fixed affinity rules), a learning-based scheduler could be trained to predict the

future resource demand of a workload from its observed behavior, and to anticipate the load and latency conditions of candidate nodes before a placement decision is made. Concretely, this could take the form of a reinforcement-learning agent that treats each placement decision as an action and uses observed post-placement metrics (e.g., achieved latency, resource contention, migration frequency) as reward signals, progressively refining its policy as the continuum evolves. The expected benefit over traditional policy-based scheduling lies primarily in handling multi-objective trade-offs (for instance, jointly optimizing for latency, energy consumption, and load balancing) that are difficult to encode as static rules, and in adapting placement strategy as the population of nodes and workloads changes over time, without requiring manual policy re-tuning. This remains an open challenge, as it requires reconciling the responsiveness needed for online scheduling with the data and training requirements typical of learning-based approaches.

Interoperability with heterogeneous orchestration environments beyond Kubernetes represents another open direction. While FLUIDOS leverages KubeEdge to extend its reach to resource-constrained far-edge devices, a broader integration with non-Kubernetes orchestrators, serverless platforms, and unikernel-based runtimes would expand the scope of the continuum.

Finally, the enforcement of data sovereignty in real-world deployments requires advances in policy expression, automated compliance verification, and runtime attestation. The REAR protocol embeds compliance metadata in resource advertisements, and verifiable credentials provide a lightweight authentication mechanism, but closing the loop between advertised properties and continuously verified behaviour under adversarial conditions remains a significant challenge, particularly in environments involving TEEs and hardware-backed security primitives.

These open challenges motivate the second part of this thesis, which applies the cloud continuum foundations developed here to the specific domain of robotics, exploring how dynamic resource offloading and reliable communication in unreliable network environments can extend the operational capabilities of mobile robotic systems.

Part II

Cloud Continuum and Robotics

Chapter 7

Introduction

Part I established the architectural and protocol-level foundations of the cloud continuum (transparent deployment, decentralized resource discovery via REAR, and infrastructure-level data governance), but did so under the implicit assumption of general-purpose, latency-tolerant workloads. This second part turns to a workload class that violates that assumption: mobile robotics, where perception and control loops impose hard real-time constraints that a general-purpose continuum cannot ignore. Rather than directly reusing the REAR/FLUIDOS machinery, Part II asks a complementary question: which continuum-style mechanisms (communication transparency, dynamic offloading, resource-aware adaptation) must be redesigned from the ground up once latency, reliability, and energy budgets become first-class constraints?

This question matters because of how robotic platforms are evolving. Modern robots, such as planetary rovers, aerial drones, and autonomous vehicles, are increasingly equipped with on-board computing hardware that enables them to execute perception and decision-making tasks. These capabilities allow robots to interact with their surroundings by identifying obstacles, interpreting sensor data, and performing autonomous navigation. However, many perception algorithms, particularly those based on Machine Learning (ML), such as object detection or scene understanding, require significant computational resources. Properly dimensioning the hardware of a robotic platform is therefore challenging, as designers must consider the computational requirements of the target tasks, mission duration, energy consumption, and cost constraints.

To address these limitations, recent approaches have explored the possibility of delegating part of the computation to external systems with fewer resource constraints. By offloading selected tasks to edge or cloud infrastructure, robots can effectively extend their computational capabilities while reducing on-board resource requirements. This paradigm, commonly associated with MEC [87, 5], enables robots to execute complex algorithms, such as ML-based perception or advanced autonomy, without requiring oversized hardware platforms. As a result, smaller and more energy-efficient robotic systems can achieve levels of performance comparable to significantly more powerful platforms.

At the same time, the growing demand for specialized hardware accelerators, such as GPUs or Field-Programmable Gate Arrays (FPGAs) used for machine learning inference, has made it increasingly difficult for individual robotic platforms to host all required resources locally. Edge computing has therefore emerged as a viable strategy to offload computation to external infrastructure equipped with the necessary hardware resources [22]. This approach can improve system performance while reducing energy consumption [88]. Consequently, modern robotic applications frequently extend beyond a single device and instead involve multiple cooperating entities communicating across the network [89].

However, the benefits of offloading are strictly limited by the reliability of the underlying wireless network [90]. Even modern wireless systems (e.g., 5G, WiFi 6), and future ones such as 6G, suffer from latency variability, packet loss, and occasional disconnections [91]. These challenges are exacerbated when robots move in and out of coverage, as not only the quality of the communication medium, but also obstacles interposing during the movement of the robot could disrupt communication with the cellular tower or access point. As a result, even if the robots and the servers for the computation are in close proximity, latency and network reliability [92–94] can still be deeply affected. Fog robotics addresses this concern by distributing computation, storage, and control closer to the robots rather than relying solely on remote cloud servers [95, 92]. Similarly, the MEC paradigm brings compute and storage capabilities even closer to the robot, reducing latency and alleviating the ‘last mile’ bottleneck [96, 97]. Nevertheless, shortening the distance between robots and computation alone is not sufficient, and adaptive mechanisms are needed to ensure safe and reliable operation under varying network conditions.

The value of next-generation network infrastructures, such as those envisioned for 6G, cannot be fully appreciated in isolation from the applications they are designed to support. Without concrete use cases that exercise the capabilities of these future architectures, the 6G paradigm risks remaining an abstract concept with limited practical significance. For this reason, the work presented in this part of the thesis validates the architectural principles of the cloud continuum through a use case grounded in robotics, demonstrating how the interplay between distributed computing and advanced connectivity can address real operational challenges.

Developing such distributed robotic applications requires software frameworks capable of coordinating multiple interacting components running on different processes and machines. Among these frameworks, the ROS [98] and its successor Robot Operating System 2 (ROS2)[1] [99] have become widely adopted platforms for building robotic systems, offering a modular architecture that enables developers to structure applications as collections of interacting nodes.

ROS2 is a middleware framework designed to support the development of distributed robotic applications. It provides a strongly typed, anonymous publish/subscribe communication model that enables message exchange between different processes and machines.

At the core of a ROS2 system lies the *ROS graph*, which represents the network of computational entities and their communication relationships. The basic participant in this graph is the *node*. A node is an independent unit of computation that uses a ROS client library to interact with other nodes. Nodes can run in the same process, in different processes, or on separate machines, allowing ROS2 systems to scale across distributed environments. Typically, each node is designed to perform a single logical function.

Nodes interact through several communication abstractions. A node may publish data to *topics*, subscribe to topics produced by other nodes, invoke or provide *services*, or interact through *actions*. Nodes may also expose configurable *parameters* that allow their behaviour to be adjusted at runtime. In practice, a node may simultaneously act as a publisher, subscriber, service server, service client, action server, and action client.

[1] https://docs.ros.org/en/kilted/index.html

Communication between nodes is enabled through a distributed discovery mechanism provided by the underlying ROS Middleware (RMW). When a node starts, it advertises its presence to other nodes within the same ROS domain, which is identified by the `ROS_DOMAIN_ID` environment variable. Other nodes respond with their own information so that compatible communication endpoints can be established. Nodes periodically re-advertise their presence to allow late-joining nodes to discover existing participants, and they notify others when they leave the system. Connections between nodes are established only when their QoS settings are compatible.

ROS2 applications communicate through three main interaction patterns: topics, services, and actions. The structure of these communication interfaces is defined using the Interface Definition Language (IDL), which allows ROS2 tools to automatically generate source code for multiple programming languages.

The following interface definitions are supported:

- **Messages (**`.msg`**)**: define the structure of data exchanged between nodes.
- **Services (**`.srv`**)**: describe request/response interactions composed of a request message and a response message.
- **Actions (**`.action`**)**: describe long-running tasks consisting of a goal, feedback messages, and a final result.

Topics implement a publish/subscribe communication model and are typically used for continuous data streams such as sensor readings or robot state information. Publishers send messages to a named topic, while subscribers receive messages from the same topic. Multiple publishers and subscribers may coexist on a topic, and any message published is delivered to all subscribers. This loosely coupled model allows nodes to dynamically join or leave the system without affecting other participants and supports powerful introspection and debugging tools.

Services implement a synchronous request/response communication model similar to remote procedure calls. A service client sends a request to a service server, which performs a computation and returns a response. Services are intended for short-lived operations where the client waits for the result. For this reason, they are not suitable for long-running tasks.

Actions extend the service model to support long-running operations. An action client sends a goal to an action server, which performs the requested task while

periodically providing feedback on its progress. Actions also support cancellation and preemption of ongoing tasks. This interaction pattern is commonly used for operations that may take significant time to complete, such as robot navigation.

Together, these communication mechanisms allow ROS2 to support a wide range of interaction patterns required in distributed robotic systems while maintaining loose coupling between software components.

7.1 Summary of contributions

Part II of this dissertation is structured into four main chapters, applying the cloud continuum foundations to the domain of distributed robotics. Specifically, the contributions of each section can be summarised as follows:

- **Chapter 8** introduces Scalable Middleware 2 (SM2), a lightweight middleware that decouples local inter-process communication from network transport. SM2 addresses the performance degradation observed in DDS-based ROS2 configurations under adverse network conditions, reducing CPU usage by $3\times$ and memory consumption by $4\times$ compared to state-of-the-art alternatives, while sustaining higher message frequencies and lower latencies as the number of communicating nodes increases.

- **Chapter 9** investigates four switching architectures for migrating stateless ROS2 computation between Kubernetes clusters, defined along the dimensions of redeployment, network policies, and Lifecycle Nodes. Experimental evaluation demonstrates that the Lifecycle Node-based approach achieves switching times at least 85% faster than redeployment-based alternatives, while preserving service continuity through controlled suppression of duplicate messages during the transition period.

- **Chapter 10** extends the offloading paradigm to unreliable network environments, where packet loss and latency variability can severely degrade the performance of remotely controlled robots. The chapter proposes a Multiplexer (MUX) component that dynamically arbitrates between local and remote navigation streams based on message freshness and reliability, achieving a 31% improvement in navigation performance over remote-only control at 55%

packet loss. A complementary Lifecycle Orchestrator (LO) reduces onboard CPU usage by up to 99% during periods of stable remote connectivity.

- **Chapter 11** synthesises the contributions of Part II, discussing how the presented work demonstrates that the cloud continuum paradigm can be effectively applied to robotics when each system layer (from communication middleware to orchestration and adaptive stream management) is designed with awareness of the constraints imposed by distributed and potentially unreliable environments. Open directions for future work are identified across communication, migration, and resilience dimensions.

7.2 Previously published works

This part includes previously published and co-authored works. In particular, Chapter 8 is adapted from an accepted, but not yet published, work. Chapter 9 is an adaptation of [89]. Chapter 10 is adapted from an unpublished work.

Chapter 8

Robotics Communications in the Cloud Continuum

Modern robotic systems rely on complex software architectures composed of numerous interacting components that must exchange data efficiently and reliably. These components often run as separate processes and must communicate under strict performance and real-time constraints. To manage this Inter-Process Communication (IPC), the robotics community has widely adopted middleware frameworks that provide standardized communication abstractions, allowing developers to focus on application logic rather than low-level networking details. Among these frameworks, the ROS [98], and more recently ROS2 [99], have become the de facto standard for building modular robotic software systems.

ROS2 introduces a flexible communication architecture that allows different middleware technologies to be used transparently by applications. Middleware implementations in ROS2 are provided through sets of packages that implement the underlying communication layer and interface with the core ROS2 APIs, such as `rmw`, `rcl`, and `rosidl`. These middleware implementations are responsible for providing fundamental communication capabilities including discovery, serialization, and message transport.

Supporting multiple middleware implementations allows ROS2 to adapt to different deployment scenarios, since no single communication solution is optimal for every system. While ROS2 was initially designed around the Data Distribution Service (DDS) and its Real-Time Publish-Subscribe (RTPS) protocol as the default

communication substrate[1], the ecosystem has progressively expanded to support alternative middleware architectures.

Although ROS2 introduces significant improvements over ROS, its default DDS-based communication stack can exhibit performance limitations in distributed deployments, particularly when operating across unreliable or heterogeneous networks [100, 101]. In particular, introducing remote subscriptions, where subscriber nodes run on different machines connected through a network, can lead to performance degradation at the publisher. Under poor network conditions, this degradation may also affect local subscribers that would otherwise operate independently. This coupling between network conditions and local node performance represents a potential reliability issue for real-world robotic systems.

DDS middleware. The DDS is an industry-standard middleware designed for real-time, data-centric communication in distributed systems. Multiple vendors provide DDS implementations, including RTI Connext DDS[2], eProsima Fast DDS[3], Eclipse Cyclone DDS[4], and GurumDDS[5].

DDS systems typically communicate over the network using the RTPS protocol, formally known as DDSI-RTPS[6]. Within ROS2, DDS provides the mechanisms for distributed node discovery, message delivery, and communication management.

One of the key advantages of DDS is its decentralized discovery mechanism, which eliminates the need for centralized coordination as used in ROS. Additionally, DDS exposes a rich set of QoS policies that allow developers to configure communication behaviour according to the requirements of different applications. These policies include parameters such as reliability, durability, and deadline constraints, enabling fine-grained control over message delivery.

Despite these advantages, DDS-based communication may introduce overhead and complexity when used in highly distributed environments or across unreliable networks. These limitations have motivated the exploration of alternative middleware solutions better suited to heterogeneous and large-scale deployments.

[1] https://design.ros2.org/articles/ros_on_dds.html
[2] https://www.rti.com/products/
[3] https://fast-dds.docs.eprosima.com/
[4] https://projects.eclipse.org/projects/iot.cyclonedds
[5] https://gurum.cc/index_eng
[6] https://www.omg.org/spec/DDSI-RTPS/About-DDSI-RTPS/

Zenoh. Zenoh [102] is a data-centric middleware designed to unify communication, storage, and computation across heterogeneous environments ranging from cloud infrastructures to constrained edge devices. It addresses the challenges of highly distributed systems where data must be efficiently exchanged, queried, and processed across networks with varying performance characteristics.

Zenoh adopts a unified abstraction based on key expressions that allow applications to publish, subscribe to, query, and store data through a common interface. This model supports multiple interaction patterns, including publish/subscribe communication, distributed queries, and persistent storage, within a single framework.

The architecture of Zenoh supports both peer-to-peer and routed communication, enabling efficient data exchange between robotic systems, edge devices, and cloud services. As a ROS2 middleware implementation, Zenoh provides a lightweight alternative to DDS while preserving key communication features such as QoS. Its reduced wire overhead and flexible routing capabilities make it particularly suitable for deployments involving heterogeneous networks or constrained devices.

Recent ROS2 integrations of Zenoh attempt to address some of the limitations of DDS-based deployments by introducing dedicated routing components that decouple local and remote communications. While effective in mitigating the performance impact of unreliable networks, these approaches may introduce additional CPU and memory overhead due to the presence of intermediary routing nodes. This overhead becomes increasingly relevant as the number of communicating nodes grows [103].

Scalable Middleware 2. To address these limitations, we introduce *Scalable Middleware 2 (SM2)*[7], a lightweight and deterministic software framework designed to provide fast, robust, and efficient local communication while maintaining support for distributed deployments.

SM2 is motivated by the need for a practical and scalable IPC solution for robotics that preserves predictable local performance even in the presence of adverse network conditions. The architecture decouples local inter-process communication from network communication, ensuring that local modules remain unaffected by network variability. External connectivity is provided through an optional, low-overhead gateway that enables communication across machines without interfering with local data flows.

[7]The number 2 does not indicate that this is the second version, but rather that it is positioned as an alternative to ROS2. https://github.com/andrea-fasano/sm2

By focusing on simplicity and performance, SM2 aims to deliver an IPC mechanism that retains the ease of use familiar from ROS2 while avoiding the architectural complexity and runtime overhead associated with DDS-based middleware stacks.

The main contributions of this work are as follows. First, we introduce SM2, a lightweight IPC solution enabling efficient intra-robot communication. Second, we present the SM2 Network Gateway (SNG), which decouples local communication from network communication and extends the architecture across multiple machines while preserving predictable performance. Third, we provide an experimental evaluation against ROS2 configurations, demonstrating that SM2 achieves comparable throughput with significantly lower resource usage and latency[8].

8.1 Related work

A significant amount of research has been dedicated to evaluating the performance of ROS2, both in comparison with its predecessor [104, 105] and across different middleware implementations [106, 101]. These works investigate how different middleware backends (such as Fast DDS, Cyclone DDS, and Zenoh) affect communication latency, throughput, and resource utilization, highlighting the impact that middleware choices can have on the performance and scalability of distributed robotic systems.

Thulasiraman et al. [107] demonstrated that lossy network conditions can severely degrade the performance of DDS-based communication, particularly when security features are enabled. Similarly, Lee et al. [108] found that DDS-based transport struggles to handle large messages over wireless links, attributing the issue to fragmentation, retransmission timing, and buffer bursts. Maruyama et al. [104] further observed low throughput and process instability during large-message transmission, both in simulated and real environments.

[8]This work was conducted in collaboration with Andrea Fasano, Daniele Cacciabue, Stefano Galantino, and Fulvio Risso. It was partly supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino and Daniele Cacciabue Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative. SM2 was developed for the DRAFT (DRones Autonomous Flight Team) team at Politecnico di Torino to participate in the Leonardo Drone Contest.

Despite extensive evidence of performance degradation under non-ideal network conditions, there remains a substantial lack of research on how network communication impacts local performance and latency in controlled environments. This gap is particularly relevant for systems requiring deterministic or low-jitter communication, where DDS's design choices can introduce non-trivial overhead even in reliable networks.

Comparative studies of available middleware have shown that each distribution exhibits unique strengths and weaknesses. DDS-based implementations generally perform worse under adverse network conditions than their counterparts based on alternative protocols. Zhang et al. [101] showed that middleware based on Zenoh outperforms DDS-based options in terms of throughput in real robotic scenarios, particularly when relying on a centralized broker, although DDS still achieves lower latency in certain cases. Chovet et al. [100] reached similar conclusions under different real-life testing setups.

In terms of local computation and intra-process communication, Macenski et al. [103] reported significant overhead when increasing the number of separate ROS2 processes. Their proposed mitigation, using executors and intra-process communication instead of standard IPC, reduces wasted resources but is not always applicable or advisable across all development and deployment contexts.

Several studies have analysed DDS-based communication in specialized domains. Paul et al. [109] evaluated vendor-specific ROS2 DDS implementations for cooperative driving, showing that latency and reliability depend heavily on data size, connection type, and bridging overhead. Puck et al. [110] demonstrated that with careful real-time tuning (e.g., real-time-safe memory allocation, thread prioritization, and CPU shielding), ROS2 can sustain deterministic communication with sub-millisecond latency, proving its feasibility for real-time control on standard hardware. Kouril et al. [111] conducted similar experiments for automated driving, reporting latencies below 2ms and packet losses under 1%, with Cyclone DDS providing the most consistent performance, sufficient for reliable soft real-time operation but not for strict hard real-time guarantees.

Beyond DDS itself, Time-Sensitive Networking (TSN) has been proposed as a complementary technology to ensure bounded delay and jitter under heavy load. Gutiérrez et al. [112] demonstrated TSN's potential for deterministic, low-latency robotic communication, suggesting its integration with DDS as a promising direction.

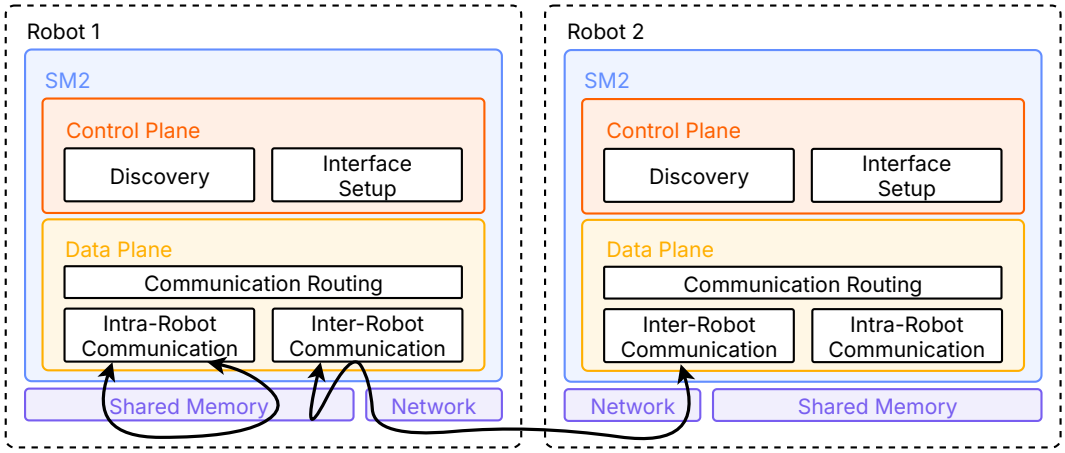


Fig. 8.1 SM2 logical view depicting the communication architecture and data exchange paths involved in intra-robot and inter-robot communication.

Similarly, San Vicente Gutiérrez et al. [113] evaluated ROS2 on a real-time Linux kernel (PREEMPT-RT), finding that careful configuration (thread priorities, memory locking, etc.) can achieve bounded communication latency, though full real-time determinism remains elusive.

Overall, while numerous studies characterize ROS2 and its DDS-based middleware across different setups, there remains no comprehensive solution addressing the intrinsic latency variability and inefficiencies of DDS in both networked and local contexts. This gap highlights the need for alternative communication approaches or optimizations capable of delivering predictable, low-latency performance across heterogeneous and dynamic robotic environments.

8.2 SM2 core

The highest priority is to guarantee reliable and deterministic local communication, regardless of external factors. The design of the solution mirrors this requirement by decoupling local and network communication. In the following, we first detail the control plane functionalities offered by SM2 for communication setup, moving then to the data plane functionalities to guarantee reliable communication (see Figure 8.1).

Table 8.1 Comparison of communication frameworks for robotic systems.

Feature	**ROS**	**ROS2 (DDS)**	**ROS2 (Zenoh)**	**SM2**
Node Discovery	Centralized (ROS Master)	Decentralized (Multicast)	Centralized Router or Peer-to-Peer	Centralized (Manager Node)
Intra-Robot Communication	TCP/UDP sockets	Shared Memory (optional) or UDP fallback	Shared Memory or TCP/UDP	Dedicated Shared Memory
Inter-Robot Communication	TCP/UDP sockets	DDS over UDP	TCP/UDP via Zenoh Routers	QUIC via SNG
Scalability	Limited by ROS Master bottleneck	Moderate (discovery traffic overhead)	High (router-based)	High (Manager + SNG)

8.2.1 Control plane: discovery and interface setup

The control plane handles discovery, registration, and interface setup within the SM2 domain. The SM2 Library is a C++ IPC library designed for robotics applications with strict real-time and reliability requirements. Similar to ROS, the library allows separate software modules to exchange data over a topic interface (publish/subscribe) or an interface like Remote Procedure Call (RPC) (service). At its core, SM2 builds on abstractions of system-level IPC primitives. Control messages are exchanged through the Socket abstraction, primarily derived from Unix Domain Sockets, while bulk data transfer is handled through the SharedMemory abstraction, built on Portable Operating System Interface (POSIX) shared memory.

In ROS, node discovery is centralized: a process called ROS Master is responsible for discovery and for storing node-related information. However, if the ROS Master fails, it not only disables the discovery of new nodes but also undermines currently running connections. ROS2 overcomes such limitations by performing the discovery process using multicast traffic, which removes the need for a central ROS Master, preventing the single-point-of-failure scenario. The solution fully decouples components, allowing the system to continue working even in the case of one (or more) component failure. However, in modern robotics, a robot is no longer an isolated entity but part of a distributed computing environment. This shift strongly

impacts scalability in multi-robot systems, which often rely on lossy networks where multicast delivery can be unreliable. To avoid the multicast delivery problem, in SM2 the registration and setup of communication are centralized, whereas data exchange and feedback occur in a peer-to-peer manner. The central authority overseeing communication setup is the *Manager* node, which is the only publicly accessible node within an SM2 local domain. All other nodes must register with the Manager and interact with it to create interfaces and establish peer-to-peer communication with their peers. As a concrete example, consider two robotic nodes, A and B, within the same SM2 local domain. Node A first registers with the Manager, declaring the topics and services it intends to publish or expose. When node B subsequently wants to subscribe to one of A's topics, it queries the Manager, which returns the connection information needed to establish a direct peer-to-peer channel between A and B. From this point on, all data exchanged between A and B flows directly over this peer-to-peer channel, without involving the Manager; the Manager is only consulted again if the topology changes (e.g., a new node joins, or an existing interface is torn down). A summary of these characteristics is provided in Table 8.1.

This centralized approach has both advantages and disadvantages. The obvious critique to such a design is that the Manager is a single point of failure and, in case of many connected clients, it could become a bottleneck in the local domain. There are, however, multiple considerations that mitigate these potential problems:

- The operations that the Manager needs to conduct are fairly simple both computationally and logically, with appropriate data structures ensuring they can be completed in constant or logarithmic time.
- The Manager follows a strict policy to validate messages from clients, and a deviation from the protocol causes immediate disconnection of the client.
- Manager operations are restricted to changes in the local domain topology (e.g., creation or deletion of interfaces), meaning that the Manager is not involved in the data path for messages exchanged by the clients.
- Even in the case of Manager failure, the already established peer-to-peer connections between clients will keep operating, meaning that a previously set-up connection can continue to transmit data regardless of Manager status.

The Manager is indeed still a Single Point of Failure (SPOF) for control-plane operations (new registrations, new connections, topology changes), and in this narrow sense it does not eliminate the SPOF pattern that ROS2's multicast-based discovery was designed to avoid. The key difference is one of failure scope rather than failure elimination: a ROS Master failure undermines already-running connections (as noted above), whereas an SM2 Manager failure leaves established peer-to-peer data flows unaffected, degrading the system to *no new connections* rather than *no connections at all*. This is a deliberate trade-off, accepting a narrower, data-plane-safe SPOF in exchange for avoiding the multicast-delivery reliability problems that motivated moving away from ROS Master in the first place, rather than a claim that the SPOF problem has been eliminated.

It should be noted that the current design does not include an automatic Manager re-election or failover mechanism: if the Manager process crashes or disconnects, existing peer-to-peer connections continue to operate (as noted above), but no new connections or topology changes can be negotiated until the Manager is manually restarted or replaced. Extending SM2 with a leader-election protocol (e.g., based on a lightweight consensus mechanism such as Raft, given the Manager's modest state and operation set) to support automatic Manager recovery is identified as future work.

Having a centralized management of the local domain topology also brings a number of advantages. Since all state information about the local domain is gathered in one place, it becomes straightforward at any moment to determine its exact condition or query it if needed. The presence of a central Manager relieves individual clients of the burden of keeping track of changes, as the Manager assumes responsibility for informing them whenever updates occur. This arrangement not only simplifies the protocol but also avoids unnecessary communication between nodes: they no longer need to interact with peers they have no direct interest in, as all communication setup is routed through the Manager. As a result, the need for a distributed discovery system disappears, since nodes are not required to find each other autonomously. Furthermore, the centralized approach makes it possible to host multiple local domains on the same machine, simply by deciding which Manager the clients connect to.

8.2.2 Data plane: intra- and inter-robot communication

The data plane is responsible for the actual exchange of data between processes, either within the same robot or across different robots. When two processes have established communication by creating the appropriate interfaces and interacting with the Manager, they communicate over sockets following a strict protocol that depends on the interface type. All control messages are exchanged over sockets and can potentially carry a reference to a shared memory buffer, allowing processes to share variable-size data in bulk. The shared memory buffers are created and written to by one process and then shared with one or multiple other processes. Sharing is a very lightweight operation, effectively implemented by leveraging operating system features, such as Control Message (CMSG) on Linux. The implementation of the buffers themselves also ensures that any unplanned behavior or process crash does not lead to dangling or leaking resources. After sharing, multiple processes have access to the same memory area. To prevent any erroneous or malicious operation on shared memory, the SM2 protocol requires that the process sharing the memory seal it beforehand. Sealing is an abstracted operation, based on Linux memory sealing, that prevents anyone from creating a write-access mapping of the memory, ensuring that no process can tamper with the data after it has been finalized.

Building on this abstraction, the SM2 Network Gateway (SNG) is a dedicated application that extends communication between SM2 local domains across multiple machines. Acting as a bridge, it can connect to one or more peers and selectively manage what data is exchanged and in which direction. To achieve this, the SNG provides fine-grained, configurable rulesets that allow or deny the transmission of topics and services, with options for bidirectional, incoming-only, or outgoing-only flows. This prevents unnecessary traffic, conserves network bandwidth, and limits the exposure of sensitive data. The gateway emphasizes both flexibility and security. It automatically discovers peers on the same subnet via IPv6 multicast, supports authentication of peers, and ensures encrypted communication. Quick UDP Internet Connections (QUIC) was chosen as the transport protocol because it natively provides authentication, confidentiality, and parallel independent streams, thereby avoiding head-of-line blocking and ensuring robust performance under unreliable network conditions. Within each local domain, the SNG dynamically adapts to the current topology, its configuration, and peer requests, minimizing overhead and reducing the interaction burden on local processes. By integrating seamlessly with

the SM2 Manager node, it reflects local topology changes and propagates them to remote peers when permitted. This ensures that SM2 retains its predictability and efficiency even in distributed, multi-robot deployments.

8.3 Experimental evaluation

To validate the effectiveness of the approach, and compare the behaviour of the two main technologies available in ROS2, we used two laptops, we will call A and B. The two machines are running Ubuntu 20.04 and are connected using 2.4GHz WiFi, which acts as an unreliable connection medium.

The WiFi connection exhibited moderate reliability, with a measured bitrate of 30.8Mbit/s, jitter of 0.234ms, and a packet loss rate of 4%. While the average latency remained below 15ms, occasional spikes of up to 102ms and 1030ms were observed, indicating transient disruptions. These measurements were obtained using iPerf3. All tested solutions were evaluated using their default configurations.

This section addresses the following research questions:

RQ1. How does SM2's resource consumption compare to ROS2 with the Zenoh RMW under representative robotic communication workloads (Section 8.3.2)?

RQ2. What latency does SM2 introduce relative to alternative middleware (Section 8.3.3)? The evaluation scenario (single-host intra-robot publish/subscribe communication at fixed message sizes and rates) is intentionally narrow: it isolates middleware-level overhead from confounding factors such as network conditions or multi-robot coordination, which are evaluated separately in Chapter 9 and Chapter 10.

8.3.1 Network communication tests

The main driving force behind the development of SM2 was the all-around drop in performance when introducing network communication with ROS2 and DDS-based middleware. For this reason, SM2 was compared with ROS2 (Humble) with FastDDS in a test designed to highlight the impact of the condition of the network on local communication. The test was also conducted with the Zenoh RMW for ROS2 to benchmark SM2 against a state-of-the-art ROS2 configuration, and to gather comparative data.

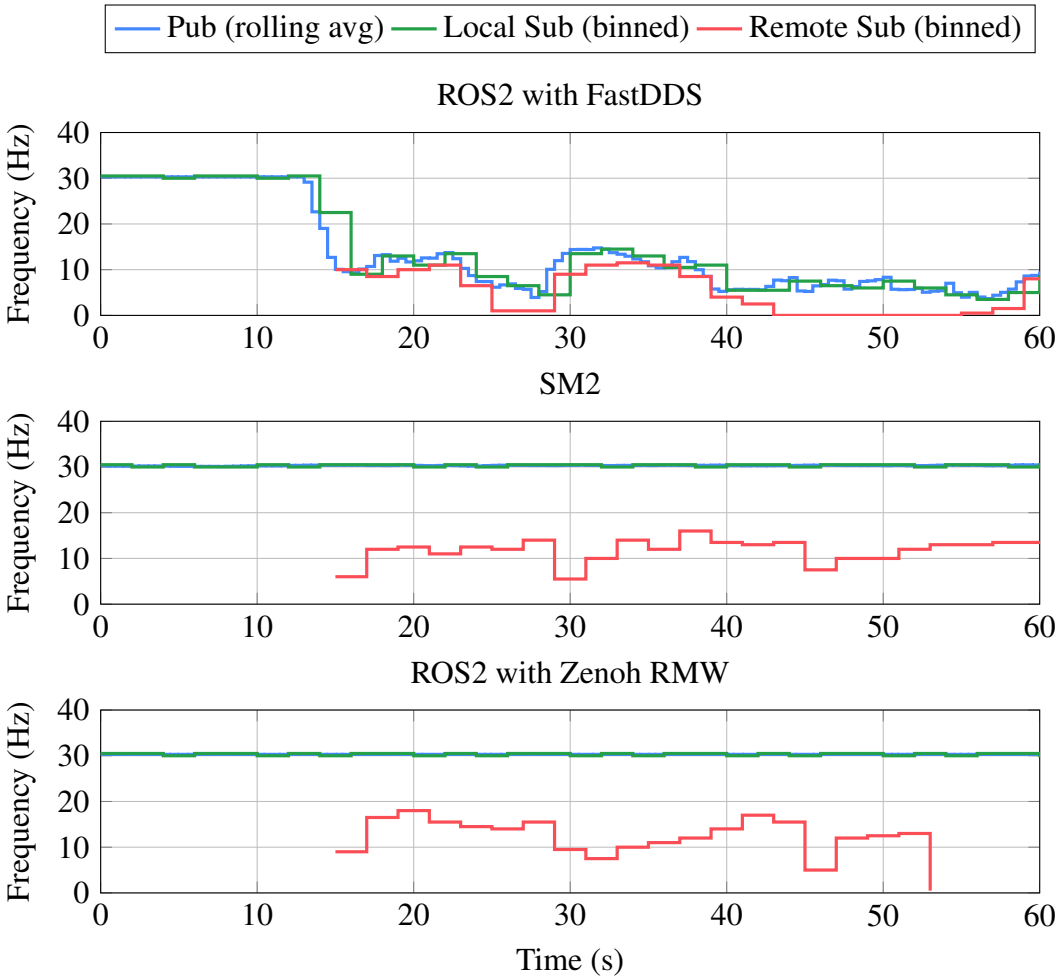


Fig. 8.2 Frequency of sent and received messages before and after introducing a remote subscriber.

The network tests are organized as follows:

1. A publisher node running on laptop A publishes large messages (1MB each) at a fixed frequency (30Hz).
2. A subscriber node is also started on A, and the frequency and latency of communication is measured, providing a baseline.
3. Another subscriber is started on B (the machine is connected over WiFi) subscribing to the same topic, after approximately 15s from the beginning of the test.

4. Metrics from all participants in the communication are recorded for later analysis.

The message size of 1MB was chosen as representative of compressed camera frames and moderate-resolution sensor payloads commonly exchanged in robotic perception pipelines, rather than as a worst-case or application-specific value. Section 8.3.2 additionally reports results at 10MB to assess sensitivity to larger payloads such as uncompressed images or point clouds.

Figure 8.2 shows the difference between ROS2 with DDS-based middleware and the behaviour of SM2 and Zenoh, which obtain very similar results. Neither SM2 nor Zenoh show any impact of remote communication on the local side and achieve comparable throughput. FastDDS expectedly suffers from a major performance penalty at the publisher, affecting all participants in the communication, both local and remote. It is interesting to note how even considering only message throughput in network communication, FastDDS manages to deliver far fewer messages, with protracted periods where no messages are delivered at all. The experiment was carried out with both reliable and best-effort QoS settings, which yielded completely analogous results.

Given the glaring issue introduced in DDS-based communication, for the following tests, we decided to limit the following tests only on SM2 and Zenoh to find out how they compare on other metrics.

8.3.2 Resource consumption

Although SM2 and ROS2 with Zenoh RMW behave similarly in the tests designed to highlight the correlation between network communication and drops in local performance, the latter takes a greater toll on system resources. The tests were run locally, using only laptop A. As evidenced by Figure 8.3, when running a publisher (10MB messages) at a frequency of 10Hz, ROS2 with Zenoh RMW uses up to four times as much memory and three times as much CPU time as SM2. CPU utilization is expressed as a percentage of a single core (100% = one core). This disparity is especially pronounced when the number of subscribers increases (i.e., moving from "1 to 1", one publisher and one subscriber, to "1 to 5", one publisher and 5 subscribers).

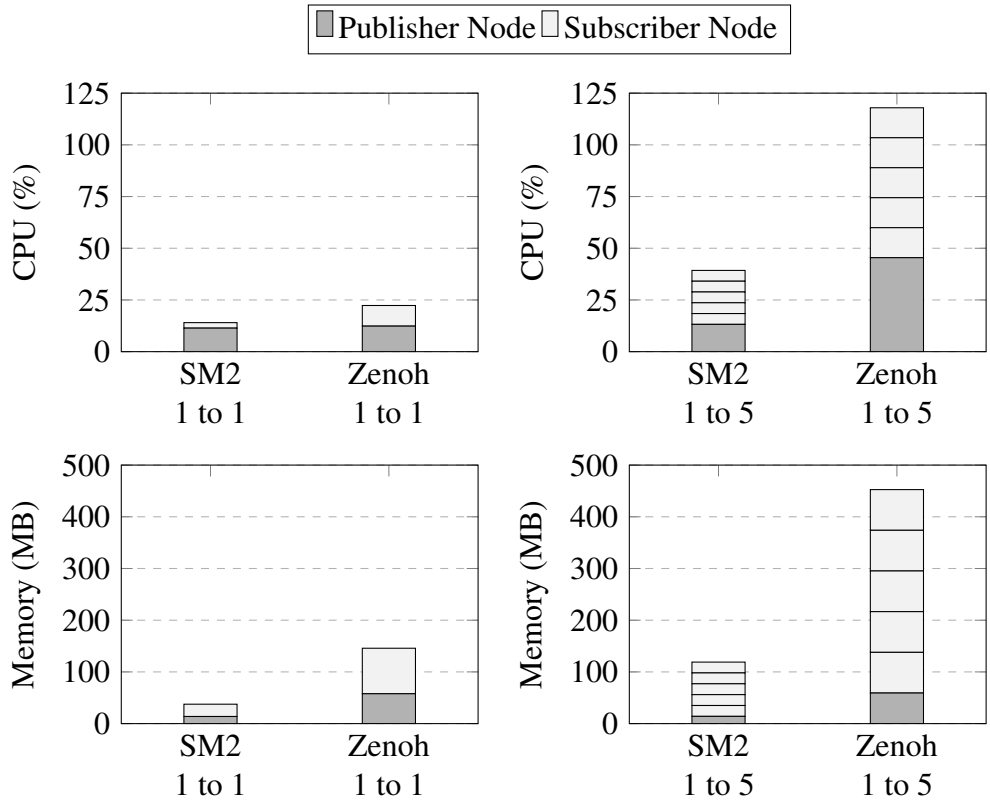


Fig. 8.3 CPU and memory usage of SM2 and ROS2 with Zenoh, in 1-to-1 and 1-to-5 configurations.

This gap can be attributed to architectural differences rather than to a single dominant factor. First, SM2's local data path is built directly on a dedicated shared-memory abstraction (Section 8.2.1), with control messages handled separately over lightweight Unix Sockets. Zenoh, by contrast, is a general-purpose pub/sub middleware that must support a broader range of transports and deployment topologies (peer-to-peer and routed, local and networked, see Table 8.1), and its local-communication path consequently carries additional serialization, key-expression matching, and routing logic that SM2's narrower, robotics-specific design avoids. Second, Zenoh's discovery and session-management layer is designed to operate transparently across both local and networked scenarios, which requires maintaining more general-purpose state and processing per message than SM2's centralized, single-purpose Manager. While this generality is a deliberate design choice that gives Zenoh broader applicability beyond robotics, it comes at the cost of higher per-message CPU and memory overhead in the specific intra-robot communication scenario evaluated here.

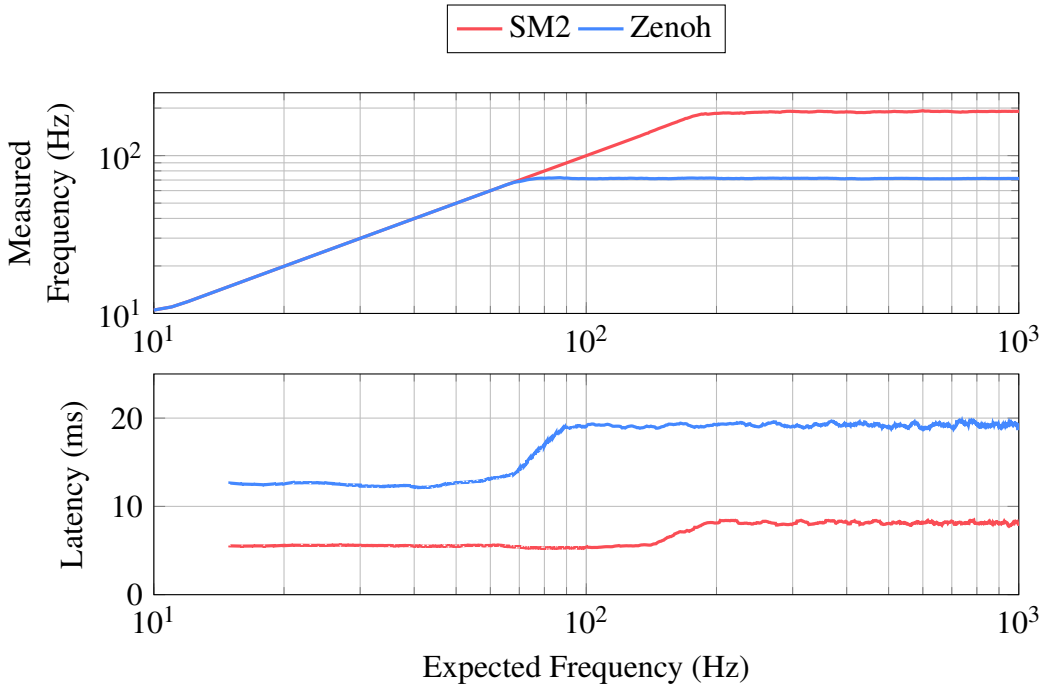


Fig. 8.4 Latency and achieved frequency at different target publication frequencies, in 1 to 5 local communication. SM2 and ROS2 with Zenoh compared. Message size is 10MB.

This difference in resource usage is significant, as it directly translates to the need for more powerful hardware to deploy similar capabilities. However, even under conditions that should not saturate system resources, it still leads to important differences in performance as we will show in the following subsection.

8.3.3 Latency

Starting from the previously described configuration, we were able to understand how Zenoh and SM2 compare in terms of latency. Figure 8.4 shows the result of a test in which message latency was measured at increasing publication frequencies. This test highlights how SM2 is capable of operating at significantly higher frequencies (with a peak frequency around 200Hz) under the same conditions, with Zenoh only able to manage a peak frequency of around 70Hz. SM2 is also capable of consistently delivering messages at lower latency, with both solutions experiencing an expected latency increase at saturation.

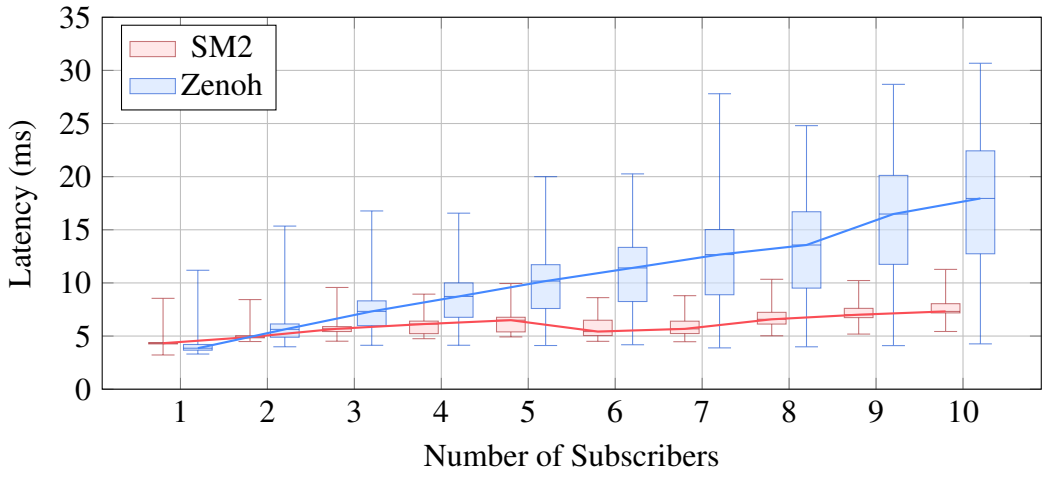


Fig. 8.5 Comparison of latency with different subscriber counts between SM2 and ROS2 with Zenoh.

All these performance differences are exacerbated when increasing the number of publishers or subscribers taking part in the communication. Figure 8.5 shows how the average latency of the messages delivered grows significantly for each added subscriber, and, as for the previous results, SM2 clearly outperforms the Zenoh implementation.

8.4 Concluding remarks

This work presented SM2, a lightweight middleware designed to ensure reliable and deterministic inter-process communication in robotic systems, even in the presence of adverse or unreliable network conditions. By decoupling local IPC from network transport, SM2 prevents the performance degradation commonly observed in DDS-based ROS2 configurations, where remote subscriptions can negatively impact local communication.

The architecture of SM2 combines a centralized management model with efficient abstractions over system-level IPC primitives, enabling predictable communication while minimizing resource usage. Furthermore, the introduction of the SNG extends this reliability to inter-machine communication, offering secure, configurable, and low-overhead connectivity through the QUIC protocol.

Experimental evaluation demonstrated that SM2 achieves comparable throughput to state-of-the-art alternatives such as ROS2 with Zenoh RMW, while consuming significantly fewer CPU and memory resources. SM2 reduced CPU by $3\times$ and memory by $4\times$ compared to Zenoh RMW. Additionally, SM2 supports higher message throughput and lower message latencies, particularly as the number of participating nodes increases. These results highlight SM2 as a practical and scalable middleware for robotics, offering usability comparable to ROS2 while delivering predictable performance and reduced overhead.

Future work on SM2 includes expanding platform support and improving serialization compatibility across architectures. The SNG could benefit from enhanced parallelism and per-connection interface handling to better isolate faulty peers.

Chapter 9

Strategies for Offloading Robotics Tasks in the Cloud Continuum

Offloading robotic computation to the Cloud Continuum introduces new challenges in maintaining the correct operation of the system when components change their execution location. In particular, robotic software modules may need to dynamically transition between local execution on the robot and remote execution on edge or cloud resources. Enabling such transitions requires mechanisms that preserve service continuity and ensure that the system continues to operate correctly with minimal disruption.

The main objective of this work is to develop an effective methodology to offload the execution of stateless ROS2 modules to edge clouds. The main requirement is to guarantee service continuity and transparency, that is, if a ROS2 system is completely or partially moved to the cloud (or vice versa), it must be able to continue working correctly, minimizing service disruption. We performed a preliminary exploration this paradigm in a previous paper [114] and in this chapter we analyse four core concepts to implement it. Through the analysis of their performance, we aim to identify the most suitable technology or approach to reach such goal.

We define *switching* as the process of transitioning the active logic of a component from one location to another. The ideal switching is fast and is undetectable by any other entity in the system that interacts with the migrated component. This capability is central to the solutions discussed in this paper, the four approaches will be analysed according to their distance from ideal switching. In switching, computation *moves*

from one location to another. However, the process whose function has changed location may not have moved as well. Especially in mission-critical settings, the old copy of the function might be kept running in the original location, but prevented from communicating with the rest of the system. This can be done to deploy a fallback-mechanism that allows the system to rapidly re-enable the old instance if the remote become unreachable. In cases in which such redundant local copy is not needed, the solution is to add a re-deployment step to the switching procedure. In this case, after having moved the computation, the old logic component is terminated, turning the switching operation into a full-fledged migration. Of the four solutions studied in this paper, two will include a re-deployment step.

It is worth clarifying the relationship between this chapter and Chapter 8. SM2 addresses low-latency communication within and between robots at the middleware level, independently of where each communicating component is deployed. The switching mechanisms introduced in this chapter operate one layer above: they govern whether a given computational task runs locally on the robot or is offloaded to a remote node, regardless of which IPC middleware the task's communication relies on. In this sense, SM2 and the switching strategies discussed here are complementary but not dependent on each other: a deployment could use the switching mechanisms of this chapter together with SM2, with ROS2/Zenoh, or with another middleware, as the switching decision concerns workload placement rather than the communication layer itself[1].

9.1 Related work

Chen et al. [115] introduced FogROS, a tool for easily deploying robot software components to the cloud via ROS, enhancing computing resources like GPUs with minimal setup. Building on this, Ichnowski et al. [116] presented FogROS2 for ROS2, enabling simple cloud and fog computing integration for robots. This platform allows for shifting heavy computations to the cloud with few script changes,

[1]This work was conducted in collaboration with Daniele Cacciabue, Francesco Aglieco, Marco Levorato, Domenico Perroni, and Fulvio Risso. It was partly supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino and Daniele Cacciabue Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative.

without altering the robot's code, focusing on VMs rather than Kubernetes for cloud interactions. Expanding on these concepts, Chen et al. [117] later developed FogROS2-SGC, furthering FogROS2's capabilities to connect robots across different locations securely and efficiently, without requiring code changes. Similarly, Anand et al. [118] introduced an algorithm designed to mitigate the constraints of serverless computing, such as communication and bandwidth issues, employing various work-sharing strategies to enhance cost efficiency and reduce execution time.

Doan et al. [119] proposed a framework that separates MEC application design into processing and state management, using a distributed key-value store to ensure seamless service continuity. This design enables state synchronization across MEC servers and anticipates user handovers between MEC nodes. Its modular, containerized approach enhances its practical applicability in MEC settings.

Machen et al. [120] tackled the challenge of minimizing service downtime and overall migration time in mobile edge clouds. They developed a layered framework that breaks down cloud applications into multiple layers, allowing for the transfer of only the missing layers to the destination. This framework is applicable to both VMs and containers and can be implemented with existing tools.

Chebaane et al. [121] introduced a method for offloading time-sensitive tasks from the initiating application to nearby Fog nodes using Docker containers and Checkpointing. This approach results in a layer-oriented framework that limits offloading to essential steps within just two message exchanges. The strategy assumes that offloading occurs solely from the vehicle hosting the application to an adjacent Fog node.

Some researches have concentrated on the live migration of VMs [122, 120], transferring all VM data during runtime. Conversely, other research has explored the live migration of containers [120, 123], emphasizing the movement of containerized applications without interrupting their operation.

Our approach differentiates from the above solutions because it does not handle the case of stateful migration, moreover, it analyses specifically the migration of ROS2-based services, during execution. Unlike Machen et al. [120], our approach does not tackle primarily a client-server communication paradigm, but mainly focuses on a client-subscribe one, which ROS2's middleware is based on. This study differentiates from FogROS2 [116] due to the selection of Kubernetes and containers instead of VMs, which allows for a lower resource footprint, easier management and

the ability of bringing the the benefits and practices of cloud-native development to robotics.

9.2 Switching architectures

Within the scope of implementing switching capabilities using ROS2, we identify four potential approaches. Solutions are presented side by side, highlighting that the effectiveness of a strategy varies with the scenario, making it impossible to choose a one-size-fits-all option in advance. A key difference in the solutions proposed stands on the fact that two of them switch computation by redeploying the Pod, the other two rely on preemptively deploying multiple copies of the Pod in different locations and only keeping one of them active at the same time. In our discussion we will refer to two Kubernetes clusters: one designated as *robot* and the other as *cloud*.

9.2.1 Switching with Redeployment

In addressing the challenge of efficiently allocating ROS2 Nodes within switching architectures, the pivotal issue becomes their placement. This dilemma can be reframed as a scheduling challenge, focusing on identifying the most appropriate Kubernetes node and cluster for hosting the ROS2 Node. The Switching with Redeployment (SwR) architecture emerges from this concept, entrusting a scheduler with the task of initially deploying the ROS2 Node and subsequently executing a rollout to relocate it to another location as required. Initially, the scheduler operates in a basic capacity, merely transferring the ROS2 Node between locations without incorporating any metrics or contextual analysis. The responsibility for managing the Pod lifecycle is transferred to Kubernetes. Therefore, once the scheduler has assigned the ROS2 Node to a specific cluster node, Kubernetes assumes control over all further processes, adhering to its standard operational procedures.

9.2.2 Switching with Redeployment and Network Policies

Building upon the SwR model, the Switching with Redeployment and Network Policies (SwRNP) architecture introduces an additional step following the scheduling phase. After a new ROS2 Node instance is scheduled, enters the *Running* state, and

the previous node transitions to *Terminating*, the scheduler deploys a NetPol. This policy blocks all outgoing traffic from the old ROS2 Node, minimizing the overlap of DDS messages between the old and new Nodes. The introduction of NetPol ensures that the outgoing messages from the terminating ROS2 Node do not affect the newly configured Node. After deleting the old ROS2 Node, the NetPol is removed, as it is no longer necessary.

9.2.3 Switching with Network Policy

Th Switching with Network Policy (SwNP) solution is not based on redeployment. It starts from the condition in which more than one copy is active at the same time, one in the *robot* cluster and one in the *cloud* cluster. By default, only one of the two instances is active at the same time. This is implemented using a NetPol to block any outgoing communication from the disabled instance, preventing it from communicating. When the switch needs to be performed, the inactive Pod needs to be enabled and the disabled Pod enabled, which allows for effectively switch the active logic from one location to another. Since this solution does not leverage the existence of the Kubernetes scheduler, enabling and disabling of a Pod needs to be performed by a custom-built orchestrator, which communicates with the Kubernetes API server to enforce the NetPol.

9.2.4 Lifecycle Node

The Lifecycle Node (LCN) solution, still not based on redeployment, is based on a ROS2 feature called Lifecycle Nodes[2]. The latter are a special type of Node in ROS that can be managed through four primary states.

- **Unconfigured**: initial state of the Node after instantiation; the node is ready for configuration but not yet operational.
- **Inactive**: state of a Node that is not performing any processing; its main purpose is to provide a state where a Node's behaviour can be changed (re-configured) without requiring it to be in active state.

[2]https://design.ros2.org/articles/node_lifecycle.html

- **Active**: once ready to be fully functional, the Node transitions to the active state; at this point, it has all the behaviours of a standard ROS Node (it responds to service requests, reads and processes data, and produces output).
- **Finalized**: terminal state; the only transition from here is to be destroyed, which frees up the Node's memory.

A node can be asked to change state using proper service-based interfaces exposed by every Lifecycle Node. A custom-built orchestrator can implement the switching algorithm using such services, suitably activating or deactivating node instances. A deactivated node executes less code, which results in a lower battery consumption compared to the the solutions based on NetPol. This approach can only be used with ROS2 Lifecycle Nodes, making it a less portable solution that relies on ROS2-specific features instead of Kubernetes.

9.3 Implementation

The experimental setup was designed to fully utilize the capabilities of Zenoh, Kubernetes, and Liqo. This architecture encompasses two VMs hosting Kubernetes clusters, labelled as the *Robot* and the *Cloud*, and it is depicted in Figure 9.1. Each VM is configured with 4 CPU cores and 8GB of RAM, and shares a common virtualized Ethernet network. Within this setup, ROS2 Nodes communicates using DDS middleware, which relies on UDP multicast traffic to implement peer-discovery. Both clusters were configured using K3s, a lightweight Kubernetes distribution, with Cilium serving as the CNI. This configuration was specifically chosen due to K3s' default CNI (Flannel) supporting multicast within the cluster. In case of managed Kubernetes, the choice of CNI is up to the cloud provider and not to the single tenant. By taking the most restrictive option, this study ensures that the solution is always applicable, independently of the chosen CNI supporting multicast or not.

The *robot* cluster presents the same architecture as the *cloud* cluster, it employs the same CNI and the same Pod organization. This mirrored architecture across both clusters allows for deployment transparency, since the redeployed Pod does not need to be modified depending on the nature of the CNI where it is running. This simplification allowed for the use of a simpler redeployment logic, which does not need to take into account details such as the CNI installed in each cluster. The

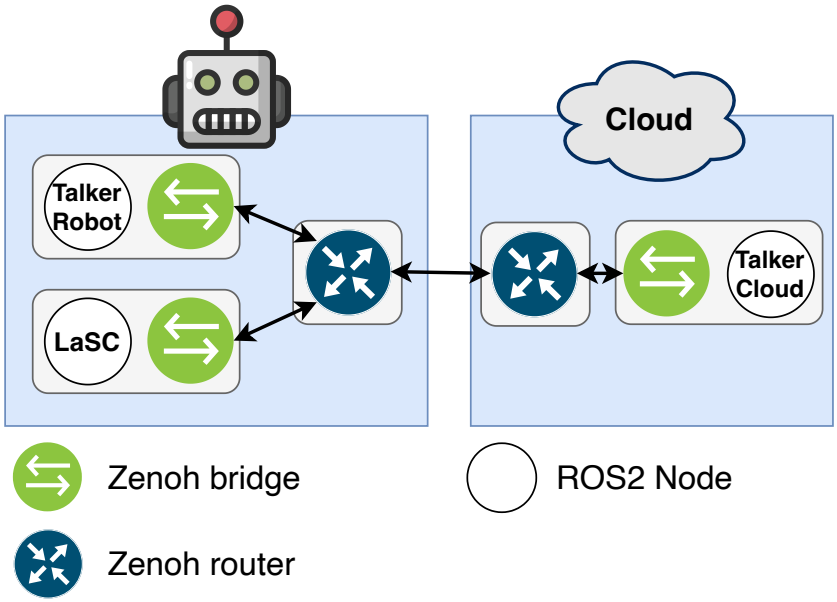


Fig. 9.1 Architecture interconnecting cloud and robot clusters.

challenge of integrating the presented solutions in clusters with heterogeneous CNIs is left for future works.

To circumvent the absence of multicast, a Zenoh-based overlay network was devised, which allows discovery traffic to be forwarded to any other ROS2 Node, bypassing limitations due to the CNI. Our approach integrates the Zenoh Plugin ROS2DDS[3], functioning as a bridge within a sidecar container in each Pod, alongside the main container running the ROS2 Node These bridges connect to the central Zenoh Router within the cluster, facilitating seamless DDS communication between ROS2 Nodes across different clusters.

Liqo was installed and used to establish a peering from the *robot* towards the *cloud* cluster. Liqo enables the *robot* cluster to easily use resources from the *cloud* cluster: it creates a namespace on the *cloud* cluster and it is managed transparently, as if it was local. This greatly simplifies the management of multiple clusters, eliminating the need for exposing Zenoh routers through a Load Balancer, managing IP address changes, and instead relying on using a Fully-Qualified Domain Name (FQDN) for the service in the remote cluster.

[3]https://github.com/eclipse-zenoh/zenoh-plugin-ros2dds

9.3.1 Test environment

The basic structure of the test environment consisted of two ROS2 Nodes, namely *talker* and *listener*. The *talker* publishes a message every 100ms on a topic, in the form of *location: counter*, where *location* is the Kubernetes cluster on which the Pod is currently scheduled (e.g., *robot*) and *counter* is an increasing number, so each message will have the counter of the previous message increased by 1. The *listener* subscribes to the same topic of the talker and waits for messages. When it receives a message, this is written to a InfluxDB instance which records it together with the relative timestamp.

The *Switch Controller (SC)* is a component responsible for triggering or enacting the switch transition. When the system starts, the SC sets the active talker to be on the robot and it listens on the talker topic to ensure that there is only one instance of talker active at the given time. When such condition is met, it triggers the switching procedure, recording the event in the InfluxDB instance. One important implementation decision regarding this component was to decide a convention regarding the order in which a Node is enabled or disabled. This choice matters for the SwNP and LCN case, because they do not rely on the Kubernetes scheduler to execute their logic. Since the default behaviour used by Kubernetes to migrate a container is to first start a new copy on another cluster node, then to terminate the old copy, the choice was to implement the SC to first enable the currently disabled node, and only afterwards to disable the previously active one. The SC is also able to monitor the state of the Pod, checking when it transitions to *Running* or when the Pod is effectively deleted. The actual implementation resulted in the merging of the *listener* and the SC into the same ROS2 Node, which we will now refer to as *Listener and Switch Controller (LaSC)*. Such Nodes were deployed on the architecture in Figure 9.1.

9.4 Experimental evaluation

To assess the duration needed for the architectures to accomplish the switching, we identified three distinct intervals that offer insights into the swiftness of the switching procedure. These intervals illustrate each architecture's capability to either

switch ROS2 Nodes or to activate/deactivate them. The intervals are characterized as follows.

- **Time *t1***: measured from the moment the switch request is initiated to the point where the new ROS2 talker begins to transmit data. This encompasses the period from issuing the switch command to the instance the listener detects the initial message from the new talker (Figure 9.2a).
- **Time *t2***: defined as the duration from the first message sent by the new talker to the moment the listener ceases to obtain data from the previous talker. It can assume either a positive value (Figure 9.2b) or negative (Figure 9.2c).
- **Time *t3***: time interval from the last message sent by the old talker to the time when the latter is fully deleted.

This section addresses the following research questions:

RQ1. How much time does each switching strategy require to redirect a workload from local to remote execution, and vice versa (Sections 9.4.2 to 9.4.4)?

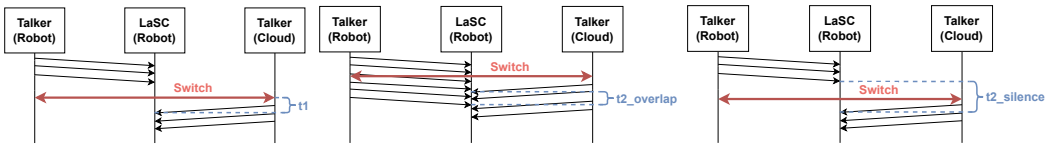
RQ2. Which switching strategy offers the best trade-off between switching speed and implementation complexity for the scenarios considered (Section 9.4.5)?

9.4.1 Metrics collection

The relevant metrics have been collected using an InfluxDB instance. InfluxDB has been selected as the database of choice because of its time-series structure which allows not only to register events, but also the time at which they have occurred. The gathered data has then be processed offline, which allowed for decoupling the data collection from the data analysis phase.

The LaSC inserts into InfluxDB the events related to (i) the arrival of a new message, (ii) the time at which the switching procedure is started, (iii) the time at which the talker Pods change their state to *Running* or are effectively deleted.

Data was gathered through 200 measurements and it has been organized in two InfluxDB buckets. The *listener metrics* bucket stores information regarding the direction of the switching (i.e. robot-to-cloud or cloud-to-robot), the messages sent by each talker, and the measurement iteration (e.g., 0, 1, 2, etc.). The *SC metrics*



(a) *t1* example. (b) $t2_{overlap}$ example. (c) $t2_{silence}$ example.

Fig. 9.2 Definition of *t1* and *t2*; *t2* can be positive ($t2_{overlap}$) or negative ($t2_{silence}$).

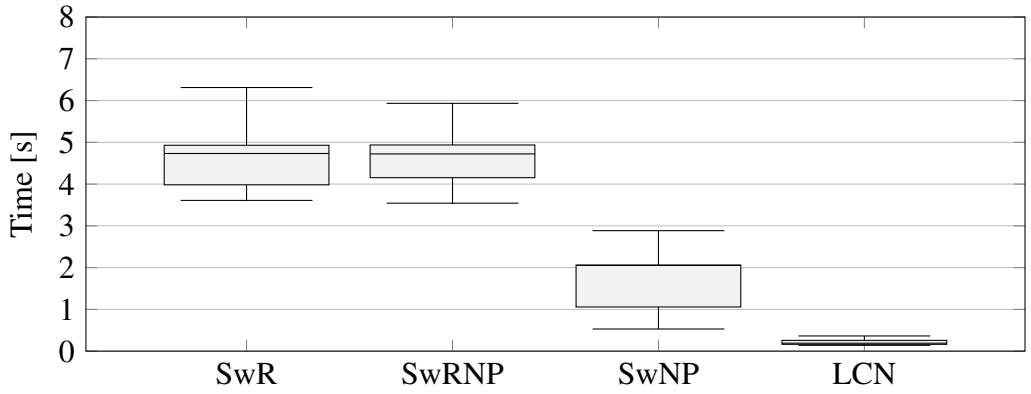
bucket is dedicated to track the other metrics we need to compute the final times *t1*, *t2*, and *t3*. These metrics are the timestamps related to the switching issued commands, when the new ROS2 Node is running and when the old ROS2 Node has been deleted.

After the metrics have been collected, the data was processed to compute the times *t1*, *t2*, and *t3* for all of the four architectures. Such data was gathered with the intent of allowing to easily compare the differences in switching time for each solution. The computed times *t1*, *t2*, and *t3* have been drawn on boxplot graphs, to easily compare them.

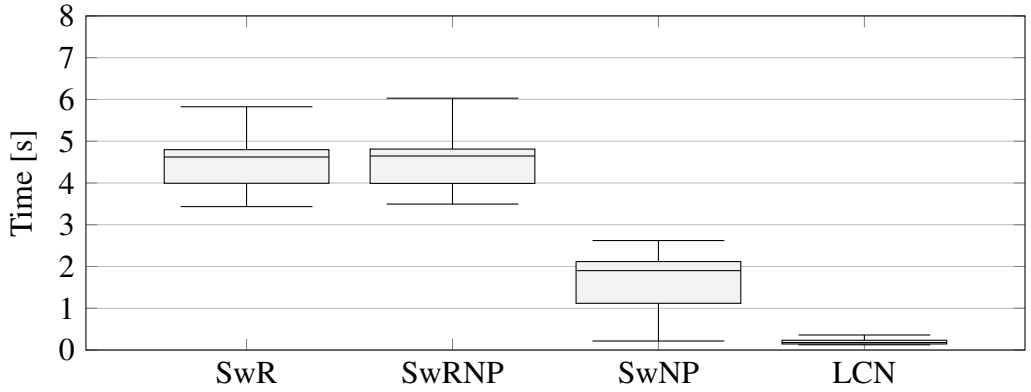
9.4.2 Time *t1*

The results regarding the time *t1* for the robot-to-cloud and cloud-to-robot case are shown in Figure 9.3a and Figure 9.3b respectively. The data shows some slight differences between the two transitions, but the results show similar performance independently of the messages direction.

Unsurprisingly, the two cases with redeployment (SwR and SwRNP) perform worse as far as time *t1* is concerned, this is due to them requiring time to deploy the new talker Node on the other cluster. On the other hand, the two solutions with redeployment (SwR and SwRNP) and the one with only NetPol (SwNP) show how the LCN solution performs dramatically better, showing a clear downside of using NetPol when the need is to quickly restore the network traffic. Considering *t1*, the most performing solution is the LCN, which, considering the average values, allows for a sensibly faster switching time reduced of at least 85% compared to the other solutions.



(a) $t1$ Robot to Cloud.



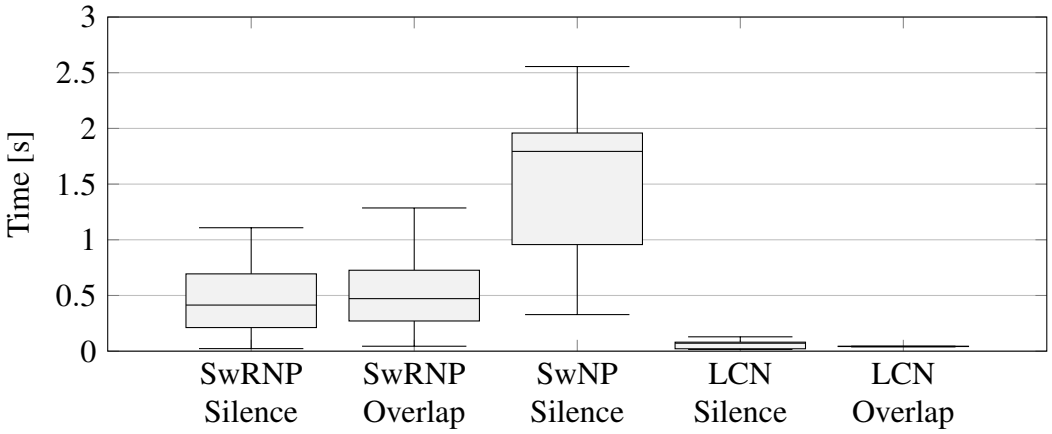
(b) $t1$ Cloud to Robot.

Fig. 9.3 Time $t1$.

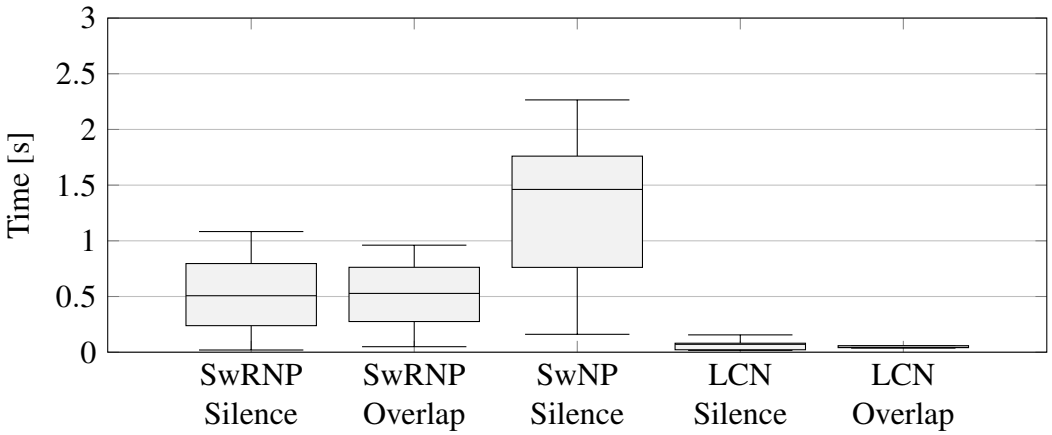
9.4.3 Time *t2*

Given that both the default behaviour of the Kubernetes scheduler and the implementation of the SC prioritize the activation of a new node over disabling an old one, it was natural to assume that the new talker would be able to send a message before the old talker was terminated, resulting in strictly-positive samples of *t2* signalling an overlap condition.

The results depict a different picture and only the SwR case conformed to the expectations. Both overlap and silence have been measured in the SwRNP. In the SwNP case, only one sample of overlap was measured out of 400 measurements comprising both the robot-to-cloud and cloud-to-robot directions, effectively showing how, the previous hypothesis does not apply to this type of switching. The LCN case showed both overlap and silence as well, but in this case, since the activation and



(a) $t2$ Robot to Cloud.



(b) $t2$ Cloud to Robot.

Fig. 9.4 Time $t2$.

deactivation of the Lifecycle Nodes was made asynchronously, some kind of silence was expected, due to the not deterministic nature of such kind of RPC calls.

Due to the presence of both positive and negative values, the time *t2* was split into two subcategories respectively called $t2_{overlap}$ and $t2_{silence}$. The boxplot graphs in Figure 9.4, show the relevant silence and overlap statistics. The SwR overlap case (there was no silence in for SwR) has been excluded from the graph as it showed a median value of $(29.9084 \pm 0.5541)s$ for the Robot to Cloud and $(29.412 \pm 0.6144)s$ for the Cloud to Robot. Being this value significantly higher than all the ones from the other solutions, it was not included in Figure 9.4 with the purpose of better showing the difference in performance among the remaining solutions. The SwNP overlap boxplot is missing because it consisted of only one event (outlier).

Another unexpected result shown by this graph is the disappointing performance of the SwNP case which was expected to perform similarly to SwRNP. One possible explanation for this result is that Kubernetes takes significantly less time to apply a NetPol than to remove it, thus the SwRNP, which relies on NetPol only to disable the node that is scheduled for termination, is not affected by the time required to remove it.

This hypothesis also explains why, in our measurements, the SwNP case does not show any $t2_{overlap}$, further investigation on this topic will be left for future works, from the graphs shown, the LCN case clearly resulted in being the most performant also with respect to the *t2* parameter.

9.4.4 Time *t3*

Since *t3* is the time interval between the last message of a talker to its actual deletion, it is a measure that bears meaning only for the switching cases that include a redeployment step, i.e. SwR and SwRNP. Since LCN and SwNP never delete the old talker but only disable it, they will not be considered further in this subsection. The main result derived by this interval consists in quantifying how, the application of a NetPol on the old talker, allows for a reduction in the amount of unnecessary messages sent on the DDS multicast network. In our metrics, we saw a time of $(1.2455 \pm 0.3043)s$ Robot to Cloud and $(1.2315 \pm 0.2734)s$ Cloud to Robot for the SwR, and $(31.3253 \pm 0.3622)s$ Robot to Cloud and $(31.2824 \pm 0.3457)s$ Cloud to Robot for the SwRNP. This difference is due to the fact that in the SwR case, the Pod is deleted practically immediately after the last message arrives. In the SwRNP case the Pod is first silenced by the NetPol and then deleted afterwards, thus the time *t3* increases.

9.4.5 Discussion of results

The results show that the Lifecycle Node architecture outperforms all other options evaluated, which makes it the most promising solution to enable a swift and efficient transition of a ROS2 Node from one cluster to another. In hindsight, given the reached performance, further tests may be required in order to test the maximum

publishing frequency that can be supported by the Lifecycle Nodes solution while keeping the *t1* and *t2* metrics as low as possible.

As far as the solutions which incorporate a redeployment phase, we hypothesize that the optimal switching with redeployment solution would closely resemble SwRNP, but would utilize Lifecycle Nodes to disable the old instance instead of NetPol. This kind of solution would be able to leverage the Kubernetes scheduler to move the deployment from cluster to cluster and would leverage the very good Lifecycle Nodes performance to minimize any problems regarding the presence of either overlap or silence.

9.5 Concluding remarks

Our study has made significant strides in evaluating the performance of different methods for switching stateless, ROS2 computation from one Kubernetes cluster to another using a reference architecture based on Liqo, Zenoh and Kubernetes. We also shown how integrating a method to halt communication from the old Pod can be employed in ROS2 use cases to mitigate the issue of duplicate messages, allowing a stateless workload to be effectively redeployed without sacrificing business continuity. Our findings provide a measure of switching speed of ROS2 Nodes from one cluster to another using various technologies.

However, several topics warrant further investigation. These include quantifying the time needed by Kubernetes to enforce a NetPol once created and to delete it. By understanding what makes the solution based on NetPol less efficient, it may be possible to make it a suitable competitor to the LCN solution, as it allows for the switching algorithm to be applied to other applications other than ROS2-based ones. The four solutions analysed in this paper are not meant to be exhaustive and there may be other suitable technologies and ideas to implement the switching operation. The identification and the analysis of them is another research path that deserves to be taken.

Chapter 10

Offloading Robotic Tasks in Unreliable Network Environments

As discussed in Chapter 7, offloading computation to edge or cloud infrastructure can significantly extend the capabilities of resource-constrained robots. However, the benefits of this approach are strictly limited by the reliability of the underlying wireless network [90]. Mobile robots deployed in dynamic, mission-critical environments [124, 125], such as autonomous delivery, industrial inspection, healthcare assistance, and disaster response, face particularly severe challenges in this regard.

The problem of intermittent network connectivity in MEC- or fog-enabled settings has been studied in literature, mostly accounting for end-user mobility in supporting the handover process. However, robotics applications have much more stringent requirements as they typically operate in safety-critical environments requiring (near) real-time feed. This problem is still deeply unexplored. To this end, in this chapter, we investigate the trade-offs between local and remote execution in mobile robotics, focusing on how network unreliability impacts performance and safety. Conceptually, the superior processing capability at the edge provides enhanced situational awareness, allowing robots to operate and move much more precisely; however, the unpredictability of remote streams may lead to unexpected (and catastrophic) outcomes if not managed properly. While modern low-power ARM Systems on a Chip (SoCs) have substantially narrowed the gap for many workloads, inference for vision-based perception models still benefits markedly from dedicated GPU acceleration, which most mobile robotic platforms cannot accommodate within

their size, weight, and power budget: this is the specific gap that motivates offloading in the scenarios considered in this chapter.

Aligned with our goal, this chapter proposes: (*i*) A hybrid approach leveraging both local (on the robot) and remote (on the edge/cloud) processing, cleverly chosen by a smart switching component referred to as the *Multiplexer (MUX)*. The MUX acts as a proxy and collects both local and remote application streams, dynamically switching between them to ensure a safe (local) fallback option when the network condition worsens, while providing more intelligent data input when the remote source is available. (*ii*) A companion component, called *Lifecycle Orchestrator (LO)*, which disables local instances when remote execution is stable for a reasonable time window, reducing energy consumption without compromising safety and reactivating it when the network worsens. (*iii*) An overall architecture based on ROS2 that allows the previous components to operate, particularly an extension to the RMW layer, referred to as *Timestamp Tracking*. This extension adds two additional timestamps to each message: one indicating when the message is generated at the start of the processing chain, and another recording when it is emitted by the most recently executed processing module. This enables correct actuator input under strict timing constraints, even in the presence of highly variable and unstable network conditions.

We develop a simulation framework using ROS2, Gazebo [126], and Docker Compose, incorporating real-world latency and packet loss traces to evaluate system performance in various scenarios. Through extensive simulations, we demonstrate that our architecture significantly improves robustness and energy efficiency compared to traditional offloading approaches, making it suitable for deployment in challenging environments with unreliable connectivity. Finally, a working implementation of the proposed approach on an RB-SUMMIT[1] mobile robot by Robotnik Automation S.L. further strengthens the applicability in real-life settings[2].

[1] https://robotnik.eu/products/mobile-robots/rb-summit/

[2] This work was conducted in collaboration with Daniele Cacciabue, Stefano Galantino, and Fulvio Risso. It was partly supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). This research was conducted as part of Jacopo Marino and Daniele Cacciabue Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative. Finally, I would like to thank Sebastián Montoya Zuluaga for his support.

Table 10.1 Comparison of State of the Art solutions.

Extension	Comm. Type	Fault Tolerance	Latency Handling	Evaluation Scope
FogROS2-FT [127]	Service	Replicated servers, first response	No latency awareness	Ethernet, no packet loss
FogROS2-PLR [128]	Service	Multi-path request routing	Fastest path selection	Ethernet, fixed delay only
FogROS2-LS [129]	Service	Dynamic server selection	Anycast latency-aware routing	No packet loss, no disconnections
FogROS2-SGC [117]	Topic	Secure global connectivity	DDS-agnostic, encrypted streams	Focus on security, not resilience

10.1 Related work

Recent research has explored how robotics can benefit from integration with distributed computing paradigms. Pujol et al. [95] present a holistic overview of fog robotics, emphasizing the mutual benefits between robotics and fog computing. They highlight advantages such as reduced latency, improved privacy, greater autonomy, and enhanced robustness, while also proposing that mobile robots themselves can act as dynamic mobile fog nodes. At the same time, they identify open challenges in orchestration, scalability, and machine learning that must be addressed to realize this integrated fog–robotics ecosystem fully.

Some initial work on ROS computation offloading has been presented in FogROS [115], FogROS2 [117], and Cloudroid [130], which provide frameworks for transferring computation to remote servers. In particular, FogROS2 has been extended in multiple directions to improve cost efficiency, fault tolerance, and network reliability for cloud robotics. FogROS2-Config [131] introduces a toolkit that assists roboticists in selecting cost-effective cloud servers for ROS2 applications. It models the trade-offs between latency and cost under user-defined constraints, supports multiple cloud providers, and requires no changes to the application code. K. Chen et al. [127] proposed FogROS2-FT, a fault-tolerant extension that replicates ROS2 services in multiple cloud servers. By returning the first available response from replicas, it

ensures reliable, low-latency performance while enabling the use of cost-efficient spot VMs. However, it relies on stateless, service-based communication and requires no modifications to existing robot code. This design choice excludes topic-based publish/subscribe scenarios, thereby limiting its applicability. Indeed, communication in modern robotics applications mostly follows an asynchronous producer-consumer model, where message flows are not necessarily synchronized and individual messages are not intrinsically related to one another. Moreover, reliance solely on remote computation fails to account for scenarios in which local processing is required, such as complete disconnection from cloud providers. To address latency variability, FogROS2-PLR [128] sends requests through multiple independent network interfaces to replicated servers, selecting the fastest response. This reduces long-tail latency and improves responsiveness in unreliable network conditions, again without requiring application changes. However, similarly to FogROS2-FT, it targets ROS2 services rather than topic-based communication and does not consider local computation, thus failing to address scenarios such as complete disconnection from cloud providers. FogROS2-LS [129] extends the framework with latency-sensitive service selection based on Anycast routing. Mobile robots are dynamically connected to the most suitable service replica according to real-time latency constraints, without modifications to the application. Although the approach demonstrates low additional latency when connecting to edge servers, its evaluation does not cover more severe failure modes such as packet loss, complete disconnections, or server crashes. The experiments were carried out with Ethernet connections, and wireless scenarios (e.g. WiFi, 5G) were only approximated by adding fixed delays, limiting the realism in mobile settings. Finally, FogROS2-SGC [117] focuses on secure and scalable global connectivity, allowing ROS2 mobile robots to communicate in disjoint networks and geographic locations. It introduces globally unique identifiers, encrypted communication, and a zero-copy architecture, while supporting topic-level access control in a DDS-agnostic manner.

In general, these extensions to FogROS2 demonstrate significant progress toward making ROS2 applications cloud-ready by addressing cost optimization, fault tolerance, latency reliability, and global connectivity (Table 10.1). However, most frameworks remain primarily focused on service-based communication, leaving topic-based publish/subscribe messaging, which underpins many robotics applications largely unaddressed. In addition, their evaluations often overlook failure scenarios such as packet loss, disconnections, or server crashes, which are common

in mobile and wireless deployments. These gaps highlight the need for complementary mechanisms that can ensure continuous operation when connectivity is unstable, motivating approaches like our proposed MUX component.

Beyond cloud offloading frameworks, a complementary line of research investigates *when* computation should be offloaded rather than executed locally. Energy-aware approaches [132] model the trade-off between local and remote execution in terms of energy consumption, jointly optimising the execution configuration and the choice of wireless access point. However, these methods assume a stable network connection and do not explicitly address scenarios in which offloading becomes unreliable or infeasible.

A related body of work tackles the problem of maintaining control continuity when the remote connection degrades. Redundancy-based approaches [133, 134] keep a local backup of the controller that is activated when the remote link becomes unavailable. While effective against complete disconnections, these binary switching schemes do not account for subtler failure modes such as intermittent delays or partially degraded links. Prediction-based strategies [135] attempt to anticipate drops in network quality and proactively switch between remote and local controllers. Although more adaptive, they rely on controller-level switching driven by predicted network conditions and do not reason about the timeliness of individual messages.

Our approach differs from and extends these prior efforts in several respects. Unlike redundancy-based methods, we combine local and remote control providers through a message-level fallback mechanism: the MUX injects default commands whenever fresh data from either provider is missing, enabling graceful degradation even when both sources experience transient failures. Unlike prediction-based strategies, we base provider selection on message-level validity and timing constraints rather than on forecasted network state, allowing fallback decisions even when a controller is nominally reachable but its output is no longer timely. More broadly, our contribution advances the state of the art along several dimensions:

- **Network impact evaluation**: we analyse how packet loss, computation delay, sensor frequency, and latency affect robotic performance, establishing both the motivation and the baseline for our solution.

- **Topic-based communication**: we operate on a publish/subscribe model, which underpins most modern robotics applications yet is rarely addressed by existing offloading frameworks.
- **Adaptive lifecycle management**: our LO dynamically deactivates and reactivates local computation nodes based on stream reliability, improving energy efficiency without sacrificing availability.
- **Real-world trace emulation**: we evaluate the system using latency and packet-loss traces collected from a physical robot, enabling realistic simulation of network dynamics.
- **Message validity and timing assurance**: the MUX assesses message freshness and origin timestamps to guarantee safe fallback behaviour under asynchronous or degraded communication.

10.2 System architecture

To ensure robust and safe operation of mobile robots under unreliable network conditions (e.g., packet loss or variable delay), we designed a modular architecture composed of four main components: the *Navigator*, the *MUX*, the *LO*, and the *Message Metrics Collector (MMC)* (see Figure 10.1).

At the core of the architecture is the Navigator, responsible for generating control commands from sensor data. It uses the robot's sensors to detect static obstacles and dynamically adjusts its speed to avoid collisions, replicating a simplified navigation control logic. The Navigator processes each message and moves the robot according to the following logic, applied each time sensors provide new data:

1. If no obstacle is detected, the robot moves uninterrupted.
2. If an obstacle is detected, it continues to move forward at the highest possible speed (including acceleration if the obstacle is sufficiently far) that still allows it to stop safely before a collision.
3. When the robot reaches the minimum distance required to stop before colliding, it begins to brake at its maximum deceleration.

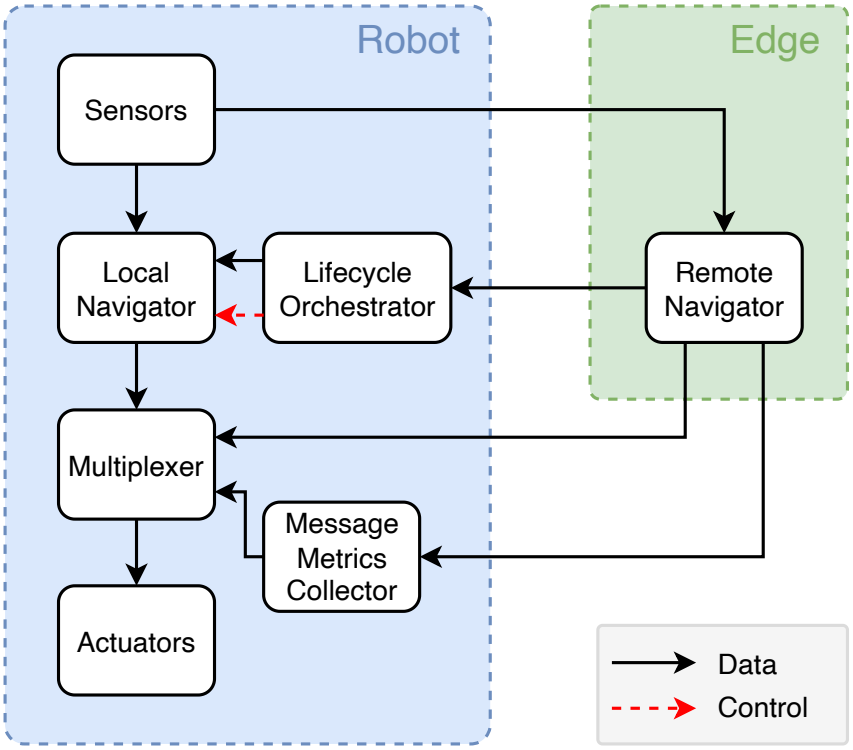


Fig. 10.1 System high-level architecture.

4. Once the robot stops, it starts turning from the incidence angle toward the calculated reflection angle. After completing the turn, it cycles back to the first point.

Two instances of this component run simultaneously: one locally on the robot and one remotely on an edge server leveraging the same algorithm, but with different processing capabilities e.g., in terms of time needed to process the camera image and detect possible obstacles. The local instance guarantees autonomy when connectivity degrades, while the remote instance offers reduced processing delay and access to more powerful computing resources. Each Navigator publishes velocity commands to a dedicated topic. It is worth noticing that although the proposed architecture is based on simplified navigation logic, it can be easily extended to include enhanced obstacle avoidance logic without compromising the validity of our approach.

The MUX is the component responsible for deciding which navigation stream should be forwarded to the robot's actuators at any given moment. It monitors the freshness and reliability of each input stream by evaluating message timestamps and transmission regularity. When the remote stream is stable, it takes precedence,

leveraging the reduced computational delay provided by the edge server. When communication deteriorates, the MUX transparently switches to the local stream or, if necessary, to a default fallback command that stops the robot to prevent collisions. The stream switching operation is extremely critical. There is no guarantee that local and remote streams agree on how to achieve the same objective, and switching too frequently may cause conflicting commands that slow the robot's progress. To avoid this issue, switching between the local and remote streams must be performed carefully, taking into account the priority and reliability of each stream.

To further optimise resource utilization, the LO oversees transitions of the local Navigator between two different states: *active* and *inactive*. When the remote Navigator is deemed reliable (e.g., the robot hovers near the access point), it deactivates the local instance (i.e., sets it to *inactive* state), thereby reducing redundant computation and power consumption. Conversely, it promptly transitions the local Navigator to the *active* state when network degradation is detected. This mechanism allows dynamic adaptation to changing network conditions without compromising safety. Specifically, the switching logic is determined using two thresholds, both configurable at runtime depending on the safety-criticality of the environment:

- **Offloading Threshold**: the number of consecutive packets that must be successfully received from a source for it to be considered reliable.
- **Fallback Threshold**: the number of consecutive packets missed from a source before it is considered unreliable.

The Offloading and Fallback Thresholds are defined in terms of packet loss, where a packet is considered lost if no new packet is received within a specified time window. This window is computed based on the interval between two consecutive messages. Further details are provided in Section 10.3.5.

The current thresholds deliberately consider only consecutive packet-loss run length, chosen for its simplicity and low computational cost on resource-constrained robotic platforms. Incorporating additional signals such as round-trip latency, jitter, or duplicate/out-of-order packet rates could provide a more complete picture of link degradation and potentially trigger more timely or more conservative switching decisions. Extending the switching logic to a multi-metric decision rule, rather than packet-loss run length alone, is identified as a promising direction for future work, complementary to the sensitivity analysis discussed in Section 10.4.6.

The choice of these threshold values is not arbitrary: Section 10.4.6 reports a dedicated sensitivity analysis of the Offloading and Fallback Thresholds, conducted by sweeping their combinations against recorded network traces and measuring the resulting CPU active time and navigation distance. That analysis is, however, primarily focused on the Energy-Aware MUX configuration under moderate packet loss (below 20%); a broader sensitivity analysis covering the full range of network conditions evaluated in this chapter (up to 95% packet loss) and explicitly relating threshold choice to the safety-criticality trade-offs introduced here is left as a natural extension of the present evaluation.

The assumption that the time interval between messages is constant is addressed by introducing the MMC component. It observes network latency, packet loss, and computation delay, providing the system with updated estimates of expected round-trip time. These measurements are used by all the components to adjust validity thresholds (the Δt between two messages) and dynamically adapt robot speed to current network performance.

This scenario, a single mobile robot performing vision-based navigation while dynamically switching between local and remote (edge-offloaded) processing under a degrading wireless link, was selected as representative of a broad and practically important class of mobile robotics deployments: any robot operating beyond reliable, short-range connectivity (e.g., outdoor inspection robots, warehouse robots at the edge of WiFi coverage) faces the same trade-off between onboard processing limitations and intermittent network access. The empirical network traces used to drive the simulation (Section 10.4.1) were collected from a physical robot under real WiFi conditions, grounding the scenario in measured rather than purely synthetic network behaviour.

10.3 Problem formulation

We consider a mobile robot navigating a bounded two-dimensional environment containing static or moving obstacles. The system operates in discrete time steps $k \in \mathbb{N}$. At each step, the robot receives a sensor observation $s_k \in \mathscr{S}$ and must apply a control command $u_k \in \mathscr{U}$ (comprising linear velocity v_k and angular velocity ω_k).

The robot is equipped with a hybrid architecture capable of obtaining control commands from a set of execution streams $\mathscr{I} = \{\text{loc}, \text{rem}\}$, corresponding to the Local Navigator and Remote Navigator. Although for simplicity we formulate the stream switching as a binary problem, it is worth mentioning that the model can be easily extended to include more than two streams. Finally, let $\sigma_k \in \mathscr{I}$ denote the *stream selection variable* determined by the MUX at step k.

10.3.1 Delay and latency dynamics

The age of the control information reaching the actuators is critical for safety. Each stream $i \in \mathscr{I}$ produces a command $u_k^{(i)}$ associated with a total delay $\tau_k^{(i)}$:

$$\tau_k^{(i)} = t_{sens} + t_{proc}^{(i)} + t_{net}^{(i)}, \tag{10.1}$$

where:

- t_{sens} is the sensor acquisition latency.
- $t_{proc}^{(i)}$ is the processing time. Typically, $t_{proc}^{(\text{rem})} \ll t_{proc}^{(\text{loc})}$ due to the superior computational capacity of the edge server.
- $t_{net}^{(i)}$ represents the bidirectional network delay. For the local stream, $t_{net}^{(\text{loc})} \approx 0$, while $t_{net}^{(\text{rem})}$ is a stochastic variable subject to jitter and packet loss.

The system relies on a MMC to provide an online estimate $\hat{\tau}_k$ of this delay based on recent network conditions.

10.3.2 Safety constraints

Let $d_{obs,k}$ denote the distance to the nearest obstacle along the robot's path at each time instant k. To prevent collisions, the robot must be able to come to a complete stop within this distance, accounting for the reaction time induced by the system delay. The required stopping distance is defined as:

$$d_{stop}(v_k, \hat{\tau}_k) = \underbrace{v_k \cdot \hat{\tau}_k}_{\text{Reaction Dist.}} + \underbrace{\frac{v_k^2}{2a_{brake}}}_{\text{Braking Dist.}} \tag{10.2}$$

where a_{brake} is the maximum mechanical deceleration of the robot. The safety constraint requires $d_{stop} \leq d_{obs,k}$. Solving this inequality for v_k yields the maximum admissible velocity v_{max} as a function of the delay:

$$v_{max}(\hat{\tau}_k) = -a_{brake}\hat{\tau}_k + \sqrt{(a_{brake}\hat{\tau}_k)^2 + 2a_{brake}d_{obs,k}} \tag{10.3}$$

Equation (10.3) formally demonstrates that as network latency ($\hat{\tau}_k$) increases, the maximum safe velocity (v_{max}) decreases, limiting performance.

10.3.3 Energy consumption model

To model the efficiency gains provided by the LO, we define the power consumption E_k at step k. The local navigator node can be toggled between an *active* and *inactive* state, denoted by the binary variable $\phi_k \in \{0,1\}$. The system energy cost is:

$$E_k(\phi_k) = E_{base} + E_{net} + \phi_k \cdot E_{cpu}^{(loc)} \tag{10.4}$$

where $E_{cpu}^{(loc)}$ is the power cost of running the local navigation stack, and E_{net} is the transmission cost associated with streaming sensor data to the edge server. Because sensor data is always transmitted to the remote Navigator regardless of whether local computation is active, E_{net} appears as a constant term. The only variable contribution is $E_{cpu}^{(loc)}$, which is incurred only when the local Navigator is in the active state ($\phi_k = 1$). When the LO deactivates the local Navigator ($\phi_k = 0$), the onboard energy cost is reduced to $E_{base} + E_{net}$, removing the computational overhead of the local navigation stack. The Remote Navigator's power consumption is externalized to the edge server and is not included in the robot's onboard energy budget.

10.3.4 Optimization objective

The objective of the proposed architecture is to maximize the distance travelled over a finite horizon N, accounting for the robot trajectory described by each position p_k, while minimizing onboard energy consumption and strictly satisfying safety

constraints. This is formulated as finding the optimal sequence of stream selections $\{\sigma_k\}$ and node states $\{\phi_k\}$:

$$\max_{\{\sigma_k,\phi_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} (||p_{k+1} - p_k|| - \lambda E_k(\phi_k)) \tag{10.5}$$

s.t.

$$p_{k+1} = p_k + f(p_k, u_k) \tag{10.6}$$

$$u_k \in \mathscr{U}_{valid} \tag{10.7}$$

$$||u_k|| \leq v_{max}(\hat{\tau}_k^{(\sigma_k)}) \tag{10.8}$$

Here, λ is a weighting factor balancing performance and energy efficiency. Equation (10.6) determines the position at $k+1$ as a combination of the current position and the trajectory f computed based on the control command u_k issued to the wheels. Equation (10.7) implies that a stream σ_k can only be selected if its message is fresh (within validity timestamps). The MUX solves this selection problem in real-time to maintain high velocity (via remote offloading) while ensuring safety (via local fallback), and the LO optimises ϕ_k to minimize E_k. Finally, Equation (10.8) ensures that the velocity command issued to the robot does not exceed the maximum velocity in accordance with Equation (10.3). Here, $\hat{\tau}_k^{(\sigma_k)}$ denotes the delay estimate associated with the selected stream σ_k, ensuring that the velocity bound is always computed using the latency of the stream currently driving the robot.

10.3.5 Algorithm overview

The MUX selects the active stream based on the most recent valid timestamp, while respecting stream reliability and priority. Its operation is summarized in Algorithm 1.

At initialization, the MUX is configured with a predefined set of candidate streams, in our case remote, local, and emergency. At this stage, no stream is marked as reliable, no active (best) stream is selected, and no last valid message is stored for any stream.

Whenever a new message arrives from a stream, the MUX first checks its validity by comparing the message timestamp with the timestamp of the last valid message

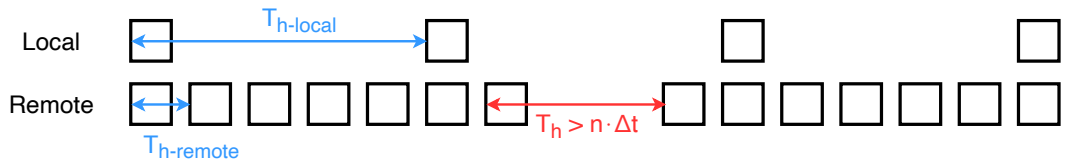


Fig. 10.2 Decision threshold used in stream selection to switch between local and remote streams.

received from the same stream. If the timestamp is newer, the message is stored as the last valid message for that stream and the stream reliability state is updated. The set of reliable streams is then recomputed, and the stream with the highest priority among the reliable ones is selected as the current active stream. The last valid message associated with this selected stream is forwarded, and its watchdog timer is restarted. Conversely, if the incoming message carries an outdated timestamp, it is discarded and only the stream reliability state is updated.

Stream reliability is evaluated based on message inter-arrival times. Each stream is associated with a nominal timeout Δt, representing the expected interval between consecutive messages. If no message is received within n multiples of this timeout ($T_h > n \cdot \Delta t$), the message is considered lost (as illustrated in Figure 10.2). When a stream incurs m consecutive lost messages, it is marked as unreliable and excluded from the set of selectable streams.

In addition, each stream is monitored by a watchdog timer. The timer is reset whenever a new valid message is received from that stream. If the watchdog timer expires, the corresponding stream is immediately removed from the set of reliable streams. The MUX then selects the highest-priority stream among the remaining reliable ones, forwards its last valid message, and restarts the associated watchdog timer.

10.4 Experimental evaluation

This section evaluates our MUX-based approach using both simulation-driven and real-world implementation-driven experiments. In the following, we first focus on the simulation, moving then to the actual implementation.

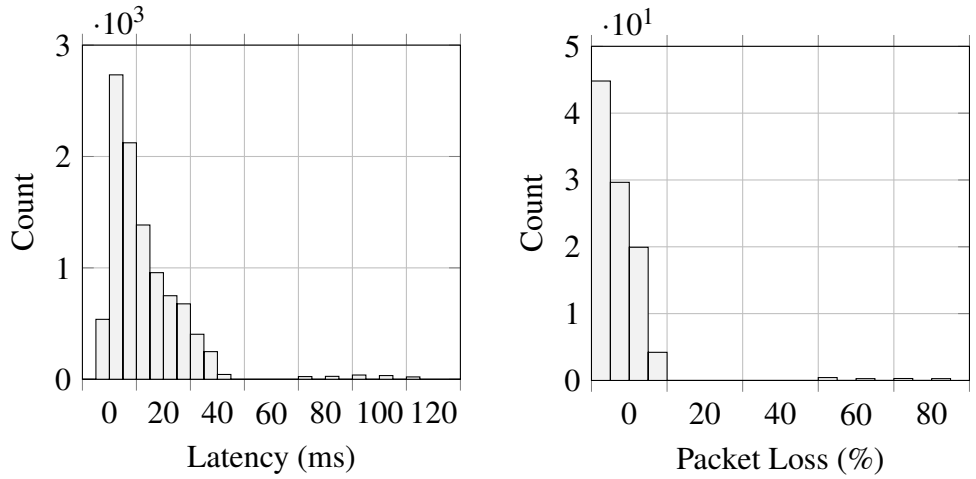
This section addresses the following research questions:

Algorithm 1 MUX algorithm

```
function OnInit()
    streams ← [remote, local, emergency]
    reliableStreams ← []
    bestStream ← None
    lastMsg ← [None, None, None]
end function

function OnNewData(message msg, stream stream)
    if msg.ts_src > lastMsg[stream].ts_src then
        lastMsg[stream] ← msg
        UpdateReliability(stream)
        reliableStreams ← FilterReliableStreams(streams)
        bestStream ← SelectHighestPriority(reliableStreams)
        SendLastMessage(bestStream)
        RestartTimer(bestStream)
    else
        Drop(currentMsg)
        UpdateReliability(stream)
    end if
end function

function OnWatchdogTimer(stream stream)
    RemoveStream(stream, reliableStreams)
    bestStream ← SelectHighestPriority(reliableStreams)
    SendLastMessage(bestStream)
    RestartTimer(bestStream)
end function
```



(a) Latency distribution.

(b) Packet loss distribution.

Fig. 10.3 Network characteristics of the Wi-Fi IEEE 802.11 link in terms of latency and packet loss used to model the communication medium in the simulations.

RQ1. How do data acquisition frequency and processing time affect navigation performance under local vs. remote execution (Section 10.4.2 and Section 10.4.3)?

RQ2. How does packet loss affect the performance of each navigation strategy, and how does the proposed Energy-Aware Multiplexed approach compare to simpler alternatives (Section 10.4.4 and Section 10.4.5)?

RQ3. How sensitive are the Offloading/Fallback thresholds to the choice of their parameters (Section 10.4.6)?

10.4.1 Simulation setup

The system is implemented using the ROS2 and deployed utilising containerization technology to ensure consistency and scalability. Each functional component, including all ROS2 nodes, is encapsulated within a dedicated *Docker image*. The simulation environment is orchestrated using *Docker Compose*, which manages concurrent deployment and allows for flexible configuration via environment variables based on specific simulation parameters. Finally, the entire stack is tested in the Gazebo simulation framework.

The network environment is emulated using empirical data collected from a physical robot. During the robot traversal, network metrics were recorded via a WiFi

IEEE 802.11 link using the `iperf3` and `ping` utilities. The resulting dataset captures real-world latency and packet loss trajectories (see Figure 10.3). Analysis of 10,000 samples revealed the following characteristics:

- **Latency:** The distribution exhibited a median of 19ms and an average of 23ms (Figure 10.3a). While 99% of observations fell between 9ms and 90ms, peak latency reached 120ms.
- **Packet Loss:** Considering 100s windows, the packet loss is measured for each window as the ratio of delivered packets. Samples were primarily concentrated at low rates (0-15%), with 99% of values remaining below 80% (Figure 10.3b). However, distinct spikes were observed at 60%, with sparse occurrences at 70% and 80%.

Unless otherwise specified, the following tests integrate these empirical distributions to accurately mimic real-world degradation, ensuring robust testing of the navigation system under adverse conditions.

To rigorously evaluate the system and highlight the advantages of the developed components, a structured set of simulation scenarios was defined and executed. The five distinct simulation scenarios are detailed below:

- **Local-Only Navigation:** The robot exclusively relies on the local Navigator. The local Navigator has a computational delay of 500ms.
- **Remote-Only Navigation:** The robot utilizes only the remote Navigator. The link is explicitly subject to induced network impairments (latency and packet loss), and the remote Navigator has a computational delay of 10ms. A minimal timeout-based stop command is issued if no message is received within a fixed window, preventing indefinite forward motion, but no local fallback stream is available.
- **Multiplexed Navigation:** Both the local and remote Navigators operate concurrently. The dedicated MUX component processes both streams and selects the output deemed most reliable. The robot is immediately stopped if neither navigation stream is assessed as reliable by relying on a fallback stream.
- **Energy-Aware Multiplexed Navigation:** Both Navigators run in parallel at start-up. The MMC allows for simulation parameters to be adapted based

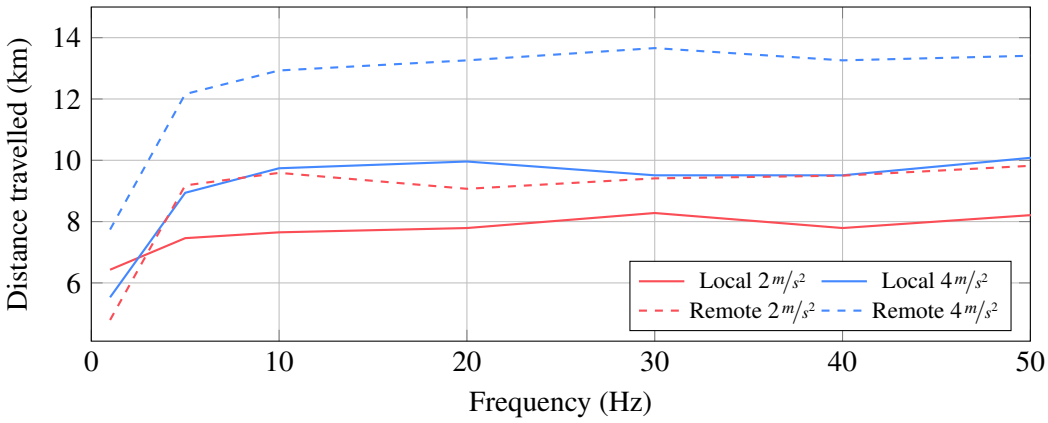


Fig. 10.4 Impact of sensor acquisition frequency on the traveled distance in case of local and remote processing under ideal network conditions.

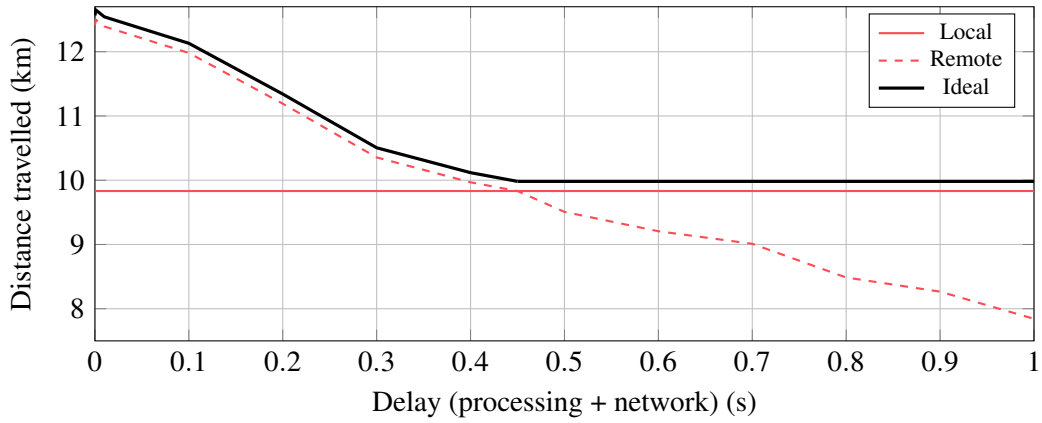


Fig. 10.5 Impact of delays (network + processing) on distance traveled, both in the case of local and remote processing.

on predictions selected via a Kalman filter, dynamically adjusting validity-window thresholds for the MUX, and LO may disable the local Navigator to reduce power consumption.

- **FogROS-based Navigation:** The robot follows the offloading strategy typical of state-of-the-art frameworks, using only the remote navigator. Faults or disconnections are managed solely by the underlying offloading mechanism, providing a critical baseline comparison.

The computational delay values are not arbitrary: as detailed in Section 10.4.2, they were obtained by measuring the same vision-based perception workload running locally on the robot's onboard processor versus on a GPU-equipped edge server. The

gap therefore reflects a measured hardware capability difference for this specific perception workload, rather than an assumed or convenient parameter choice.

10.4.2 Impact of data acquisition frequency

The first parameter analysed is the influence of the data acquisition frequency from input sensors on the distance travelled by the robot. Intuitively, a higher acquisition frequency facilitates a faster response to unexpected events (e.g., an object unexpectedly crossing the robot's path). Consequently, this enhanced situational awareness of the surrounding environment enables the robot to maximize its travelled distance.

Assuming an ideal communication channel (e.g., no packet loss), we quantify the performance gap for the same inference workload executed locally on the autonomous robot (equipped with an i7 processor) versus remotely on an edge server (with a dedicated GPU), using the robot's video stream as input. In these tests, on-board processing achieved an average of 2 Frames Per Second (FPS) (i.e., 500ms processing time), whereas the remote execution attained 40FPS (i.e., 25ms processing time). Figure 10.4 illustrates the relationship between the distance travelled by the robot and two maximum acceleration values, for both local and remote only processing modes. While an increase in the robot's acceleration predictably extends the travelled distance, the performance gap between local and remote processing becomes increasingly pronounced at higher acceleration settings. This widening gap is attributed to the shorter response time of the edge server, which facilitates highly precise control decisions, a factor that becomes critically important at elevated operational speeds. Furthermore, our experimental results indicate that, within this specific configuration, no significant improvement in the total distance traversed is achieved by increasing the video feed rate beyond 10Hz. This ceiling is reached because the robot's control logic does not effectively utilize the additional information provided by higher sampling frequencies. As a result, there is a clear advantage when offloading the computation to remote edge servers; however, this assumption holds only for ideal network channels. In the following, we add perturbations in the communication medium to assess the trade-offs of the offloading process.

10.4.3 Impact of processing time

This subsection analyses how processing time affects the robot's performance, and, more specifically, how the network latency can affect the computation. We compared local execution (no network delay, $t_{proc} = 500ms$) with remote execution, where processing time varies between 40ms (accounting for no network delay, used only as a baseline for comparison) and 1000ms ($t_{proc} = 40ms, t_{net} = 960ms$). The simulations analyse the cases with a sensor frequency of 10Hz, as it represents the ceiling point for our setup, as highlighted in the previous test.

Figure 10.5 confirms a direct correlation between the distance travelled and the network latency incurred when utilising remote processing resources. While local processing maintains a stable travel distance due to its inherent predictability, remote computation is highly sensitive to network delay, which compromises the system's real-time capability. Yet, provided the network remains relatively stable, the superior computational capacity of the edge server effectively mitigates the impact of the transmission delay. Conversely, when network latency increases, the remote performance suffers a sharp reduction compared to the local alternative. Therefore, the figure underscores the requirement for an adaptive switching mechanism (designated as *ideal*), capable of continuously assessing network quality and optimally transitioning between local and remote processing modes.

10.4.4 Influence of packet loss

This test compares the performance of using the Local-Only, the Remote-Only, and the Multiplexed Navigation approach, evaluating the distance travelled by the robot as a result of the source of the navigation commands. We performed 10 simulations for each box plot of Figure 10.6, starting from 0% packet loss, with 5% step increase up to 100%. The network latency was fixed at 15ms.

As shown in Figure 10.6, relying on the Remote-Only Navigator stream under high packet loss conditions leads to significant performance degradation. The superior processing capacity of the remote server allows the robot to travel at a much higher speed than in the Local-Only configuration. However, the travelled distance begins to drop sharply around 55% packet loss. At this point, the robot has already lost 20.8% of its performance relative to the 0% packet loss baseline.

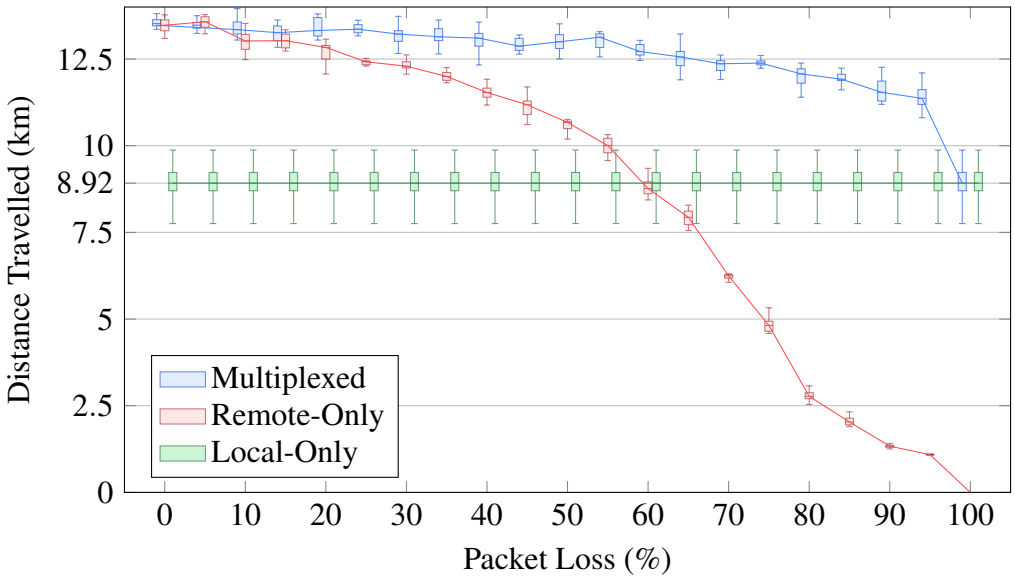


Fig. 10.6 Distance travelled under Local-Only, Remote-Only, and Multiplexed navigation at varying packet loss rates.

Conceptually, an increase in packet loss degrades the data flow coming from the remote Navigator instance, ultimately impairing the robot's functionality. Conversely, the Local-Only configuration is not affected by network perturbations; however, because of the limited local computing power available, this configuration proves to be effective only under lossy network conditions (i.e., $>60\%$), drastically surpassing the Remote-Only configuration. In contrast, keeping both the local and remote navigator active by means of the MUX allows to alternate the streams depending on network conditions. The implementation remains stable: as packet loss increases, the MUX progressively prioritizes messages from the local Navigator over the remote one, thereby maintaining consistent performance. Additionally, even if a packet drop affects the availability of the remote stream, the few packets that are still received allow the robot to travel at a consistently higher speed compared to the Local-Only configuration. With 0% packet loss, the redundant approach allows the robot to travel a median distance of 13.521km, while at 95% packet loss, the robot still travels 11.365km, corresponding to a reduction of 15.9%. These results underscore that by carefully forwarding the few messages received from the remote stream under bad network condition, the MUX approach can even exceed the *ideal* configuration presented in Figure 10.5.

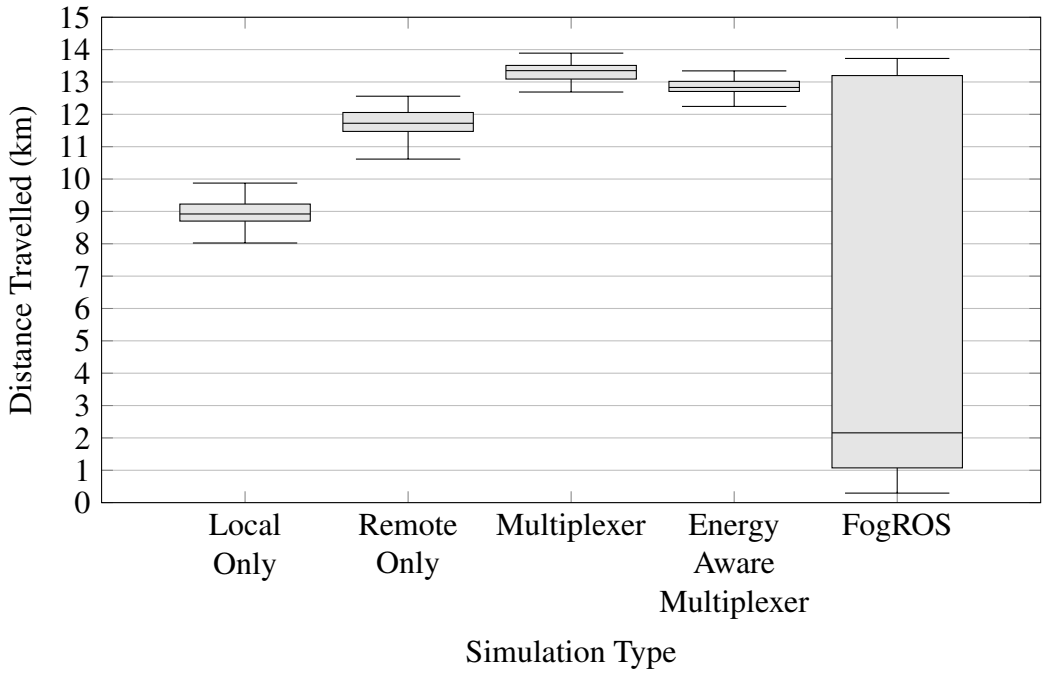


Fig. 10.7 Distance travelled per simulation type across 100 independent runs.

10.4.5 Replay-based navigator scenarios comparison

To evaluate the robot's behaviour, we conducted a series of comparative tests across the network scenario detailed in the testbed setup. Figure 10.7 illustrates the performance distribution for each configuration via box plots, each aggregating the results of 100 independent simulation runs.

The evaluation begins with two primary baselines: **Local-Only Navigation** and **Remote-Only Navigation**. In the Local-Only configuration, the robot relies exclusively on the local Navigator under the assumption of negligible latency ($\approx$0ms) and zero packet loss. Since this setup bypasses network emulation, the robot travelled a median distance of 8929m. Conversely, the Remote-Only baseline utilizes the remote Navigator exclusively. Despite the presence of packet loss, which triggers safety fallback mechanisms such as stop commands to prevent collisions, this configuration achieved a significant performance gain, outperforming the local baseline by approximately 27%.

Building upon these baselines, we introduced two multiplexing strategies designed to enhance reliability. The standard **MUX** setup concurrently runs both local

and remote Navigators, where the MUX forwards local commands only when the remote stream becomes unreliable due to network instability. This collaborative approach resulted in a median distance of 13.364km, representing a 14% improvement over the Remote-Only baseline. The **Energy-Aware MUX** was tested to assess the trade-off between navigation performance and computational efficiency. In this configuration, the local Navigator remains deactivated while the remote stream is stable, reactivating only upon detected unreliability to resume navigation. While this power-saving strategy resulted in a slight performance decrease compared to the standard MUX, travelling a median distance of 13.128km (approximately 96% of the MUX performance), it still maintained a 9.5% improvement over the Remote-Only baseline, confirming its viability for resource-constrained operations.

Finally, the **FogROS** test exhibits high variability because of a lack of a fallback mechanism to stop the robot when the connection to the remote system is lost. In this scenario, the robot can travel a distance comparable to that of our solutions; however, the absence of such a mechanism means that safe operation is not guaranteed. In some cases, the robot crashes after travelling only 200m, and this behaviour accounts for the variability observed in the graph.

10.4.6 Energy aware MUX threshold analysis

The LO is integrated to enhance operational efficiency by dynamically modulating the operational state of the local Navigator. By deactivating the local copy during periods of stable remote connectivity, the system significantly reduces computational overhead and conserves energy. To evaluate the efficacy of the LO and determine optimal configuration parameters, we conducted experiments across various scenarios using the realistic network conditions recorded previously. The primary objective was to identify the most effective threshold values for transitioning between local and remote states.

The impact of these parameters on system behaviour is illustrated in Figure 10.8, which presents the local component's CPU active duration for various combinations of Offloading and Fallback Thresholds. Experimental data indicate that under the observed conditions, where packet loss typically remains below 20%, the total distance travelled does not significantly fluctuate in response to changes in the local component's active time Detailed analysis of the network traces reveals that at low

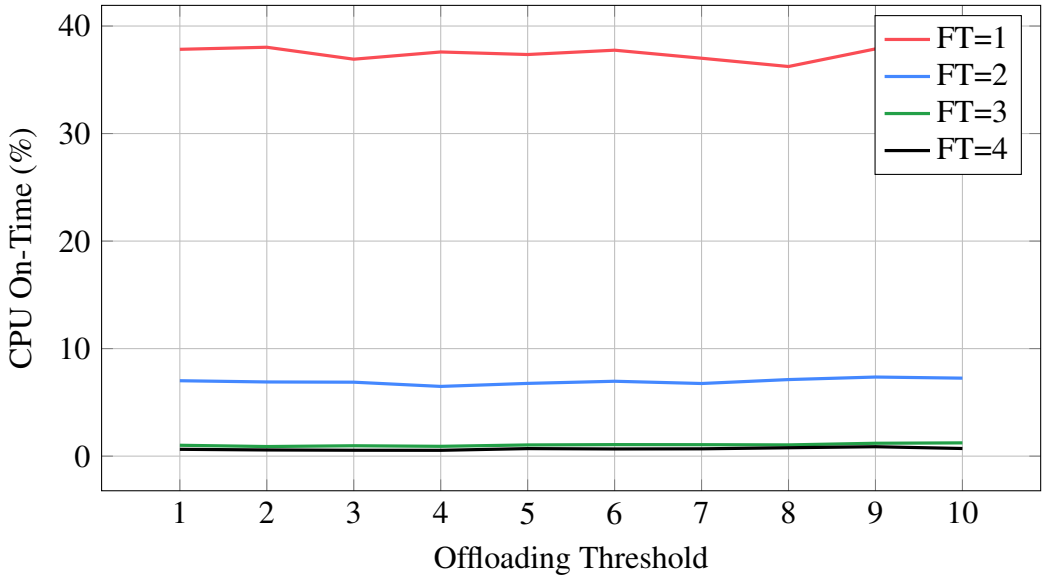


Fig. 10.8 CPU On-Time percentage for different Offloading Thresholds, grouped by Fallback Threshold (FT).

packet loss rates, instances of multiple consecutive packet drops are rare; typically, the reception of a single packet is followed by a stable stream of subsequent messages. Consequently, a Fallback Threshold of 3 proved sufficient for the system to detect remote stream outages reliably. While the current low-loss environment precluded definitive conclusions regarding the offloading threshold, the results demonstrate that judicious selection of fallback parameters allows for the prioritized use of the remote navigator. This ensures that the local navigator is activated only during periods of pronounced network instability, effectively balancing navigation performance with energy conservation, by reducing the on-board CPU usage of nearly 40% with a significant increase in energy efficiency.

10.4.7 Real robot test

To validate the proposed solution, we evaluated the MUX in a real-world scenario using a physical robot. The RB-SUMMIT is a four-wheeled, skid-steered robot capable of rotating in place. It is equipped with multiple sensors, including a Light Detection and Ranging (LiDAR) and an Intel RealSense D435 camera. The robot can move forward and backward at speeds of up to 3m/s, and each wheel is driven by an independent 500W motor. The onboard computing unit features 8 CPU cores

and 32GB of RAM; however, no GPU or other dedicated acceleration hardware is available on the robot. The edge server is deployed as a virtual machine equipped with 8 CPU cores, 32GB of RAM, and an NVIDIA A40 GPU. The robot connects to the edge server via Wi-Fi through an access point, which is in turn connected to the edge server via a wired link. Finally, the robot is equipped with a battery whose metrics are exposed through a BMS, enabling the measurement of power consumption under different operational conditions and workloads.

The experimental scenario is defined as follows: the robot is tasked with moving forward until a stop sign is detected. During operation, the robot advances in a straight line while continuously monitoring the environment for the presence of a stop sign. Once detected, the robot estimates the distance to the sign and aims to stop at a predefined target distance. The onboard camera captures RGB images at 30FPS along with the corresponding depth maps, allowing the system to estimate distances to objects within the scene. This depth information is combined with object detection results obtained through a machine learning model to determine the distance between the robot and the detected stop sign. The object detection component is implemented using the You Only Look Once (YOLO) framework [136]. Based on this estimate, the robot dynamically adjusts its speed to approach and stop at the target distance without collision safely. A test is considered successful if the robot stops at the correct target distance. Conversely, the test is classified as a failure if the robot stops beyond the target distance (i.e., the stop sign is detected too late for the robot to decelerate in time) or if the robot fails to stop entirely due to the stop sign not being detected.

The robot was evaluated under two configurations: Local-Only and Multiplexed Navigation. Due to the absence of a dedicated GPU on the robot, the onboard system uses the YOLOv8n model, whereas the edge server runs the YOLOv12x model. The onboard model achieves a processing rate of approximately 7FPS and is therefore unable to process all frames captured by the camera (i.e., 30FPS in our setting), handling only about 23% of the total frames. In contrast, the edge server can process up to approximately 70FPS, which exceeds the 30FPS camera input rate. As a result, the edge server is capable of real-time processing, assuming ideal network conditions and disregarding network reliability constraints.

To evaluate the MUX on the real robot, we conducted repeated experiments over a fixed duration of 5 minutes, with the robot performing identical navigation behaviour

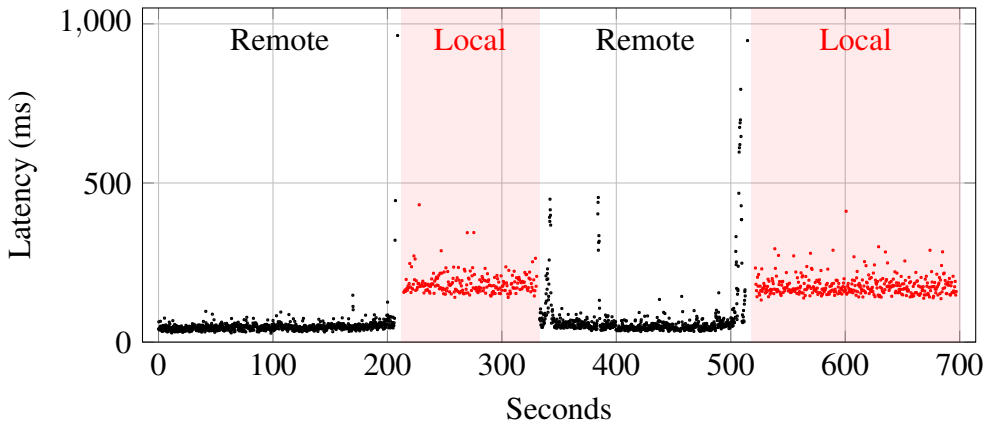


Fig. 10.9 End-to-end inference latency. Remote inference includes RTT in the total response time. The overlay shows the stream selected by the MUX.

throughout. The robot travels straight until it detects a stop sign; the two stop signs are placed 6m apart. Upon reaching one, the robot stops, rotates 180° in place, and moves forward until it reaches the other. It then rotates again and repeats this behaviour until the time expires. When running the workload entirely onboard, the robot consumed on average 1.7% of battery every 5 minutes, equivalent to a depletion rate of 20.4% per hour, giving an estimated battery lifetime of approximately 4.9 hours. When running the workload with the MUX, battery consumption dropped to 1.6% per 5 minutes (19.2% per hour), extending the estimated lifetime to approximately 5.2 hours, i.e., 6% increase in battery life.

Figure 10.9 shows the latency of the Local and Remote Navigators. When the Remote Navigator is deemed reliable and active, the Local Navigator produces no data, indicating that the LO has disabled it to reduce power consumption. Conversely, when Remote Navigator latency exceeds the reliability threshold, the LO re-enables the Local Navigator and the MUX switches to forwarding its messages. No data are shown for the Remote Navigator in this phase, as the robot lost connectivity to the edge server entirely. Comparing the Multiplexed and Local-Only configurations, the robot achieves 4.7% greater distance travelled alongside a 5.9% reduction in battery usage, which means 6.1% more estimated lifetime. These results demonstrate that even partial remote offloading yields net benefits: disabling the Local Navigator when the Remote Navigator is reliable increases travelling distance and reduces power consumption. The tests use Transmission Control Protocol (TCP) rather than UDP. This choice is driven by the nature of the transmitted data: image frames

are large and must be transferred intact from the robot to the edge server. A single frame spans multiple layer-4 packets, so the loss of even one packet corrupts the entire frame, rendering it unusable as input to the YOLO algorithm. UDP offers no mechanism to recover from such losses, making it unsuitable here; TCP is therefore used to ensure reliable image transmission.

10.5 Concluding remarks

This chapter addressed the critical challenge of maintaining reliable job offloading for mobile robots operating over unstable network infrastructures. While edge computing offers significant computational advantages, our study confirms that relying solely on remote execution introduces catastrophic safety risks during packet loss or latency spikes.

To mitigate these risks, we introduced a hybrid architecture centred on two novel components: (*i*) The MUX, to perform real-time, message-level arbitration between local and remote streams. By prioritizing high-frequency remote data when stable and seamlessly falling back to local or default safety commands during degradation, the MUX ensures continuous and safe navigation. (*ii*) The LO: Enhances operational efficiency by dynamically deactivating the local navigator when the remote stream is reliable. This reduces redundant computation without sacrificing the robot's ability to react to sudden network failures.

In extreme conditions (e.g., 55% packet loss), where remote-only systems typically fail, the MUX maintained a 45% performance lead over local-only execution by effectively utilizing partial remote data. In real-world network traces, the LO achieved a reduction in local CPU usage during stable periods, resulting in a 6% increase in overall battery life on a physical RB-SUMMIT robot. Our architecture consistently outperformed traditional offloading frameworks like FogROS, which exhibited dangerous bimodal behaviour and frequent crashes due to a lack of local fallback mechanisms.

Chapter 11

Concluding Remarks

This part of the thesis investigated the application of Cloud Continuum principles to robotic systems, addressing the challenges that arise when computation is distributed across local, edge, and cloud resources. The work was motivated by the observation that the value of next-generation network infrastructures, including those envisioned for 6G, can only be assessed through concrete use cases that exercise their capabilities in realistic settings. Robotics, with its stringent requirements in terms of latency, reliability, and energy efficiency, provides a compelling application domain for validating such architectures. Across three complementary contributions, this part addressed the communication, orchestration, and resilience dimensions of distributed robotic systems.

Chapter 8 tackled the communication layer by introducing SM2, a lightweight middleware that decouples local inter-process communication from network transport. The key insight behind SM2 is that local communication performance should not be affected by the presence or condition of remote subscribers, a property that DDS-based ROS2 configurations fail to guarantee under adverse network conditions. Experimental results showed that SM2 achieves comparable throughput to state-of-the-art alternatives such as ROS2 with Zenoh RMW, while reducing CPU usage by $3\times$ and memory consumption by $4\times$. Furthermore, SM2 sustains higher message frequencies and lower latencies as the number of communicating nodes increases, making it a practical solution for resource-constrained robotic platforms. The SNG extends this reliability to inter-machine communication through secure, configurable connectivity based on the QUIC protocol.

Chapter 9 addressed the problem of migrating stateless ROS2 computation between Kubernetes clusters, evaluating four switching architectures that differ in their use of redeployment, NetPols, and ROS2 Lifecycle Nodes. The results demonstrated that the Lifecycle Node approach outperforms all other options, achieving switching times at least 85% faster than alternatives based on redeployment or NetPols. This chapter also highlighted the importance of controlling duplicate messages during the transition period and showed that mechanisms to halt communication from the old instance can effectively preserve service continuity.

Chapter 10 extended the offloading paradigm to unreliable network environments, where packet loss and latency variability can severely degrade the performance of remotely controlled robots. The proposed MUX component enables seamless arbitration between local and remote navigation streams based on message freshness and stream reliability, ensuring continuous and safe operation even under adverse conditions. At 55% packet loss, the MUX-based approach achieved a 31% improvement in travelled distance over remote-only control and a 47% improvement over local-only control. The LO complements this mechanism by dynamically deactivating the local navigator when remote execution is stable, resulting in a reduction of up to 99% in local CPU usage with only a marginal decrease in navigation performance.

Taken together, these contributions demonstrate that the Cloud Continuum paradigm can be effectively applied to robotics when each layer of the system, from communication middleware to orchestration and adaptive stream management, is designed with awareness of the constraints imposed by distributed, heterogeneous, and potentially unreliable environments. The results confirm that hybrid local-remote execution strategies represent a viable path toward edge-enabled robotics, combining the computational benefits of offloading with the robustness required for safe autonomous operation. In doing so, this work provides concrete evidence that the architectural principles underlying next-generation infrastructures, such as those envisioned for 6G, yield tangible benefits when paired with demanding real-world applications.

Several directions remain open for future investigation. On the communication side, SM2 could benefit from expanded platform support and improved serialization compatibility across heterogeneous architectures, while the SNG could be enhanced with greater parallelism and per-connection interface handling to better isolate faulty peers. Regarding computation migration, a deeper understanding of the time required

by Kubernetes to enforce and remove NetPols could make policy-based switching a more competitive alternative to Lifecycle Nodes, broadening the applicability of the switching paradigm beyond ROS2. On the adaptive offloading front, predictive switching strategies that anticipate network degradation before failures occur represent a natural evolution, potentially leveraging machine learning models trained on historical network traces. Finally, extending the proposed approaches to more complex robotic workloads involving multiple concurrent tasks, stateful computation, and multi-robot coordination would further strengthen their applicability in real-world deployment scenarios.

References

[1] Peter Mell and Tim Grance. The NIST definition of cloud computing, 2011.

[2] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015.

[3] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, sep 2015.

[4] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[6] Dejan Milojicic. The edge-to-cloud continuum. *Computer*, 53(11):16–25, 2020.

[7] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Trans. Internet Technol.*, 19(2), apr 2019.

[8] Mulugeta Ayalew Tamiru, Guillaume Pierre, Johan Tordsson, and Erik Elmroth. Instability in geo-distributed kubernetes federation: Causes and mitigation. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8, 2020.

[9] Lars Larsson, William Tärneberg, Cristian Klein, Erik Elmroth, and Maria Kihl. Impact of etcd deployment on kubernetes, istio, and application performance. *Software: Practice and Experience*, 50(10):1986–2007, 2020.

[10] Lirim Osmani, Tero Kauppinen, Miika Komu, and Sasu Tarkoma. Multi-cloud connectivity for kubernetes in 5g networks. *IEEE Communications Magazine*, 59(10):42–47, 2021.

[11] Abdullah Yousafzai, Abdullah Gani, Rafidah Md. Noor, Mehdi Sookhak, Hamid Talebian, Muhammad Shiraz, and Khurram Khan. Cloud resource allocation schemes: review, taxonomy, and opportunities. *Knowledge and Information Systems*, 50, feb 2017.

[12] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, Marco A. S. Netto, Adel Nadjaran Toosi, Maria Alejandra Rodriguez, Ignacio M. Llorente, Sabrina De Capitani Di Vimercati, Pierangela Samarati, Dejan Milojicic, Carlos Varela, Rami Bahsoon, Marcos Dias De Assuncao, Omer Rana, Wanlei Zhou, Hai Jin, Wolfgang Gentzsch, Albert Y. Zomaya, and Haiying Shen. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Comput. Surv.*, 51(5), nov 2018.

[13] Pieter-Jan Maenhaut, Bruno Volckaert, Veerle Ongenae, and Filip De Turck. Resource management in a containerized cloud: Status and challenges. *Journal of Network and Systems Management*, 28(2):197–246, 2020.

[14] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. Computing without borders: The way towards liquid computing. *IEEE Transactions on Cloud Computing*, 11(3):2820–2838, 2023.

[15] Lars Nagel, Juan Jose Hierro, Eugenio Perea, Douwe Lycklama, Christoph Mertens, et al. Design principles for data spaces: Position paper. Technical report, E. ON Energy Research Center, 2021.

[16] Jacopo Marino, Leonardo Camiciotti, Francesco Cheinasso, Alessandro Olivero, and Fulvio Risso. Enabling compute and data sovereignty with infrastructure-level data spaces. In *Proc. 3rd Eclipse Secur. AI Archit. Model. Conf. Cloud Edge Continuum*, page 77–85. Association for Computing Machinery, 2023.

[17] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E. Haque, et al. The architectural implications of autonomous driving: Constraints and acceleration. In *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, page 751–766. Association for Computing Machinery, 2018.

[18] Muhammad Zeeshan Khan, Saad Harous, Saleet Ul Hassan, Muhammad Usman Ghani Khan, Razi Iqbal, and Shahid Mumtaz. Deep unified model for face recognition based on convolution neural network and edge computing. *IEEE Access*, 7:72622–72633, 2019.

[19] Loris Belcastro, Fabrizio Marozzo, Alessio Orsino, Domenico Talia, and Paolo Trunfio. Edge-cloud continuum solutions for urban mobility prediction and planning. *IEEE Access*, 11:38864–38874, 2023.

[20] Jacopo Marino and Fulvio Risso. Is the computing continuum already here? In *I-CiTies 2023*. arXiv, sep 2023.

[21] Stefano Galantino, Elisa Albanese, Nasir Asadov, Stefano Braghin, Francesco Cappa, Andrea Colli-Vignarelli, Amjad Yousef Majid, Eduard Marin, Jacopo Marino, Lorenzo Moro, Liubov Nedoshivina, Fulvio Risso, Domenico Siracusa, Antonio Skarmeta, and Luca Zuanazzi. Building the cloud continuum with rear. In *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, pages 67–72, 2024.

[22] Stefano Galantino, Jacopo Marino, Fulvio Risso, Stefano Braghin, Liubov Nedoshivina, Antonio Fernando Skármeta Gómez, and Domenico Siracusa. Edge-to-cloud continuum made real. *Computer*, 59(3):51–59, 2026.

[23] The FLUIDOS consortium. Fluidos public deliverable - d2.1 scenarios, requirements and reference architecture. https://fluidos.eu/public-deliverables/, 2023.

[24] ETSI. Mobile-edge computing – introductory technical white paper, sep 2014.

[25] ETSI. Harmonizing standards for edge computing - a synergized architecture leveraging etsi isg mec and 3gpp specifications, jul 2020.

[26] TIM. Edge cloud computing, jan 2022.

[27] Andrea Calvi, Giuseppe Catalano, and Ivano Guardini. L'evoluzione verso il telco cloud l'edge computing, mar 2020.

[28] Telefónica. Telefónica open access and edge computing - white paper, feb 2019.

[29] FiberCop. FiberCop: con Microsoft, Dell Technologies e 6WIND prime soluzioni edge cloud per i servizi telco avanzati. https://www.fibercop.com/2026/02/25/fibercop-con-microsoft-dell-technologies-e-6wind-prime-soluzioni-edge-cloud-per-i-servizi-telco-avanzati/, feb 2026.

[30] Fathiyah Haji Ahmad, S. H. Shah Newaz, D S Sankar, Rudy Ramlie, and Nazmus Shaker Nafi. Fog computing in optical access networks: An energy-efficient and deadline-aware task scheduling mechanism. In *2023 13th International Conference on Information Technology in Asia (CITA)*, pages 66–69, 2023.

[31] S. H. Shah Newaz, Wida Susanty binti Haji Suhaili, Gyu Myoung Lee, Mohammad Rakib Uddin, Alaelddin Fuad Yousif Mohammed, and Jun Kyun Choi. Towards realizing the importance of placing fog computing facilities at the central office of a pon. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 152–157, 2017.

[32] Carmine Cesarano, Alessio Foggia, Gianluca Roscigno, Luca Andreani, and Roberto Natella. Genio: Synergizing edge computing with optical network infrastructures. *IEEE Communications Magazine*, 63(7):154–160, jul 2025.

[33] Diana Goovaerts. Telcos have the perfect edge cloud infrastructure for AI up their sleeve. https://www.fierce-network.com/cloud/telcos-have-perfect-edge-cloud-infrastructure-ai-their-sleeve, oct 2024.

[34] Serhiy O. Semerikov, Tetiana A. Vakaliuk, Olga B. Kanevska, Oksana A. Ostroushko, and Andrii O. Kolhatin. Edge intelligence unleashed: a survey on deploying large language models in resource-constrained environments. *Journal of Edge Computing*, 4(2):179–233, nov 2025.

[35] Senyao Li, Haozhao Wang, Wenchao Xu, Rui Zhang, Song Guo, Jingling Yuan, Xian Zhong, Tianwei Zhang, and Ruixuan Li. Collaborative inference and learning between edge slms and cloud llms: A survey of algorithms, execution, and open challenges, 2025.

[36] Diana Goovaerts. Ziply turns old central offices into data centers. https://www.fierce-network.com/cloud/whats-old-new-again-ziply-turns-old-cos-data-centers, oct 2024.

[37] Vodafone. Deutsche Telekom, Orange, Telefónica, TIM and Vodafone achieve the first pan-european federated edge continuum at MWC 2026. https://www.vodafone.com/news/newsroom/technology/pan-european-federated-edge-continuum, feb 2026.

[38] Michaela Kühn. Milestone for europe's digital sovereignty: European edge continuum goes live. https://www.telekom.com/en/media/media-information/archive/milestone-for-europe-s-digital-sovereignty-1102498, feb 2026.

[39] Electronic Communications Resilience & Response Group (EC-RRG). Telecommunications networks - a vital part of the critical national infrastructure, version 1.1, 2010.

[40] République Française. Rapport d'activité - tome 2: La régulation de l'arcep au service des territoires connectés, édition 2025, jun 2025.

[41] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[42] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: bringing the cloud to the mobile user. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, page 29–36, New York, NY, USA, 2012. Association for Computing Machinery.

[43] Tom Goethals, Filip De Turck, and Bruno Volckaert. Extending kubernetes clusters to low-resource edge devices using virtual kubelets. *IEEE Transactions on Cloud Computing*, 10(4):2623–2636, 2022.

[44] Lars Larsson, Harald Gustafsson, Cristian Klein, and Erik Elmroth. Decentralized kubernetes federation control plane. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 354–359, 2020.

[45] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[46] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, page 1–12, New York, NY, USA, 1987. Association for Computing Machinery.

[47] Aris Leivadeas and Matthias Falkner. A survey on intent-based networking. *IEEE Communications Surveys & Tutorials*, 25(1):625–655, 2022.

[48] Barbara Martini, Molka Gharbaoui, and Piero Castoldi. Intent-based network slicing for sdn vertical services with assurance: Context, design and preliminary experiments. *Future Generation Computer Systems*, 142:101–116, 2023.

[49] Ankur Chowdhary, Abdulhakim Sabur, Neha Vadnere, and Dijiang Huang. Intent-driven security policy management for software-defined systems. *IEEE Transactions on Network and Service Management*, 19(4):5208–5223, 2022.

[50] Tarandeep Kaur and Inderveer Chana. Energy efficiency techniques in cloud computing: A survey and taxonomy. *ACM Computing Surveys*, 48(2):22:1–22:46, oct 2015.

[51] Michal Sedlacko, Andre Martinuzzi, and Karin Dobernig. A systems thinking view on cloud computing and energy consumption. In *Proceedings of the 2014 conference ICT for Sustainability*, pages 95–102. Atlantis Press, aug 2014.

[52] Jonathan Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3), mar 2011.

[53] Ana Radovanovic, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, Saurav Talukdar, Eric Mullen, Kendal Smith, MariEllen Cottman, and Walfredo Cirne. Carbon-aware computing for datacenters, jun 2021.

[54] Young Geun Kim, Udit Gupta, Andrew McCrabb, Yonglak Son, Valeria Bertacco, David Brooks, and Carole-Jean Wu. Greenscale: Carbon-aware systems for edge computing, apr 2023.

[55] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David Irwin, and Prashant Shenoy. Quantifying the benefits of carbon-aware temporal and spatial workload shifting in the cloud, jun 2023.

[56] Noman Bashir, David Irwin, and Prashant Shenoy. On the promise and pitfalls of optimizing embodied carbon. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, pages 1–6, New York, NY, USA, aug 2023. ACM.

[57] Jaylen Wang, Udit Gupta, and Akshitha Sriraman. Peeling back the carbon curtain: Carbon optimization challenges in cloud computing. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, pages 1–7, New York, NY, USA, aug 2023. ACM.

[58] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Act: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 784–799, New York, NY, USA, jun 2022. ACM.

[59] Christoph Buck, Christian Olenberger, André Schweizer, Fabiane Völter, and Torsten Eymann. Never trust, always verify: a multivocal literature review on current knowledge and research gaps of zero-trust. *Computers & Security*, 110, 2021.

[60] World Wide Web Consortium (W3C). Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations, 2022. Accessed on April, 2024.

[61] W3C. Verifiable credentials data model v1.1, 2022. Accessed on April, 2024.

[62] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. A survey on essential components of a self-sovereign identity. *Computer Science Review*, 30:80–86, 2018.

[63] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. Rsvp: A new resource reservation protocol. *IEEE network*, 7(5):8–18, 1993.

[64] Anup Kumar Talukdar, BR Badrinath, and Arup Acharya. Mrsvp: A resource reservation protocol for an integrated services network with mobile hosts. *Wireless Networks*, 7:5–19, 2001.

[65] Xin Wang and Henning Schulzrinne. Rnap: A resource negotiation and pricing protocol. *Transit*, 6(B7):B8, 1999.

[66] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. Rfc3209: Rsvp-te: Extensions to rsvp for lsp tunnels, 2001.

[67] Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, and Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Job Scheduling Strategies for Parallel Processing: 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002*, pages 153–183. Springer, 2002.

[68] Srikumar Venugopal, Xingchen Chu, and Rajkumar Buyya. A negotiation mechanism for advance resource reservations using the alternate offers protocol. In *2008 16th Interntional Workshop on Quality of Service*, pages 40–49. IEEE, 2008.

[69] Erik Elmroth and Johan Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. In *International Workshop on Applied Parallel Computing*, pages 1061–1070. Springer, 2004.

[70] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[71] Ramraj Dangi, Praveen Lalwani, Gaurav Choudhary, Ilsun You, and Giovanni Pau. Study and investigation on 5g technology: A systematic review. *Sensors*, 22(1):26, 2021.

[72] GSMA. Gsma operator platform telco edge requirements 2022, 2022. Accessed on March, 2024.

[73] GSMA. Gsma operator platform group – east-westbound interface apis, 2023. Accessed on March, 2024.

[74] IB Otto, S Steinbuß, A Teuscher, IS Lohmann, A Auer, S Bader, H Bastiaansen, H Bauer, IP Birnstil, and M Böhmer. International data spaces association—reference architecture model—version 3.0, apr 2019.

[75] R. Bias. Architectures for open and scalable clouds, 2022.

[76] CNCF Staff. Cncf survey 2020, 2020.

[77] K. Goldenring. Announcing akri, an open source project for building a connected edge with kubernetes, 2020.

[78] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Gener. Comput. Syst.*, 105(C):475–491, apr 2020.

[79] Joerg Fritsch and Coral Walker. The problem with data. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 708–713, 2014.

[80] Stefano Braghin and Liubov Nedoshivina. MIMO: a framework for extensible and flexible intent-based workload meta-orchestration. In *Proceedings of the 2nd International Workshop on MetaOS for the Cloud-Edge-IoT Continuum*, pages 1–6, 2025.

[81] Jose Manuel Bernabe' Murcia, Eduardo Canovas Martinez, Alejandro Molina Zarca, Stefano Braghin, and Antonio Skarmeta. Cross-domain telemetry architecture: Real-time metrics in the computing continuum. In *Int. Conf. Mobile Internet Secur.*, 2024.

[82] Liubov Nedoshivina, Killian Levacher, Kieran Fraser, Anisa Halimi, and Stefano Braghin. AI-driven workload management in meta os. In *Proc. 1st Int. Workshop MetaOS Cloud-Edge-IoT Continuum*, MECC '24, page 14–20. Association for Computing Machinery, 2024.

[83] Mauro Tortonesi. The compute continuum: Trends and challenges. *Computer*, 58(03):105–108, 2025.

[84] Patrik Hummel, Matthias Braun, Max Tretter, and Peter Dabrock. Data sovereignty: A review. *Big Data & Society*, 8(1):2053951720982012, 2021.

[85] Kristina Irion. Government cloud computing and national data sovereignty. *Policy & Internet*, 4(3-4):40–71, 2012.

[86] Coral Walker and Hassan Alrehamy. Personal data lake with data gravity pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pages 160–167, 2015.

[87] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, apr 2018.

[88] Stefano Galantino, Fulvio Risso, Vlad C. Coroamă, and Antonio Manzalini. Assessing the potential energy savings of a fluidified infrastructure. *Computer*, 56(6):26–34, 2023.

[89] Daniele Cacciabue, Jacopo Marino, Francesco Aglieco, Marco Levorato, Domenico Perroni, and Fulvio Risso. Benchmarking different strategies for offloading ros2 computation to the edge. In *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, pages 49–54, 2024.

[90] Rihab Chaâri, Omar Cheikhrouhou, Anis Koubâa, Habib Youssef, and Tuan Nguyen Gia. Dynamic computation offloading for ground and flying robots: Taxonomy, state of art, and future directions. *Computer Science Review*, 45:100488, 2022.

[91] Jakub Krejčí, Marek Babiuch, Jiří Suder, Václav Krys, and Zdenko Bobovský. Latency-sensitive wireless communication in dynamically moving robots for urban mobility applications. *Smart Cities*, 8(4), 2025.

[92] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015.

[93] A. Manonmani, S. Akash, A. Aswinraj, and A. Manikandan. Literature review on advanced cloud robotics for enhanced industrial automation. In *2025 Int. Conf. Electron. Renew. Syst. (ICEARS)*, pages 46–51, 2025.

[94] Jiafu Wan, Shenglong Tang, Hehua Yan, Di Li, Shiyong Wang, and Athanasios V. Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:2797–2807, 2016.

[95] Victor Casamayor Pujol and Schahram Dustdar. Fog robotics—understanding the research challenges. *IEEE Internet Computing*, 25(5):10–17, 2021.

[96] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018.

[97] Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. Mobile edge computing: Progress and challenges. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 83–84, 2016.

[98] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation*, 2009.

[99] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), 2022.

[100] Loïck Chovet, Gabriel Garcia, Abhishek Bera, Antoine Richard, Kazuya Yoshida, and Miguel Olivares-Mendez. Performance comparison of ros2 middlewares for multi-robot mesh networks in planetary exploration, jul 2024.

[101] Jiaqiang Zhang, Xianjia Yu, Sier Ha, Jorge Peña Queralta, and Tomi Westerlund. Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems. *Journal of Intelligent & Robotic Systems*, 110(4):162, nov 2024.

[102] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. Avital, J. Loudet, C. Guimares, M. Ilyin, and D. Bannov. Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller. In *2023 26th Euromicro Conference on Digital System Design (DSD)*, pages 422–428, Los Alamitos, CA, USA, sep 2023. IEEE Computer Society.

[103] Steve Macenski, Alberto Soragna, Michael Carroll, and Zhenpeng Ge. Impact of ros 2 node composition in robotic systems. *IEEE Robotics and Automation Letters*, 8:3996–4003, 2023.

[104] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *2016 International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2016.

[105] Jaeho Park, Raimarius Delgado, and Byoung-Wook Choi. Real-time characteristics of ros 2.0 in multiagent robot systems: An empirical study. *IEEE Access*, 8:154637–154651, 2020.

[106] Lukas Johannes Dust, Emil Persson, Mikael Ekström, Saad Mubeen, and Emmanuel Dean. Quantitative analysis of communication handling for centralized multi-agent robot systems using ros2. *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, pages 624–629, 2022.

[107] Preetha Thulasiraman, Zhaolin Chen, Bruce Dale Allen, and Brian Bingham. Evaluation of the robot operating system 2 in lossy unmanned networks. *2020 IEEE International Systems Conference (SysCon)*, pages 1–8, 2020.

[108] Sanghoon Lee, Taehun Kim, Jiyeong Chae, and Kyung-Joon Park. Optimizing ros 2 communication for wireless robotic systems. *ArXiv*, abs/2508.11366, 2025.

[109] Sumit Paul, Danh Lephuoc, and Manfred Hauswirth. Performance evaluation of ros2-dds middleware implementations facilitating cooperative driving in autonomous vehicle, 2024.

[110] L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, and R. Dillmann. Performance evaluation of real-time ros2 robotic control in a time-synchronized distributed network. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1670–1676, 2021.

[111] Jorin Kouril, Bernd Schäufele, Ilja Radusch, and Bettina Schnor. Performance evaluation of a ros2 based automated driving system, nov 2024.

[112] Carlos San Vicente Gutiérrez, Lander Usategui San Juan, Irati Zamalloa Ugarte, and Víctor Mayoral Vilches. Time-sensitive networking for robotics, 2018.

[113] Carlos Gutiérrez, Lander Juan, Irati Ugarte, and Víctor Vilches. Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications, sep 2018.

[114] Daniele Cacciabue, Francesco Aglieco, Domenico Perroni, and Fulvio Risso. Stateless job offloading for mobile robots in kubernetes. In *1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (MECC 2024)*, pages 1178–1183, Athens, Greece, apr 2024.

[115] Kaiyuan, Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. Fogros: An adaptive framework for automating fog robotics deployment. In *2021*

IEEE 17th International Conference on Automation Science and Engineering (CASE), page 2035–2042, Lyon, France, aug 2021.

[116] Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Vıctor Mayoral-Vilches, Nikhil Jha, Hugo Zhan, Edith LLontop, Derek Xu, Camilo Buscaron, John Kubiatowicz, Ion Stoica, Joseph Gonzalez, and Ken Goldberg. Fogros2: An adaptive platform for cloud and fog robotics using ros 2. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5493–5500, London, UK, jul 2023.

[117] Kaiyuan Chen, Ryan Hoque, Karthik Dharmarajan, Edith LLontopl, Simeon Adebola, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. Fogros2-sgc: A ros2 cloud robotics platform for secure global connectivity. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Detroit, MI, USA, dec 2023.

[118] Raghav Anand, Jeffrey Ichnowski, Chenggang Wu, Joseph M. Hellerstein, Joseph E. Gonzalez, and Ken Goldberg. Serverless multi-query motion planning for fog robotics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7457–7463, Xi'an, China, may-jun 2021.

[119] Tung V. Doan, Zhongyi Fan, Giang T. Nguyen, Hani Salah, Dongho You, and Frank H. P. Fitzek. Follow me, if you can: A framework for seamless migration in mobile edge cloud. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1178–1183, Toronto, ON, Canada, jul 2020.

[120] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Live service migration in mobile edge clouds. *IEEE Wireless Communications*, 25(1):140–147, feb 2018.

[121] Ahmed Chebaane, Simon Spornraft, and Abdelmajid Khelil. Container-based task offloading for time-critical fog computing. In *2020 IEEE 3rd 5G World Forum (5GWF)*, pages 205–211, Bangalore, India, sep 2020.

[122] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live virtual machine migration with adaptive, memory compression. In *IEEE International Conference on Cluster Computing, ICCC*, pages 1–10, aug-sep 2009.

[123] Syed Kakakhel, Lauri Mukkala, Tomi Westerlund, and Juha Plosila. Virtualization at the network edge: A technology perspective. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 87–92, Barcelona, Spain, apr 2018.

[124] Nitin Sharma, Jitendra Kumar Pandey, and Surajit Mondal. A review of mobile robots: Applications and future prospect. *International Journal of Precision Engineering and Manufacturing*, 24(9):1695–1706, 2023.

[125] Woojae Lee, Jeeho Won, Garam Park, and TaeWon Seo. Mechanical survey on wheeled mobile robot platform for industrial and personal service

robots. *International Journal of Precision Engineering and Manufacturing*, 25(8):1739–1753, 2024.

[126] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, volume 3, pages 2149–2154 vol.3, 2004.

[127] Kaiyuan Chen, Kush Hari, Trinity Chung, Michael Wang, Nan Tian, Christian Juette, Jeffrey Ichnowski, Liu Ren, John Kubiatowicz, Ion Stoica, and Ken Goldberg. Fogros2-ft: Fault tolerant cloud robotics. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1390–1397, 2024.

[128] Kaiyuan Chen, Nan Tian, Christian Juette, Tianshuang Qiu, Liu Ren, John Kubiatowicz, and Ken Goldberg. Fogros2-plr: Probabilistic latency-reliability for cloud robotics, 2024.

[129] Kaiyuan Chen, Michael Wang, Marcus Gualtieri, Nan Tian, Christian Juette, Liu Ren, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. Fogros2-ls: A location-independent fog robotics framework for latency sensitive ros2 applications. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10581–10587, 2024.

[130] Ben Hu, Huaimin Wang, Pengfei Zhang, Bo Ding, and Huimin Che. Cloudroid: A cloud framework for transparent and qos-aware robotic computation outsourcing. In *2017 IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, pages 114–121, 2017.

[131] Kaiyuan Chen, Kush Hari, Rohil Khare, Charlotte Le, Trinity Chung, Jaimyn Drake, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. Fogros2-config: A toolkit for choosing server configurations for cloud robotics. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12083–12089, 2024.

[132] Akhlaqur Rahman, Jiong Jin, Ashfaqur Rahman, Antonio Cricenti, Mahbuba Afrin, and Yu ning Dong. Energy-efficient optimal task offloading in cloud networked multi-robot systems. *Comput. Netw.*, 160:11–32, 2019.

[133] Jan Nouruzi-Pur, Jens Lambrecht, The Duy Nguyen, Axel Vick, and Jörg Krüger. Redundancy concepts for real-time cloud- and edge-based control of autonomous mobile robots. In *2022 IEEE 18th Int. Conf. Factory Commun. Syst. (WFCS)*, pages 1–8, 2022.

[134] Swagata Biswas, Swarnava Dey, and Arijit Mukherjee. (wip) at most m - a flexible redundancy model for cloud robotics. In *2018 IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, pages 577–581. IEEE Computer Society, 2018.

[135] Xiulian Wang, Jinggao Lin, Xi Jin, Changqing Xia, and Chi Xu. A 5g-based switching method between a local controller and an edge controller. In *2023 6th Int. Conf. Intell. Robot. Control Eng. (IRCE)*, pages 7–12, 2023.

[136] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

Appendix A

List of Publications

- **Jacopo Marino**, and Fulvio Risso. "Is the Computing Continuum Already Here?," *9th Italian Conference on ICT for Smart Cities and Communities (I-CiTies 2023)*, University of Salerno, Fisciano (SA), Italy. ArXiv:2309.09822, 2023.
- **Jacopo Marino**, Fulvio Risso, and Mauro Bighi, "Dynamic Optimization of Provider-Based Scheduling for HPC Workloads," *2023 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Croatia, 2023, pp. 1-6, doi: 10.23919/SoftCOM58365.2023.10271608.
- **Jacopo Marino**, Leonardo Camiciotti, Francesco Cheinasso, Alessandro Olivero, and Fulvio Risso. 2023. "Enabling Compute and Data Sovereignty with Infrastructure-Level Data Spaces," *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum (eSAAM '23)*. Association for Computing Machinery, New York, NY, USA, 77–85. https://doi.org/10.1145/3624486.3624509
- Stefano Galantino, Elisa Albanese, Nasir Asadov, Stefano Braghin, Francesco Cappa, Andrea Colli-Vignarelli, Amjad Yousef Majid, Eduard Marin, **Jacopo Marino**, Lorenzo Moro, Liubov Nedoshivina, Fulvio Risso, Domenico Siracusa, Antonio Skarmeta, and Luca Zuanazzi, "Building the Cloud Continuum with REAR," *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, Saint Louis, MO, USA, 2024, pp. 67-72, doi: 10.1109/NetSoft60951.2024.10588885.

- Daniele Cacciabue, **Jacopo Marino**, Francesco Aglieco, Marco Levorato, Domenico Perroni, Fulvio Risso, "Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge," *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, Saint Louis, MO, USA, 2024, pp. 49-54, doi: 10.1109/NetSoft60951.2024.10588914.

- Stefano Galantino, **Jacopo Marino**, Fulvio Risso, Stefano Braghin, Liubov Nedoshivina, Antonio Fernando Skármeta Gómez, Domenico Siracusa, "Edge-to-Cloud Continuum Made Real," *Computer*, vol. 59, no. 3, pp. 51-59, March 2026, doi: 10.1109/MC.2025.3607210.

- Andrea Fasano, Daniele Cacciabue, Stefano Galantino, **Jacopo Marino**, and Fulvio Risso, "SM2: Towards Reliable and Scalable Robotics Middleware Beyond ROS2," *Workshop on 6G Connected Robotics for Collaborative Control, Sensing, and Communication (Second Edition)*, collocated with *IEEE International Conference on Communication (ICC)*, Glasgow, Scotland, UK, May 2026.