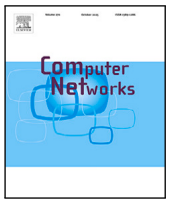
Computer Networks 286 (2026) 112443



Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet





End-to-end latency assurance for distributed augmented reality over programmable 6G networks: A DESIRE6G demonstration

Francesco Paolucci [a], Emilio Paolini [b,*], Faris Alhamed [b], Massimo Satler [b], Domenico Uomo [b], Michelangelo Guaitolini [a], Pol Gonzalez [c], Marc Ruiz [c], Luis Velasco [c], Sandor Laki [d], David Kis [d], Gergely Pongracz [e], Attila Mihaly [e], Anestis Dalgkitsis [f], Chrysa Papagianni [f], Anastassios Nanos [g], Stephen Parker [h], Vincent Lefebvre [i], Mark Angoustures [i], Juan Jose Vegas Olmos [j], Andrea Sgambelluri [b]

[a] CNIT, Via G. Moruzzi 1, 56124, Pisa, Italy
[b] Scuola Superiore Sant'Anna, Via G. Moruzzi 1, 56124, Pisa, Italy
[c] Universitat Politècnica de Catalunya, C. Jordi Girona 1-3, 08034, Barcelona, Spain
[d] Eötvös Loránd University, Pázmány P. sétány 1/C, H-1117, Budapest, Hungary
[e] Ericsson Research, H-1117, Budapest, Hungary
[f] University of Amsterdam, Science Park 900, 1098 XH, Amsterdam, The Netherlands
[g] Nubificus, 88 Queen Street, S1 2FW, Sheffield, United Kingdom
[h] Accelleran, Quellinstraat 49, 2018, Antwerp, Belgium
[i] Tages Solidshield, 06110, Le Cannet, France
[j] NVIDIA, 4000, Roskilde, Denmark

ARTICLE INFO

Keywords:
6G
In-band telemetry
Multi-agent systems
Programmable networks
Cloud-native orchestration
Augmented reality
Edge computing
Open RAN

ABSTRACT

6G networks are expected to deliver ultra-low latency, high reliability, and real-time intelligence for emerging services such as interactive Augmented Reality (AR), autonomous robotics, and digital twins. Achieving these requirements in practice demands tight coordination between networking, computing, and control domains, spanning RAN, transport, edge, and cloud. However, current 5G deployments lack pervasive telemetry, fine-grained observability, and automated control mechanisms capable of reacting at the time scales required by latency-sensitive applications.

This paper presents a full integrated demonstration of DESIRE6G, a cloud-native 6G-ready architecture that leverages programmable data planes with P4 for flexible routing and telemetry using an implementation of novel data plane protocols, achieves distributed optimization of service deployment and runtime monitoring and reconfiguration via secure multi-agent systems (MAS), combined with intent-based orchestration layer for end-to-end service assurance. The system is validated on the federated ARNO testbed using a real distributed AR application involving a remotely-controlled drone as a User-Equipment that is equipped with a camera streaming a live video through the DESIRE6G network to a Kubernetes edge cluster that executes serverless inference functions for object detection and recognition, the video is then augmented with object information and shown on a Quest 3 AR headset. The MAS monitors the end-to-end latency in real time through P4 Telemetry and responds to changes in network conditions by reconfiguring the affected segments, while the Kubernetes monitoring provides real-time visibility and scalability across different segments.

Overall, three hierarchical service assurance loops are demonstrated: (i) In-Network Control (INC) executing microsecond-scale congestion recovery in the P4 data plane, (ii) Infrastructure Management Layer (IML) performing millisecond-scale function migration and scaling, and (iii) MAS-driven cross-domain optimization operating at sub-second time scales to resolve RAN latency anomalies.

Evaluation results show stable end-to-end latency in the 15–25 ms range in steady-state conditions, with fast recovery during induced congestion ≤ 1.5 ms data plane reroute via P4 INC.

* Corresponding author.
E-mail address: emilio.paolini@santannapisa.it (E. Paolini).

https://doi.org/10.1016/j.comnet.2026.112443
Received 4 March 2026; Received in revised form 29 May 2026; Accepted 29 May 2026
Available online 8 June 2026
1389-1286/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Interactive and mission-critical services such as immersive Augmented Reality (AR), autonomous robotics, remote piloting, and collaborative digital twins are expected to drive the next wave of traffic and intelligence over future 6G networks [1]. These applications impose stringent latency, jitter, and reliability constraints, often requiring end-to-end delays below 20 ms while sustaining high throughput, continuous telemetry feedback, and adaptive computational offloading across heterogeneous hardware resources [2]. To meet such demands, 6G infrastructures must provide pervasive observability, programmable data paths, and cloud-native orchestration mechanisms that cooperatively operate across the Radio Access Network (RAN), the transport network, and the edge/cloud computing layers [1,3].

However, current 5G deployments only partially satisfy these requirements. While the 3GPP and O-RAN ecosystems have introduced virtualization, functional disaggregation, and service-based architectures, their telemetry capabilities remain limited in both granularity and reaction time [4]. Traditional monitoring relies primarily on periodic control-plane reports and centralized decision loops, leading to slow responsiveness when faced with congestion, link degradation, or resource exhaustion [5]. As a result, state-of-the-art 5G networks struggle to provide fast, fine-grained assurance for latency-critical applications, particularly when computation is split across user devices, edge servers, and cloud platforms.

To bridge this gap, the research community has recently explored a series of complementary enablers: (i) programmable data planes based on P4 and in-band telemetry (INT), enabling sub-millisecond network visibility; (ii) in-network control mechanisms capable of reacting at microsecond time scales; (iii) serverless and cloud-native orchestration engines for dynamic function scaling; and (iv) distributed intelligent agents for learning-based decision-making [6–9]. Yet, despite significant progress, very few experimental test beds demonstrate the integration of these technologies into a single, coherent 6G-ready architecture capable of sustaining real applications under real impairments.

Current 6G research is still fragmented. Most studies focus on specific areas such as RAN power efficiency, AI models for security, or new control plane designs, without deeper look at how these different parts work together. In DESIRE6G our goal is to move past these individual solutions and provide a full-system, cloud-native architecture. In this paper, we demonstrate how the system handles end-to-end monitoring from the user equipment to the edge cluster using programmable data planes. This setup allows us to reconfigure telemetry on demand based on what an application needs. We use a secure multi-agent system (MAS) to coordinate the different network segments, which keeps QoS requirements by automatically rerouting flows or triggering handovers when conditions change. To show it works in a real-world scenario, we also deployed distributed serverless applications at the edge to meet the low-latency and scaling requirements of Augmented Reality.

The demo is validated on the ARNO testbed located in Pisa, Italy, which includes 5G Open RAN equipment, a P4-enabled SDN metro network, an edge computing cluster, multiple NVIDIA Jetson nodes, and a Quest 3 AR headset. The demonstrator showcases a real distributed AR application executed across a drone, the edge cluster, and the headset, while telemetry and automated control mechanisms preserve low end-to-end latency under adverse network and compute conditions.

The contribution of this work is fourfold:

- We implement a multi-domain 6G-ready framework that integrates intent-based orchestration (SMO), programmable P4 data planes, and pervasive telemetry spanning user devices, RAN, SDN, and edge domains.
- We introduce a hierarchical service assurance model with three reaction layers: (i) microsecond-scale in-network control (INC), (ii) millisecond-scale scaling and migration via the Infrastructure Management Layer (IML), and (iii) MAS-driven cross-domain optimization with secure attestation.
- We deploy a distributed AR pipeline leveraging serverless functions running across drone and edge resources, maintaining real-time rendering on a Quest 3 headset.
- We experimentally validate the full architecture in a real multi-room testbed, demonstrating end-to-end latencies in the 15–25 ms range, fast recovery (i.e., $\lesssim$1.5 ms) from induced congestion, computational overload, and RAN latency anomalies.

Beyond the architectural design, this paper provides a full experimental validation of the approach in a real-world deployment including RAN, SDN, edge computing, and a distributed AR application. This offers reproducible insights into how control layers operating at different time scales interact. Rather than introducing new individual network components, this work presents a novel *system-level design and validation* of a hierarchical service assurance architecture for latency-critical applications. The key contribution lies in demonstrating how heterogeneous control loops—spanning programmable data planes (INC), infrastructure orchestration (IML), and cross-domain intelligence (MAS)—can be coherently integrated through in-band telemetry to achieve real-time, end-to-end service adaptation. Table 1 summarizes the current gaps in 3GPP standards and our proposals to bridge these gaps that are presented in this paper.

The rest of this paper is organized as follows: Section 3 summarizes the DESIRE6G architectural components. Section 4 details the distributed AR application. Section 5 presents the hierarchical assurance workflows. Section 6 describes the ARNO testbed and implementation. Section 7 reports experimental results. Section 8 concludes the paper and outlines future work.

2. Related work

The technological foundations for 6G service assurance span several research domains: distributed telemetry, programmable data planes, cloud-native orchestration, and multi-agent intelligence. While each has progressed independently, very few experimental platforms demonstrate their integration into a unified architecture capable of sustaining real-time XR or autonomous robotics workloads. This section reviews the most relevant directions.

2.1. Programmable networking and in-band telemetry

P4-programmable data plane frameworks have introduced unprecedented levels of control over forwarding behavior, enabling line-rate packet parsing, matching, modification, and monitoring [6]. In-band Network Telemetry (INT) extends this capability by embedding per-hop metadata into packets to expose queue occupancy, timestamping, and path evolution directly to end hosts or collectors [10]. Several studies have shown that INT offers greater precision and lower reaction time compared to SNMP, NetFlow, or passive probes [11]. However, most existing INT deployments stop at data collection, i.e., corrective actions remain centralized, slow, or limited to offline analysis. By contrast, DESIRE6G provides novel INT and *In-Network Control (INC)* frameworks for monitoring and rerouting of traffic. DESIRE6G INT starts from the UE and can be enabled for target flows and services with custom metadata depending on the application needs. While DESIRE6G INC provides a novel solution for In-network rerouting decision at the core network and transport segment, where programmable switches trigger autonomous rerouting at microsecond scale without involving a controller.

2.2. Open RAN and edge-assisted XR

The O-RAN Alliance has driven functional disaggregation and virtualization in the RAN, enabling near-real-time optimization via the RAN Intelligent Controller (RIC) and AI-enabled xApps [4]. Multi-access Edge Computing (MEC) brings the storage and processing of data to

Table 1
Gaps in current 3GPP standards and our proposals to bridge them.

Domain	3GPP 5G	Our Proposal
End-to-End latency monitoring	Focuses on average ignoring jitter, and is network centric with the UE not taking active part in the measurement	offers per packet monitoring accounting for jitter starting with the UE
Core network reaction time	Reaction time between 100 ms to 2000 ms	micro-second reaction time with INC
Service management and orchestration	Lack of automated Intent-Based Networking (IBN)	Provides API calls through the SMO for implementing IBN
AI/ML Interoperability	Lacks standards for managing AI models from different vendors	Offers a Multi-Agent System (MAS) for monitoring and configuration on a per-service fashion with a security layer and decentralized attestation mechanism

the edge of cellular networks instead of having it hosted on the cloud, bringing it closer to the data sources. In this context, previous work has studied MEC-assisted XR offloading and prediction of motion-to-photon delay on 5G slices, but most efforts rely on static policies, centralized scheduling, or limited visibility into transport-layer bottlenecks [12]. Recent test beds integrate O-RAN with edge offloading [13], but do not exploit programmable data-planes or telemetry feedback from intermediate network domains. In our work we leverage the programmability of the RIC by building and xAPP that offers an API for an AI agent that collects metrics from the RAN segment in addition to collecting per-packet latency information through the P4 telemetry offering fast reaction time for worsening network conditions by forcing handovers.

2.3. Cloud-native and serverless orchestration

Kubernetes-based orchestration has become central to cloud-native 6G architectures for scaling VNFs, allocating computing resources, and migrating microservices [8,14]. Existing platforms demonstrate elastic instantiation of VNFs and AI functions but typically react on second-level time scales due to centralized control or periodic metric reporting [15]. DESIRE6G incorporates a two-level orchestration model: (i) container-level monitoring using pervasive Kafka feeds and (ii) an Infrastructure Management Layer (IML) that performs millisecond-level function migration and resource reallocation, triggered by real-time network and compute telemetry.

2.4. Multi-agent systems for network intelligence

Multi-agent control has emerged as a promising mechanism for distributed optimization in wireless and edge computing environments, enabling localized decision-making and collaborative learning [16]. However, most MAS implementations assume trusted agents and overlook the need for attestation, tamper detection, or secure coordination across domains. The DESIRE6G framework employs the D-MUTRA security layer, a decentralized attestation mechanism that ensures trustworthy MAS coordination using a blockchain-backed smart contract for key distribution and verification. To the best of our knowledge, this is the first MAS platform that combines secure on-boarding, runtime attestation, and closed-loop optimization in a real multi-domain 6G testbed.

2.5. Experimental XR testbeds

Existing XR networking experiments typically focus on single-segment optimization, i.e., RAN-only adaptations, MEC-only inference offloading, or local Wi-Fi-based streaming [17,18]. End-to-end experiments across radio, transport, and edge domains remain rare, especially those involving distributed AR pipelines and telemetry-driven self-healing. The closest related demonstrations integrate 5G RAN with edge inference but do not include programmable dataplanes or hierarchical assurance. Our work is the first to validate a multi-layer service assurance stack, INC, IML, MAS, with real XR traffic on physical RAN, SDN, and edge infrastructure.

In summary, prior research has advanced each enabling technology in isolation. DESIRE6G distinguishes itself by unifying programmable switching, in-band telemetry, secure MAS, Open RAN, and cloud-native orchestration into a single system, experimentally demonstrating reliable AR over a federated 6G-ready infrastructure.

3. DESIRE6G architecture

The DESIRE6G architecture is designed to provide end-to-end service assurance for latency-critical applications by tightly integrating programmable data planes, pervasive telemetry, cloud-native orchestration, and secure multi-agent control. The system operates across four main domains: (i) the user and radio access network (RAN), (ii) the transport SDN fabric, (iii) the edge/cloud computing layer, and (iv) the management and orchestration plane. Fig. 1 provides a conceptual view of the main building blocks. The dash-tagged blocks are fully implemented and integrated in the Augmented Reality application demonstration.

At a high level, the architecture supports hierarchical decision loops that react at different time scales. Sub-millisecond responses are handled directly in the data plane; millisecond-scale decisions are managed by the Infrastructure Management Layer (IML); and global, cross-domain optimizations are coordinated by secure Multi-Agent Systems (MAS) and the Service Management and Orchestration (SMO) framework. In the following, we describe each architectural layer and the main components.

3.1. Programmable data plane and in-band telemetry

The data plane is composed of P4-programmable switches and P4-enabled endpoints (including the drone UE) supporting real-time monitoring and control. The architecture provides several infra-network functions that can be deployed along the desired service chain. The mandatory ones are the UE-to-Service Mapper (UE2SM) function and the Network Function Routing (NFR) function. The former is responsible to match UE traffic to an existing service and assign a DESIRE6G service label that will be used to route the service along the desired chain to reach network and application functions. The latter is used at each switching point to steer the traffic based on the service label. Packet encapsulations are programmed directly in the match–action pipeline, enabling per-service forwarding behavior, header insertion, and flow isolation.

To achieve fine-grained observability, including latency monitoring and control, DESIRE6G employs *In-Band Network Telemetry* (INT) as per-service network functions for end-to-end service monitoring, including the user equipment [10]. At the UE, a P4-based UE2SM module attaches the DESIRE6G service header together with the first INT metadata fields (timestamp, UE identity). As packets traverse switches

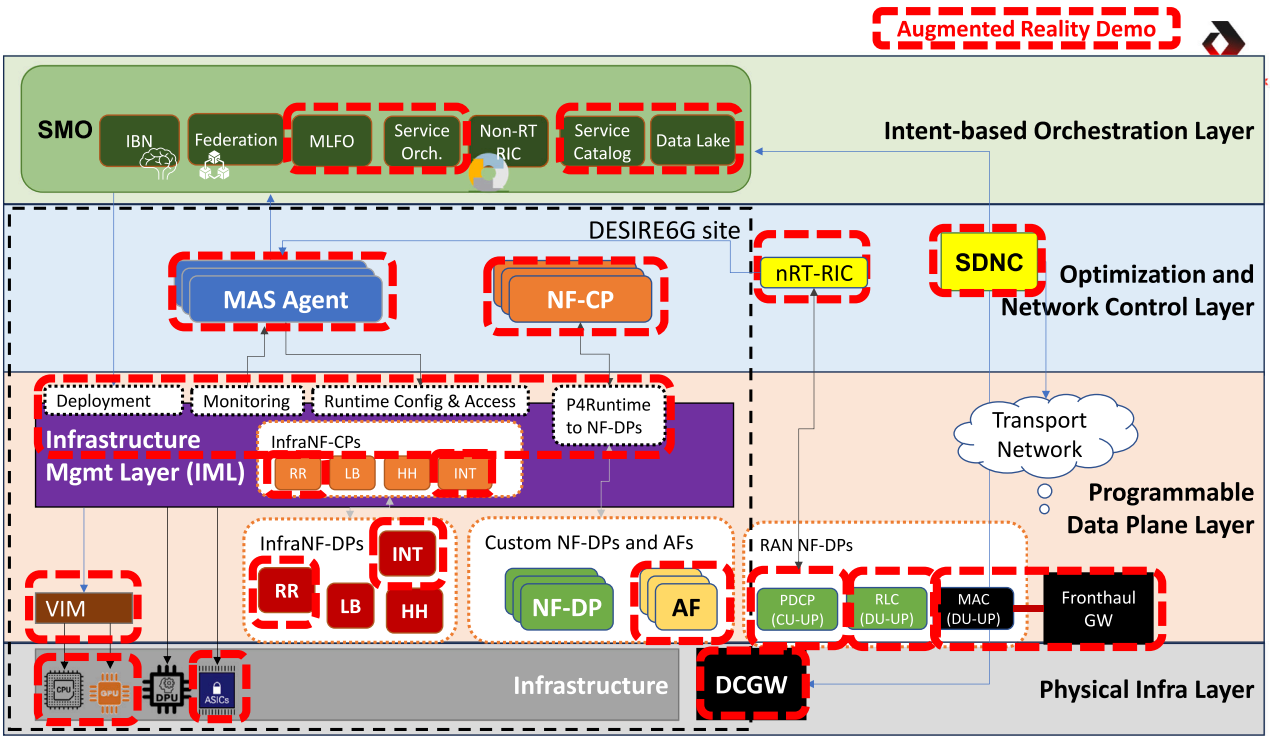


Fig. 1. DESIRE6G functional architecture of cloud-native programmable 6G.

in the RAN and SDN domains, intermediate nodes append per-hop latency, queue occupancy, and interface identifiers. INT information is exported to the Telemetry Collector for correlation and storage, enabling end-to-end performance debugging and root cause detection.

The transport fabric may also support *In-Network Control* (INC), a data-plane mechanism that allows P4 switches to autonomously react to congestion or latency violations detected locally at the switch level or within a switch network domain. A *Data Plane Active Collector* (DPAC) aggregates local INT signals in real time and installs new flow rules directly inside the network, bypassing the controller. INC therefore delivers microsecond-scale reactions for fast recovery during congestion or link degradation [19].

3.2. Open RAN and RIC integration

The access network follows a fully disaggregated O-RAN-compliant architecture. Radio Units (RU) are connected via eCPRI front-haul links to virtual Distributed Units (vDU), while the virtual Centralized Unit (vCU) and 5G Core Network (CN) operate as cloud-native containers. The RAN near real time Intelligent Controller (nRT-RIC) provides control through dedicated and programmable xApps that supervise scheduling, mobility management, and radio configuration through the CU-User Plane (CU-UP) interfaces.

The DESIRE6G framework enhances network resilience by aggregating telemetry information from the DU, CU, and the UE via in-band network telemetry to provide a granular view of per-cell performance and localized radio latency. By utilizing this comprehensive telemetry data, the multi-agent system can identify complex network anomalies, such as midhaul transport impairments, that remain invisible to standard container-level monitoring tools. Once such an issue is detected, the system initiates intelligent, RIC-driven interventions through a custom API provided by the xAPP provide proactive DU-to-DU handovers or dynamic steering of traffic across specific network slices. This architecture ultimately facilitates a synchronized response where the radio and transport layers execute distinct but harmonized corrective actions to maintain end-to-end service stability.

3.3. Infrastructure Management Layer

The Infrastructure Management Layer (IML) provides cloud-native lifecycle management for distributed network functions operating at different backends. Each IML instance interacts with the local VIM (e.g. Kubernetes) cluster to scale microservices, migrate distributed functions, or provision new network functions. Moreover, it interacts with software and hardware backends to deploy P4 infra-network or custom functions along the service chains (e.g., routing, load balancers, heavy hitters, telemetry) [20]. Unlike traditional orchestrators that rely on periodic reports, DESIRE6G uses real-time Kafka telemetry feeds combined with forecasting models to predict SLA violations before they occur [21]. When resource saturation, queue buildup, or CPU overload is detected, the IML can trigger: (i) migration of the AR processing container to another node, (ii) dynamic reallocation of GPU/CPU resources, (iii) traffic steering commands to the SDN controller. These reactions operate at millisecond time scales and ensure continuity of service without user-visible degradation. It is important to distinguish between control-plane reaction time and execution time in the IML layer. The millisecond-scale latency associated with the IML refers to the decision-making and orchestration triggering process (e.g., scaling or migration decisions), whereas some actual enforcement of these actions, such as Kubernetes pod instantiation or migration, may occur at a longer timescale, typically in the order of seconds. This distinction is fundamental for correctly interpreting the hierarchical control loops presented in this work.

3.4. Secure multi-agent systems

The intelligent control plane of DESIRE6G is composed of a set of secure MAS that perform distributed analytics, optimization, and decision-making based on ML algorithms. Each agent observes a specific network segment, i.e., RAN, SDN, or edge, and receives telemetry streams from data-plane INT or infrastructure monitoring. Agents exchange information through a distributed ledger implemented with a smart contract, ensuring trust, non-repudiation, and auditability [22].

This ledger-based mechanism is not part of the latency-critical control loop. Its role is limited to supporting trust management among distributed MAS components, namely agent onboarding, integrity verification, and secure key distribution, so that unauthorized or tampered agents cannot participate in cross-domain coordination.

A Multi-Agent System (MAS) consists of autonomous agents that interact with the environment and cooperate through information exchange to solve problems beyond the capability of a single agent. Its design requires defining both the agents' learning capabilities and perception, and the interaction model among them, including what information is shared and how coordination is achieved. A MAS solution for Network Service (NS) traffic routing was proposed in our previous work in [23], where a detailed architecture of the agents and their relation with the rest of the components in the SDN control plane was proposed. The definition of a MAS for NS traffic routing included: (i) a flow agent running at the ingress location that interacts with the network node to collect measurements and provide new routing policies; (ii) a telemetry agent running at the egress location that collects e2e telemetry measurements and shares them with the flow agent; and (iii) a supervisor running in the centralized Machine Learning Function Orchestrator (MLFO). DESIRE6G implements a distributed collaborative MAS for near-real-time network service operation [24]. Agents are deployed at selected network locations, while a centralized MAS Supervisor runs within the MLFO, which interacts with the network controller throughout the service life cycle (deployment, operation, reconfiguration, and termination). During operation, agents collect telemetry from the programmable data plane to monitor KPIs and optimize a cost function that penalizes SLA violations and excessive resource usage. In the case of latency-critical services, the target KPI is the maximum end-to-end delay. Each agent performs telemetry aggregation, short-term modeling, and decision-making based on Deep Reinforced Learning (DRL) to determine the routing policy at each domain. If delay exceeds the threshold, countermeasures are taken to detect the target recovery domain and suggest the most proper action (i.e., optical bypass, traffic steering, RAN Handover, etc.).

The DRL engine adopted in this work was originally defined and evaluated in [24], where the state and action spaces, as well as the reward formulation, are fully described. The state $s_o(t) \in \mathbb{R}^+$ represents the NS traffic condition normalized with respect to the available path capacity. Instead of relying on the instantaneous observed traffic $x_{NS}(t)$, the DRL agent operates on a short-term traffic prediction $x(t)$, improving stability under dynamic conditions. The state is defined as:

$$s_o(t) = x(t) \cdot \frac{\sum_{i \in I} k_i}{|I|} \tag{1}$$

where I denotes the set of available paths, k_i is the capacity of path i, and $|I|$ is the number of available paths. The action $a_o(t) \in [0,1]^{|I|}$ is a vector specifying the traffic allocation across the available paths:

$$a_o(t) = \{a_i(t)\}_{i \in I}, \quad \sum_{i \in I} a_i(t) = 1 \tag{2}$$

where each component $a_i(t)$ represents the fraction of $x_{NS}(t)$ routed through path i. The reward function is defined as:

$$r_o(t) = \alpha_{\text{delay}} \cdot r_{\text{delay}}(t) + \alpha_{\text{cost}} \cdot r_{\text{cost}}(t) \tag{3}$$

where $r_{\text{delay}}(t)$ penalizes actions leading to end-to-end delay $d_{NS}(t)$ exceeding a maximum threshold $d_{\max}$, while $r_{\text{cost}}(t)$ accounts for the associated network cost. DRL models are pre-trained offline within a sandbox environment managed by the Machine Learning Function Orchestrator (MLFO). A model manager continuously generates representative training scenarios and stores the resulting models in a database. At deployment time, the MLFO receives from the Service Orchestrator the characteristics of the requested network service, including traffic profile, expected load, latency requirements, and the set of available paths with their corresponding capacities and costs. Based on this information, the most suitable pre-trained model is selected from the database and used for runtime decision support. The DRL engine builds upon previously validated models [23], where its effectiveness compared to traditional routing approaches has been demonstrated. Previous work in [25] investigated the autonomous adaptation of virtual link capacity as a function of the input traffic and compared the performance of DRL against a threshold-based reactive heuristic. In that work, the DRL agent analyzed the input traffic and the current link capacity to determine the target capacity that should be allocated for the next period according to the learned policy. Conversely, the threshold-based approach reactively added or released capacity to maintain the link utilization within a predefined range. Experimental results showed that the threshold-based method, despite careful tuning, suffered from traffic losses and increased delay due to its reactive nature. In contrast, DRL-based policies learned adaptive behaviors that better matched traffic dynamics and avoided service degradation. The DRL engine builds upon these previously validated results and is integrated within the DESIRE6G orchestration framework. In this work, it is integrated within the MAS to enable adaptive and data-driven decision-making in a real deployment targeting the AR use case.

The interaction among components follows a structured control flow: telemetry data collected in the data plane is processed by the MAS, which performs anomaly detection and decision-making (optionally supported by DRL), and then triggers domain-specific actions via the SDN controller or the RAN Intelligent Controller (RIC), depending on the location of the detected issue (see Workflows 3 and 5 in Section 5).

A dedicated *SECaaS* (Security-as-a-Service) module performs code wrapping and cryptographic attestation of agents. During on-boarding, the binary of each agent is scanned, signed, and validated. During execution, agents periodically re-attest to guarantee runtime integrity. The keys for secure MAS communication are distributed through the same smart contract. This eliminates the risk of rogue agents injecting false telemetry or malicious commands.

3.4.1. Cross-domain interfaces and protocols

The interaction among DESIRE6G components across RAN, SDN, and edge domains is enabled through a set of well-defined interfaces and protocols.

At the SDN/IML level, programmable switches are controlled via P4Runtime interfaces using gRPC, allowing the SDN controller/IML to install forwarding rules, configure telemetry behaviors, and enforce traffic steering policies. The SDN controller/IML exposes northbound REST APIs that are used by higher-layer components, including the MAS, to request reconfiguration actions such as path updates or traffic redistribution.

At the RAN level, the interaction between the MAS and the RAN Intelligent Controller (RIC) is realized through the O-RAN xApp interface. This enables the MAS to trigger control actions such as handover decisions or policy updates when latency degradation is detected in the radio segment. The RIC internally translates these requests into RAN-specific control procedures.

Telemetry data is collected through in-band network telemetry (INT) mechanisms in the data plane and exported to telemetry collectors. These data streams are delivered to MAS agents through a distributed messaging infrastructure (i.e., Kafka-based pipelines), enabling near real-time analytics and anomaly detection.

Finally, interactions between MAS agents and infrastructure-level components (IML and SDN controller) rely on lightweight REST-based APIs, ensuring interoperability across heterogeneous domains. This combination of data-plane programmability, standardized RAN interfaces, and cloud-native messaging enables the coordination of cross-domain control loops in the DESIRE6G architecture.

Table 2
Hierarchical control loops in the DESIRE6G architecture.

Layer	Scope	Time	Function	Limitations if removed
INC	Per-flow/ link	µs	Fast rerouting, congestion control	No real-time reaction, latency spikes
IML	Site (edge/cloud)	1 ms/ 100 ms	Scaling, placement	No resource adaptation, SLA violations
MAS	Cross-domain	10 ms–1 s	Global optimization, anomaly detection	No cross-domain coordination
SMO	End-to-end	s–min	Service re-optimization (non-runtime)	Suboptimal long-term allocation

3.5. Service management and orchestration

The SMO acts as the global brain of the architecture. It offers northbound APIs to tenants, supports intent-based service descriptions, and exposes a service catalog with available network functions. Once a service request is received, the SMO retrieves the topology state and executes the ML-based Optimization Engine (OE) to determine the placement and decomposition of the service, from the high-level graph with latency constraints to site-specific network service descriptors [26]. The SMO coordinates the IML, MAS, and SDN controller, ensuring that local corrective actions remain globally optimal. It also collects telemetry into a Data Lake and performs long-term optimization, resource planning, and SLA validation. While SMO operates at a slower timescale and is not involved in runtime adaptation, it complements the faster control loops by enabling long-term service re-optimization and redeployment.

3.6. Hierarchical service assurance model

A distinctive feature of DESIRE6G is its multi-layer assurance stack: (i) *Microsecond layer:* autonomous rerouting and congestion mitigation through INC entirely inside the P4 switches. (ii) *Millisecond layer:* workload scaling and migration triggered by IML after predictive monitoring. *Sub-second layer:* MAS coordination across RAN and SDN segments for anomaly resolution and cross-domain re-optimization.

Table 2 highlights the complementary role of each control layer across different time scales, from real-time data-plane reactions to slower service re-optimization, and the limitations that arise when any layer is removed.

This hierarchy ensures that localized perturbations (e.g., link congestion) are handled instantly in the data plane, while more complex anomalies (e.g., RAN mid-haul delays, resource exhaustion) are managed by intelligent agents and the orchestrator. The combination of these three loops enables resilient end-to-end enforcement of strict latency targets for distributed AR workloads. All these loops will be described in terms of workflow and experimental evaluation. In this way, the DESIRE6G platform aims at unifying programmable switching, pervasive telemetry, secure distributed intelligence, and cloud-native orchestration into a single architecture optimized for latency-critical 6G services. The next section presents its implementation on the ARNO testbed and the deployment of a real distributed AR application.

4. Augmented reality application

To validate the DESIRE6G architecture under a real latency-critical workload, we implemented a distributed AR application in which a drone and an AR-equipped operator collaborate to perform a remote visual inspection task. A distinctive feature of the application is the operator's ability to perceive augmented objects even when they are partially or fully occluded by physical obstacles. Since object detection and pose estimation are performed from the drone's aerial perspective, the system can identify entities located behind walls, barriers, or other obstructions that are not directly visible from the operator's viewpoint. The edge-based reconstruction stage fuses spatial information and projects the detected objects into the operator's coordinate frame. As a result, the headset can render stable overlays that appear "locked" in the correct physical position, effectively providing enhanced situational awareness beyond line-of-sight constraints. This capability is particularly relevant in surveillance and emergency-response scenarios, where operators must reason about hidden or partially visible entities while maintaining spatial context.

The application pipeline is shown in Fig. 2 and partitioned across three execution stages, on-boarded at the drone (AR1), at the edge cluster (AR2), and on the AR headset (AR3), respectively. The AR service is orchestrated, deployed and monitored by the DESIRE6G control and data plane mechanisms.

4.1. Onboard drone processing (AR1)

AR1 executes on the NVIDIA Jetson NX embedded in the drone payload. It performs real-time perception directly at the data source. Video is acquired from dual on-board cameras and processed through a hardware-accelerated inference pipeline based on YOLO and ResNet models [27,28], optimized using NVIDIA TensorRT [29]. Video frames are encoded using a GStreamer pipeline (H.264 over RTP/UDP), while semantic metadata such as bounding boxes and keypoints are transmitted through the Zenoh publish/subscribe protocol. The AR1 stage is intentionally lightweight, ensuring minimal on-board processing delay and reduced wireless transmission overhead.

4.2. Edge processing (AR2)

This AR application module performs computationally intensive tasks that complement the on-board processing stage. These include refined pose estimation, multi-person keypoint fusion, semantic segmentation, scene enhancement, and partial 3D spatial reconstruction. By offloading heavier inference stages to the edge, AR2 balances compute distribution between drone and infrastructure while maintaining strict end-to-end latency requirements. The containerized implementation allows AR2 instances to be dynamically scaled or migrated when required by service assurance mechanisms.

4.3. AR headset rendering (AR3)

AR3 runs natively on a Meta Quest 3 headset using Unreal Engine 5 and OpenXR. It enables pass-through augmented visualization by overlaying semantic annotations onto the real-world environment observed by the operator. A custom C++ receiver ingests metadata generated by AR1 and AR2 and renders spatially aligned 2D and 3D overlays. Bidirectional communication enables operator feedback, such as navigation commands or camera adjustments, to be transmitted back to the drone, thereby closing the interactive control loop.

4.4. AR as DESIRE6G service

The distributed AR pipeline spans three main segments: wireless transmission from the drone to the RAN, programmable SDN transport from the RAN to the edge, and edge processing followed by delivery to the headset. The cumulative bidirectional latency budget must remain below approximately 20–25 ms to preserve motion-to-photon stability and immersive user experience. Maintaining this bound requires continuous supervision of both networking and compute resources. This responsibility is handled by the DESIRE6G platform. While the AR modules implement perception and rendering logic, DESIRE6G ensures deterministic service delivery through programmable routing

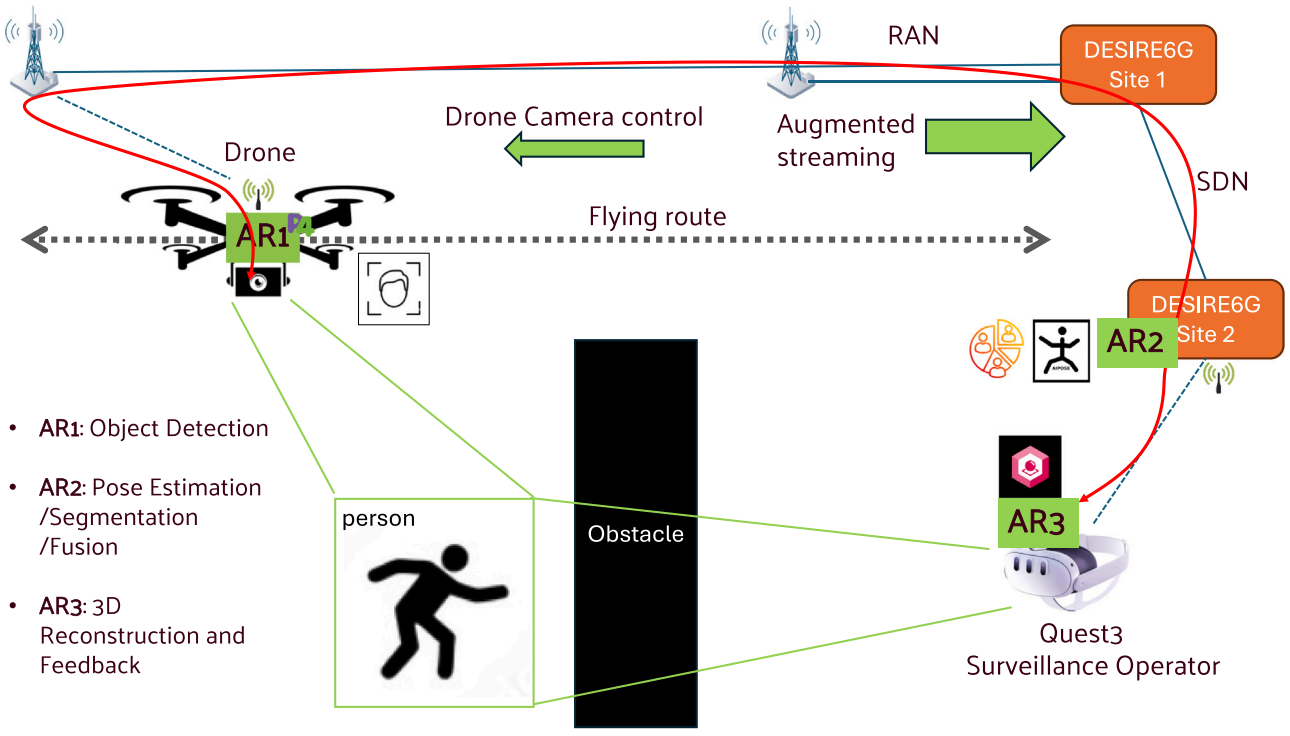


Fig. 2. Augmented Reality Application: distributed tasks and interactions.

and pervasive telemetry. Application traffic generated by AR1 enters the DESIRE6G domain through the UE2SM, which encapsulates packets with a dedicated DESIRE6G header. NFR, implemented in P4-programmable switches steers traffic across the 5G O-RAN domain, the programmable SDN fabric, and the edge ingress switches. Forwarding policies are dynamically installed and managed by the SDN controller in coordination with the IML. This design allows path reconfiguration during congestion or degradation events without modifying application components. DESIRE6G employs pervasive INT to monitor performance at packet granularity. A first telemetry header (UE-INT) is attached at the drone, embedding generation timestamps. As packets traverse the RAN and SDN switches, each hop appends metadata such as processing delay and queue occupancy. Telemetry information is exported to the Telemetry Collector at the edge and to the MAS agents, where end-to-end latency is reconstructed and decomposed into segment-specific contributions. This enables accurate localization of degradation events across the RAN, SDN, and edge domains.

5. Service deployment and monitoring closed loops

This section describes the operational procedures used to deploy the distributed AR service and the hierarchical monitoring loops that supervise its execution. Unlike Section 7, which reports experimental measurements, the present section focuses exclusively on the functional behavior of the control and assurance mechanisms implemented in DESIRE6G. The system behavior is described through workflow-based representations, which provide an explicit and reproducible description of the interactions among distributed components, better capturing the asynchronous and cross-domain nature of the proposed architecture than traditional pseudo-code formulations.

The platform operates through two main phases: (i) service deployment and secure MAS runtime verification, and (ii) continuous monitoring with multi-layer closed-loop control. In this work we describe all these functional procedures, referred to as Workflows (W). Fig. 3 shows the different workflows along with the involved layers and components of the architecture. Five workflows are presented hereafter: deployment (W1) before service operation, MAS runtime verification (W2) during service operation lifetime, and three latency assurance closed-loops upon different critical events affecting the end-to-end latency stability during service lifetime (W3–W5).

5.1. Service deployment procedure (W1)

Fig. 4 illustrates the sequence of interactions involved in the service deployment procedure across the tenant, the Service Management and Orchestration (SMO) layer, the SDN controller, and the DESIRE6G sites. The workflow starts when the tenant submits a deployment request to the Service Orchestrator (SO) (Step 1). Upon receiving the request, the SO retrieves the service description from the Service Catalog (SC) (Step 2) and obtains the current network topology from the Topology module (Step 3). Using these inputs, the SO constructs the global service graph representing both application components and required network functions.

The OE is then invoked to execute decomposition of the sites, placement and resource allocation algorithms (Step 4). During this decomposition phase, highlighted in Fig. 4, the global service graph is partitioned into site-specific sub-graphs according to resource availability, policy constraints, and performance requirements. Once the decomposition process is completed (Step 5), the SO holds all the descriptors necessary for instantiating the service across the involved DESIRE6G sites.

Following decomposition, the SO requests site-to-site transport connectivity from the SDN controller (Step 6). The controller configures the programmable data plane by installing forwarding rules across the SDN infrastructure. After connectivity is successfully established, the controller confirms that the transport layer is ready (Step 7), allowing the deployment process to proceed.

The SO then distributes the site-specific sub-graphs to the respective DESIRE6G sites (Step 8, 10). Each site receives its portion of the service graph, including network functions and application components to be instantiated locally. The IML at each site deploys the sub-graph, activating modules such as the UE2SM, NFR, in-band telemetry components, and the AR application containers. Once deployment is completed at

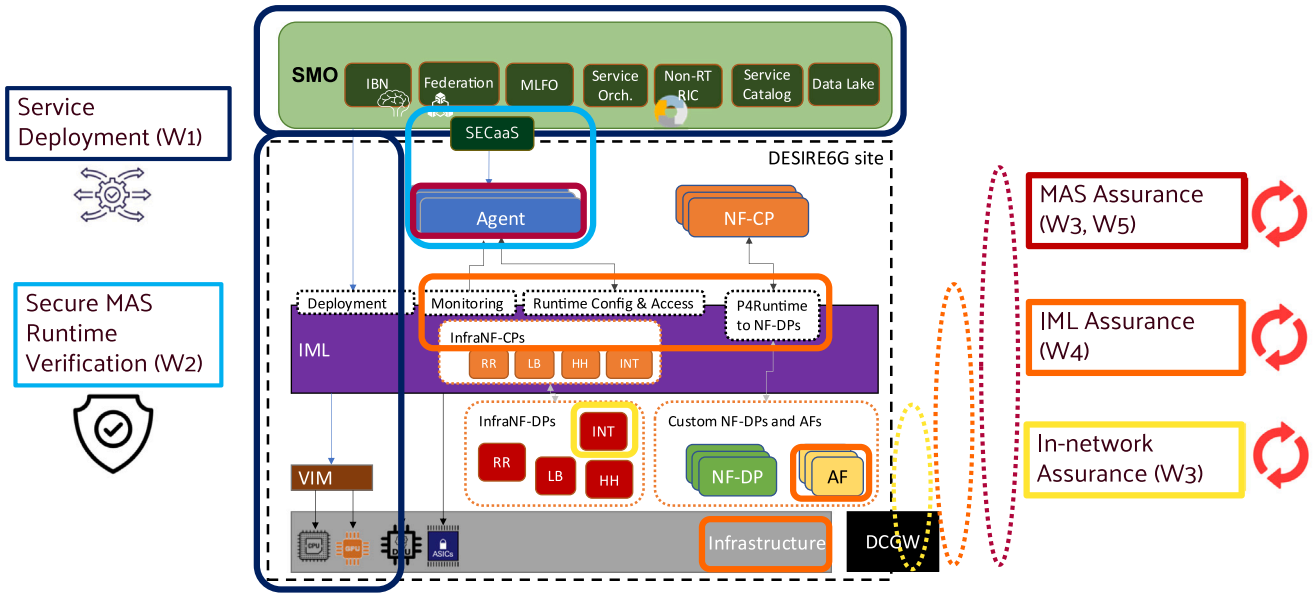


Fig. 3. Operational procedures (workflows) for establishing, monitoring and recovering latency-critical services.

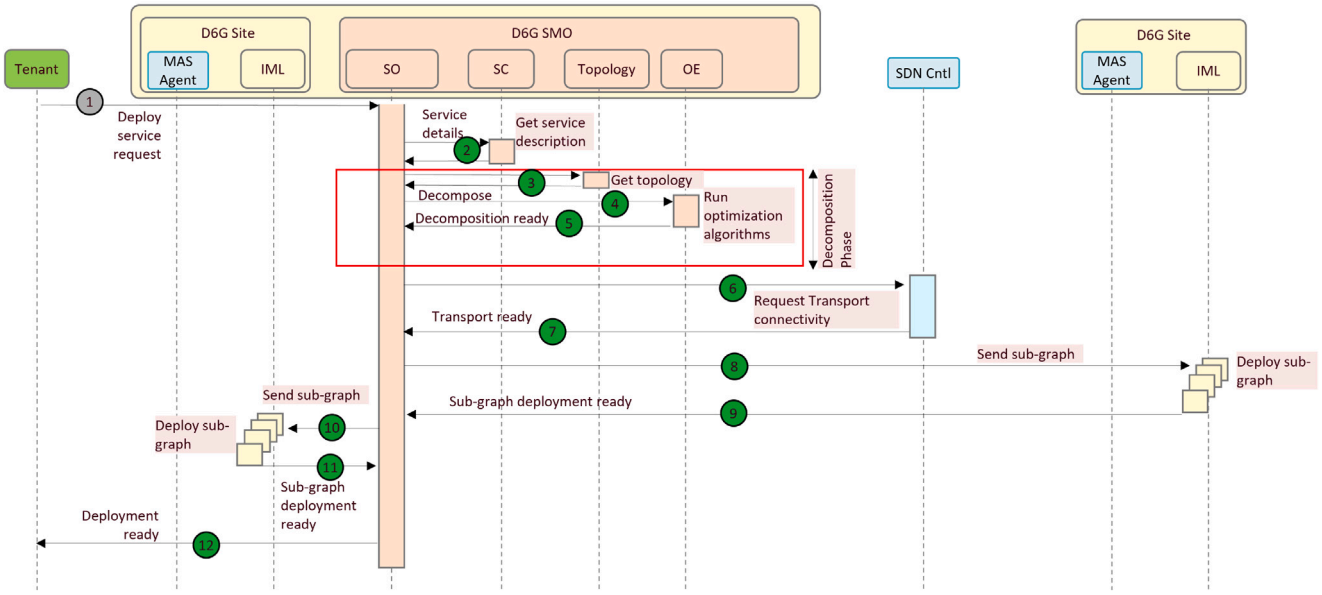


Fig. 4. Service deployment workflow (W1).

the site level, confirmation messages are sent back to the SO (Steps 9 and 11).

When all sites report successful instantiation, the SO marks the overall deployment as completed (Step 12). At this stage, the end-to-end service chain, from the drone user equipment through the RAN and SDN transport network to the edge cluster, is fully operational and ready to enter steady-state monitoring and closed-loop control.

5.2. Secure MAS runtime integrity verification (W2)

MAS agents are deployed in a per-service fashion, in order to preserve sensitive information and confidentiality. Fig. 5 illustrates the runtime integrity verification procedure of the MAS. Unlike the onboarding attestation performed during deployment, this workflow enables continuous, non-blocking integrity verification while agents are actively running.

After the service has started, each MAS agent operates normally within its DESIRE6G site. At runtime, an asynchronous attestation request may be triggered (Step 1). This request does not interrupt the execution of the agent, instead it initiates a background verification process handled by the associated sidecar component.

Upon activation, the sidecar computes a fresh integrity measurement of the agent's executable code by inspecting the relevant memory region and generating a cryptographic signature. This measurement is sent to the Distributed Ledger Technology (DLT) infrastructure, which acts as a trusted verification backbone. Within the DLT domain, a verifier node—representing the current root of trust—is selected to validate the received measurement (Step 2). The verifier retrieves the trusted reference signature stored during the wrapping phase and compares it against the newly generated runtime measurement.

If the two measurements match, the agent is confirmed as trustworthy and classified as a verified agent (Step 3). The verification outcome is recorded on the ledger, ensuring traceability and non-repudiation of

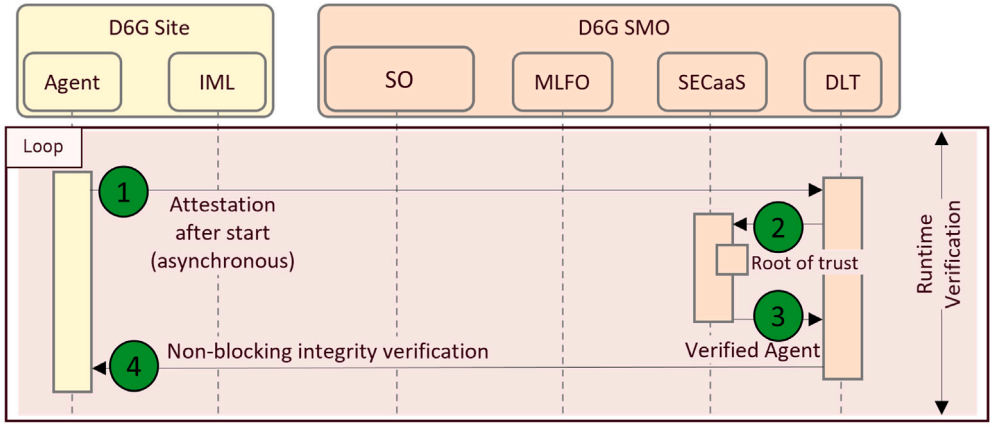


Fig. 5. Secure MAS integrity verification workflow.

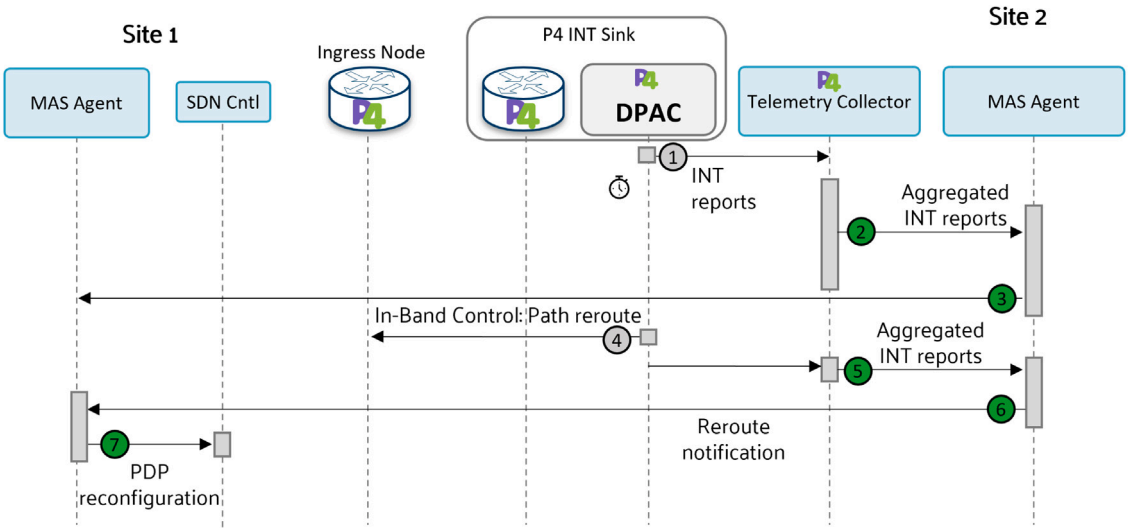


Fig. 6. Combined In-band control and MAS assurance workflow.

the integrity check. In the event of mismatch, the system would detect a potential tampering attempt and could trigger remediation policies at the orchestration layer.

A key characteristic of this workflow is its non-blocking behavior (Step 4). Runtime verification is performed asynchronously, allowing the MAS agent to continue its monitoring and decision-making tasks without service interruption. This design guarantees continuous trust assurance while preserving the responsiveness required by the hierarchical monitoring loops.

5.3. Combined data-plane and MAS service assurance closed loop (W3)

Fig. 6 illustrates the INC workflow responsible for ultra-fast mitigation of transport-layer latency degradation within the SDN domain. This mechanism operates entirely in the programmable data plane and represents the fastest closed loop in the DESIRE6G hierarchy, at the microsecond time granularity.

During normal operation, application packets carry INT metadata that are updated hop-by-hop across the P4-programmable switches. When additional traffic induces congestion in one of the SDN nodes, increased queue occupancy and per-hop delay are immediately reflected in the telemetry fields. The affected switch exports the latency anomaly to a DPAC, which aggregates local performance indicators and send data to a standard Telemetry Collector to aggregate latency information for MAS processing (Step 1,2,3).

The DPAC analyzes the collected metadata immediately and detects that the observed delay exceeds the expected threshold. This detection event is internally processed within the programmable data plane, without requiring MAS or orchestration-layer intervention. Upon confirmation of the degradation, the DPAC computes an alternative forwarding decision to a pre-planned path (Step 4) that bypasses the congested path segment.

The updated forwarding state is then installed directly in the involved P4 switches, effectively steering traffic toward a backup path. This rerouting action restores the required latency performance without service interruption. In parallel, aggregated latency information are distributed to the MAS to maintain consistency between data-plane state and controller-level topology awareness (Step 5, 6).

Finally, the MAS agent detects the reconfiguration event and may instruct performing a global re-optimization (Step 7), taking into account not only latency requirements, but also available bandwidth. However, it is important to emphasize that the first corrective action itself is executed entirely in-network, without waiting for control-plane coordination. By embedding detection and mitigation capabilities directly into the programmable data plane, the system prevents transient congestion peak events from propagating to higher assurance layers, preserving end-to-end latency stability for the AR service.

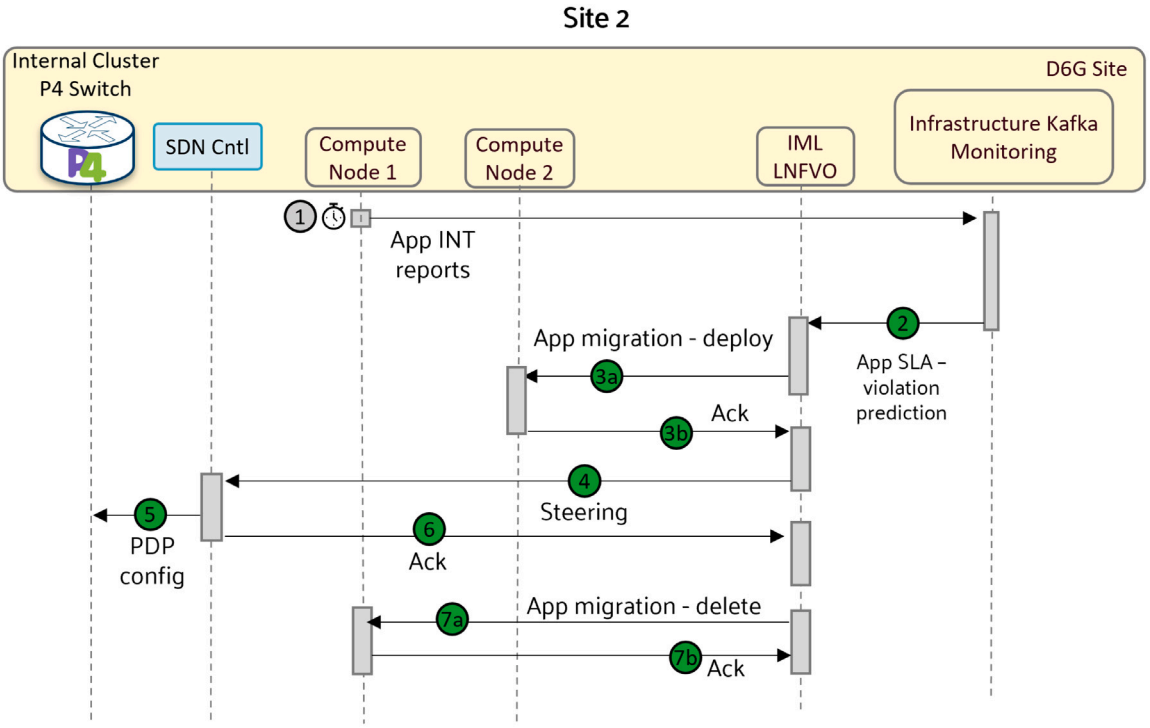


Fig. 7. IML-driven forecast and migration/scaling workflow.

5.4. IML-driven service assurance closed loop (W4)

Fig. 7 depicts the infrastructure-level closed loop responsible for handling compute resource degradation within an edge cluster, inside a single site. Unlike Workflow 3, which operates entirely in the programmable data plane, this loop supervises application containers and cluster resources through the IML. In fact, latency degradations are not always due to the network. In cloud–edge environments, the lack of available computing resources due to concurrent contention of CPU, GPU, memory and storage can compromise the runtime execution of tasks impacting the response time of application functions, especially when computation-intensive ML algorithms are executed.

During steady-state operation, the AR application pod executes on a selected compute node. Monitoring metrics, including CPU utilization and container resource statistics, are continuously collected by a Pervasive Kafka Monitoring service (Step 1). When a CPU load increase is induced on the node currently serving traffic, the monitoring system processes the collected metrics and feeds them to forecasting engines. Based on the observed trend, the forecasting component predicts a potential violation of the application Service Level Agreement (SLA) (Step 2). This prediction triggers an alarm notification to the IML, which is responsible for initiating recovery actions within a site.

Upon receiving the alarm, the IML (implemented as a Local NFV Orchestrator) starts the recovery procedure (Step 3). Two recovery strategies are supported: application migration or resource scaling.

In the migration case, the IML follows a make-before-break approach. A replica pod is first instantiated on an alternative compute node within the cluster (Step 3a). After successful deployment is acknowledged (Step 3b), the IML updates the traffic forwarding configuration so that application flows are redirected toward the new pod instance (Step 4). Although Fig. 7 depicts an interaction with an SDN controller for generality (Step 5), in the actual implementation the IML directly programs the internal P4 switch using its deployed service graph, effectively collapsing the steering and controller interaction into a single operation. Once traffic is successfully redirected and service continuity is confirmed (Step 6), the obsolete pod instance is deleted (Steps 7a and 7b).

In the scaling case, two pod instances (primary and backup) are pre-instantiated in the cluster. When the CPU overload alarm is raised, the IML skips the instantiation and deletion phases and directly performs traffic steering from the overloaded primary instance to the backup one (Steps 4–6), thereby reducing recovery time.

5.5. MAS-driven cross-domain service assurance (W5)

Fig. 8 illustrates the cross-domain service assurance workflow coordinated by the Multi-Agent System (MAS). This loop addresses latency anomalies that cannot be mitigated solely at the data-plane or infrastructure level, particularly when impairments originate within the RAN segment or additional alternative paths are available in the different network segments. It therefore represents the highest-level closed loop in the DESIRE6G control hierarchy, excluding the SMO.

During steady-state operation, application packets carry INT metadata from the user equipment to the edge, crossing the RAN and the SDN/edge segment. When a latency degradation is introduced in the RAN segment (e.g., in the midhaul path between DU and CU), the programmable nodes update the telemetry fields accordingly (Step 1). This metadata is exported to the Telemetry Collector, which aggregates per-flow latency contributions and reconstructs segment-specific delay components (Step 2).

The processed telemetry reports are forwarded to the MAS agent (Step 3), which performs anomaly detection and root-cause localization. By correlating end-to-end latency, RAN latency, and SDN latency contributions, the MAS determines whether the degradation originates in the transport network or in the RAN domain. This cross-layer visibility enables accurate problem localization even when the RAN internal control plane does not explicitly signal the impairment. Based on the anomaly localization and the overall multi-segment network state, the MAS suggests the most proper countermeasure to recover the end-to-end latency of the service.

If the anomaly is localized within the SDN domain, or SDN domain offers alternative low-latency paths, the MAS communicates with the SDN controller to request path reconfiguration (Step 5a). The controller

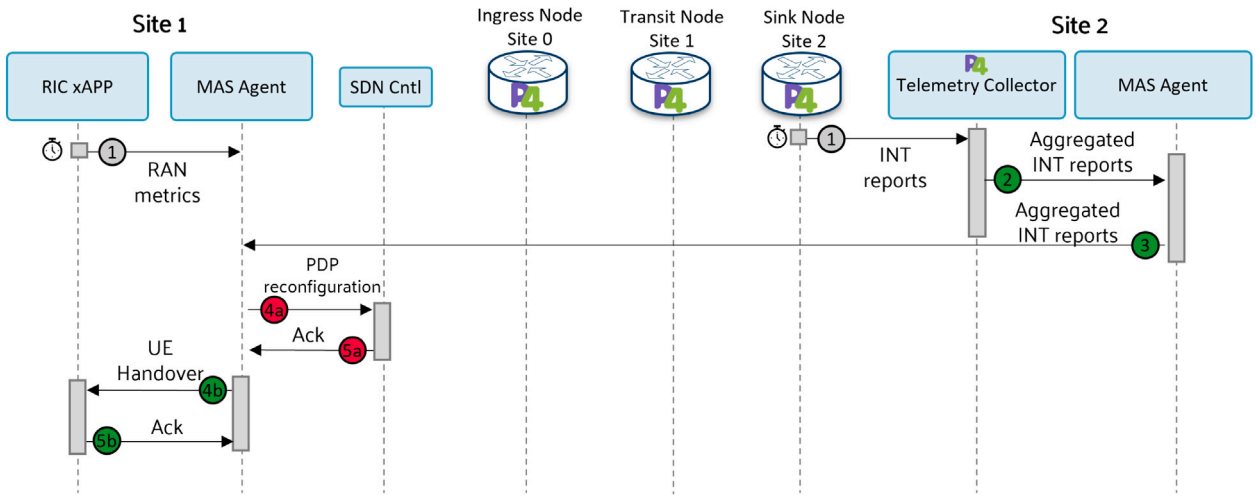


Fig. 8. MAS-driven multi-site service assurance workflow.

computes an alternative route and updates the P4 forwarding tables accordingly. Upon successful reconfiguration, a confirmation is sent back to the MAS to validate service recovery.

If instead the degradation is localized in the RAN segment, or SDN is unable to offer alternative low-latency paths, the MAS interacts with the RAN Intelligent Controller (RIC) through its xApp interface (Step 4a). The RIC may trigger corrective actions such as handover from one DU–RU pair to another (Step 4b), effectively bypassing the affected RAN path. Once the reconfiguration is completed, feedback is provided to the MAS to confirm restoration of normal performance (Step 5b).

This workflow operates at sub-second timescales and provides cross-domain coordination across RAN, transport, and edge domains. Unlike Workflow 3, which performs purely in-network mitigation, and Workflow 4, which addresses compute-level degradation, Workflow 5 integrates telemetry-driven analytics with control-plane orchestration. By correlating distributed measurements and coordinating corrective actions across multiple domains, the MAS-driven loop ensures sustained end-to-end latency compliance for the AR service even in the presence of RAN-specific impairments.

6. Demonstrator testbed setup

The DESIRE6G demonstrator has been deployed on the ARNO integrated testbed in Pisa, Italy. The setup integrates 5G O-RAN access, programmable SDN transport, Kubernetes-based edge computing, and distributed orchestration and monitoring components. Table 3 summarizes the main hardware, software, and configuration parameters of the experimental testbed, which are detailed in the next subsections.

From an architectural perspective, the demonstrator is organized into three logical DESIRE6G sites, as shown in Fig. 9: (i) the Drone Site (Site 0), (ii) the RAN Site (Site 1), and (iii) the Edge/SDN Site (Site 2). Each site hosts specific DESIRE6G components and participates in service deployment and monitoring according to the hierarchical model described in Section 5.

6.1. Drone site (UE domain)

The Drone Site (Site 0) and its components are represented in Fig. 10. It represents the UE domain and acts as the ingress point of the DESIRE6G service chain in the uplink direction. It includes the quadcopter platform equipped with a Pixhawk 6X flight controller and two companion computing boards interconnected via Gigabit Ethernet.

Fig. 11 shows the details of the drone platform assembled and implemented for the DESIRE6G demonstrator. The first board is an NVIDIA Jetson Orin NX hosting the AR1 module, responsible for real-time object detection and lightweight inference. The second board is an Intel NUC i7-1185G7 hosting the DESIRE6G user-side network functions. Application packets generated by AR1 are encapsulated with DESIRE6G headers and telemetry metadata before transmission over the 5G link. A Quectel GL-500 5G modem connected via USB 3.0 to the NUC board acts as the physical User Equipment and provides wireless connectivity in the n78 band (3.3–3.8 GHz TDD).

The Drone Site therefore acts as the application data source, the first telemetry insertion point and the initial DESIRE6G service entry node.

6.2. RAN site (5G O-RAN domain)

The RAN Site (Site 1) implements the radio access segment and forms an intermediate DESIRE6G domain between the UE and the programmable transport network. It consists of two Benetel SC-705e Open RAN radio units connected via 10 GbE eCPRI fiber to virtual distributed units (DU1 and DU2) running on the Accelleran dRAX platform. The centralized unit (CU) and 5G Core Network are deployed as virtual machine instances on a Dell PowerEdge R760 server equipped with Intel Xeon Gold processors, up to 256 GB RAM, and high-speed network interfaces. The RAN operates using O-RAN Split 7.2 with 50 MHz channel bandwidth and DSUUU 5 ms TDD configuration. From the DESIRE6G perspective, the RAN Site hosts the RAN-aware telemetry collection, the user-plane traffic steering toward the SDN domain, the MAS agents capable of interacting with the RIC, the interfaces for cross-domain service assurance. A programmable Tofino switch connects the RAN Site to the SDN transport fabric. This switch performs NFR, telemetry update, and latency measurement, acting as the RAN-to-transport boundary of the DESIRE6G service chain.

6.3. Edge/SDN site (Transport and edge computing domain)

The Edge/SDN Site (Site 2) combines the programmable transport fabric and the edge computing cluster, forming the final DESIRE6G domain where AR2 processing and orchestration components reside. The transport infrastructure is built using Intel Tofino-based P4-programmable switch (APS BF2556X), featuring a 6.5 Tb/s switching ASIC and 40/100 GbE interconnections. This node implements NFR and INT. The switch is connected to an SDN network composed of BMv2 P4 switches implementing in-band control (INC and DPAC), not shown in the figure. The edge cluster consists of four NVIDIA Jetson AGX Orin 64 GB nodes and Dell PowerEdge R7515 servers. The nodes run a Kubernetes environment hosting the AR2 application pods and

Table 3
Experimental setup and testbed parameters.

Parameter	Value/Description
Topology	3 sites (Drone/UE, RAN, Edge/SDN)
Application	AR pipeline (AR1 drone video+Yolo FaaS, AR2 edge Posenet FaaS, AR3 XR Unreal Engine 5 rendering)
RAN hardware	2x Benetel SC-705e O-RAN RUs + Accelleran vDU server (AMD Ryzen 7950X, 128GB RAM)
RAN configuration	Split 7.2; dRAX v13.1.2; CU v7.0.6; DU v4.1.0; 50 MHz channel; TDD DSUUU (5 ms); Tx power 24 dBm
RIC/Core server	Dell PowerEdge R760 (Intel Xeon Gold 6438Y+, GPU NVIDIA A16)
SDN infrastructure	2x Dell R7515 (AMD EPYC 7262), P4-programmable network (NIKSS, BMv2)
Programmable switch	Intel Tofino BF2556X (6.5 Tb/s, <400 ns latency, P4Runtime)
Edge computing	4x NVIDIA Jetson AGX Orin (64GB, Ampere GPU, 10GbE)
MAS platform	Dell PowerEdge R760 (Intel Xeon Gold, 256GB RAM)
Control/SMO	Dell PowerEdge R760 (Intel Xeon, GPU A16)
Monitoring	INT (P4) + Kafka-based telemetry pipeline
Drone platform	Jetson NX + x86 companion (Intel i7-1185G7), PX4 flight controller
Drone 5G User Equipment	Quectel GL-500, n78 band
XR device	Meta Quest 3 (Snapdragon XR2 Gen2, 120 Hz display)
Uplink traffic	Video stream (5–20 Mbps, H264/MJPEG)
Downlink traffic	Control + AR metadata streams
Deployment platform	K8S
E2E latency target	25 ms
Traffic conditions	Steady state + controlled perturbations
Perturbations	SDN congestion, RAN delay injection, edge CPU load

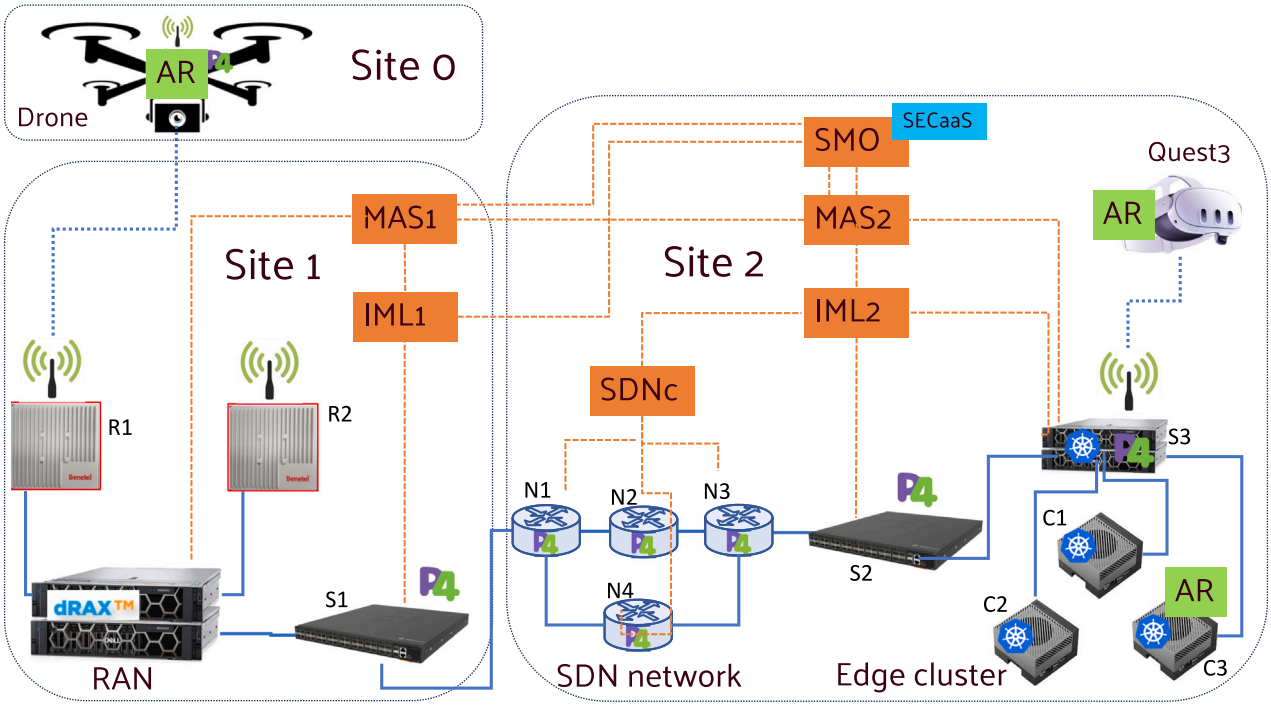


Fig. 9. Demonstrator testbed setup: sites and equipment.

the local IML-LNFVO. Internal cluster traffic is steered through a P4-programmable switch that performs DESIRE6G header decapsulation and forwards traffic to the selected application pod. Traffic steering commands are installed directly via P4Runtime programming by the local IML (see Fig. 12).

The Edge/SDN Site also hosts the Telemetry Collector aggregating INT data, the Kafka-based Infrastructure Monitoring, the forecasting engines for SLA prediction, the MAS agents for cross-domain coordination, the SMO components and the IML instance.

The Meta Quest 3 headset connects via Wi-Fi 6 to this site and runs the AR3 module for augmented rendering.

7. Experimental evaluation and results

This section presents experimental results collected on the ARNO testbed during the execution of the distributed AR service. The evaluation focuses on three key aspects: (i) RAN performance, (ii) end-to-end latency across DESIRE6G domains, and (iii) service assurance mechanisms under induced anomalies. Results confirm that DESIRE6G can meet stringent real-time performance requirements even under network stress and resource perturbations.

All experiments are conducted on the ARNO testbed under controlled conditions, where the system is first brought to a steady state before each measurement. For each workflow, measurements are obtained over multiple executions of the same experiment, ensuring consistent system configuration and workload conditions. Unless otherwise stated, reported results represent either (i) aggregated metrics collected over multiple runs or (ii) representative traces selected from stable executions, depending on the nature of the measurement. Timing measurements are obtained through instrumentation of system components (e.g., telemetry collectors, MAS agents, and control-plane APIs). While the experimental results are obtained from repeated executions, a full statistical characterization (e.g., confidence intervals) is limited by the constraints of the physical testbed, including resource availability

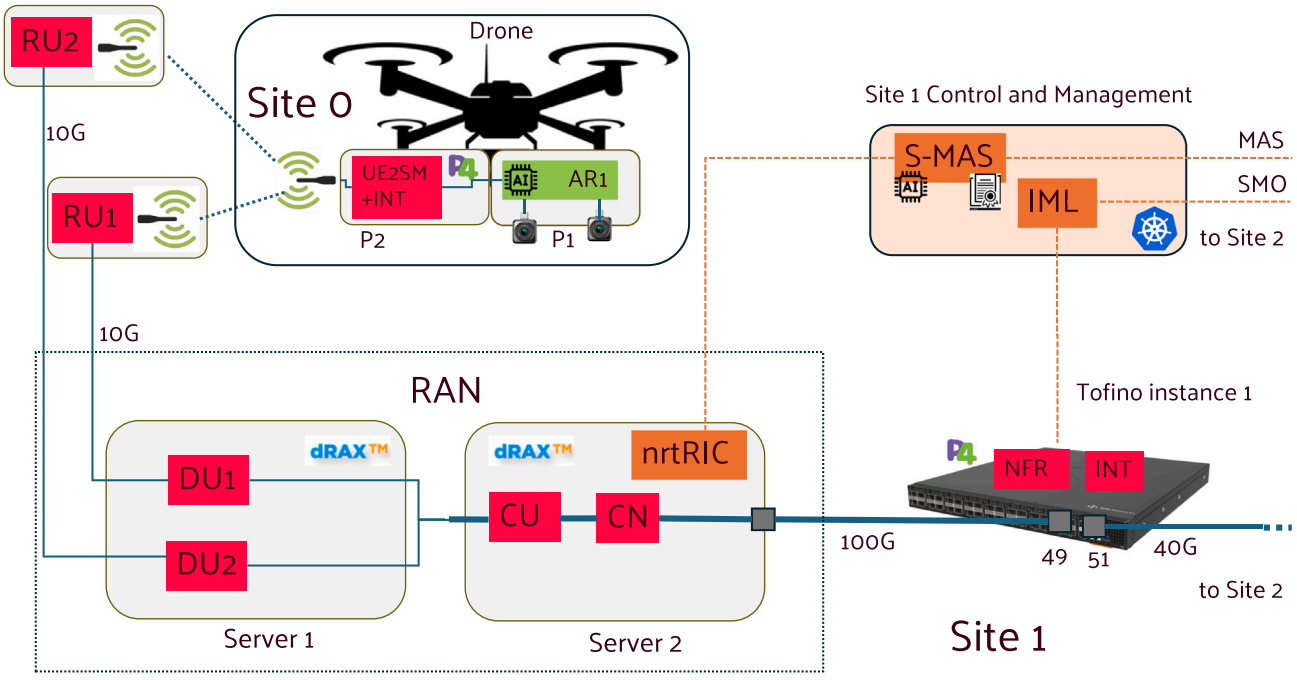


Fig. 10. Demonstrator testbed setup: Site 0 and Site 1.

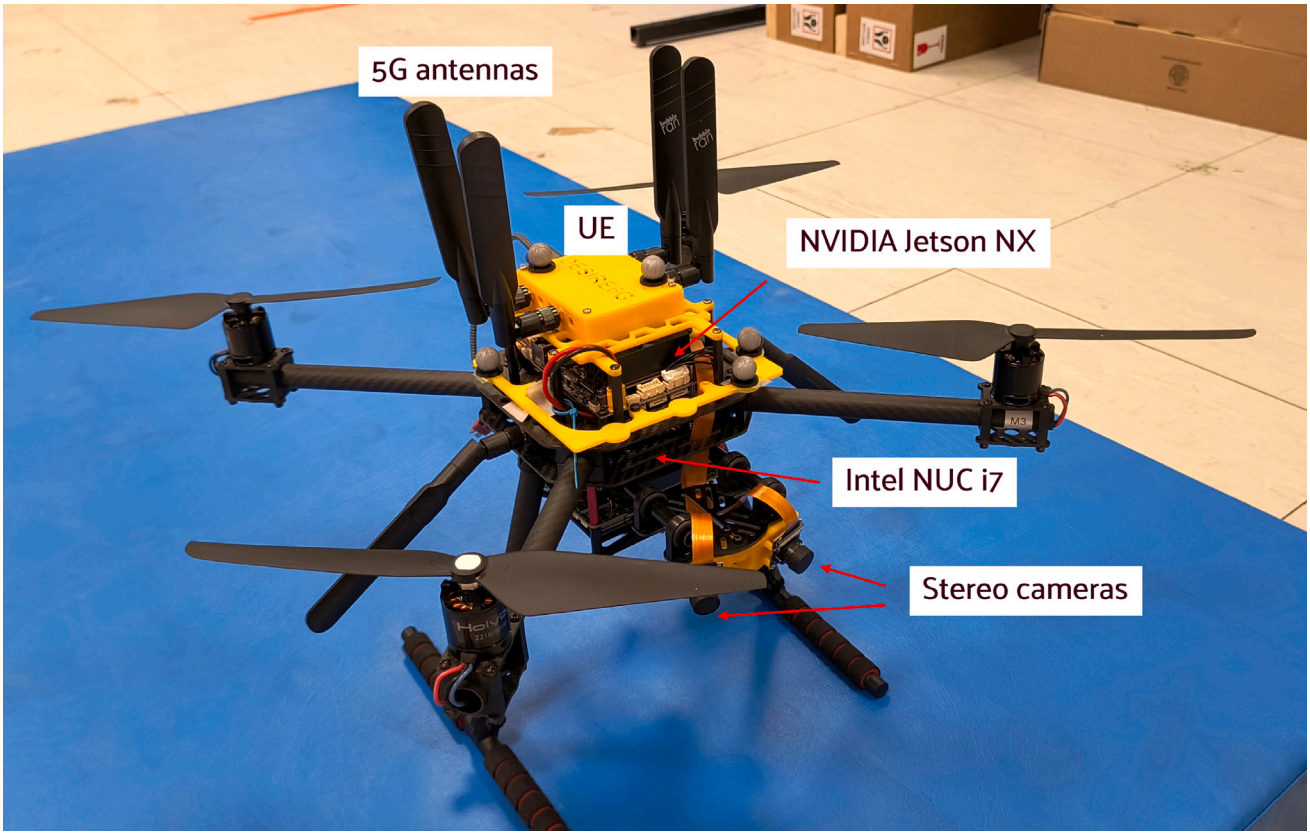


Fig. 11. Drone quadcopter with cameras, double board payload and User Equipment.

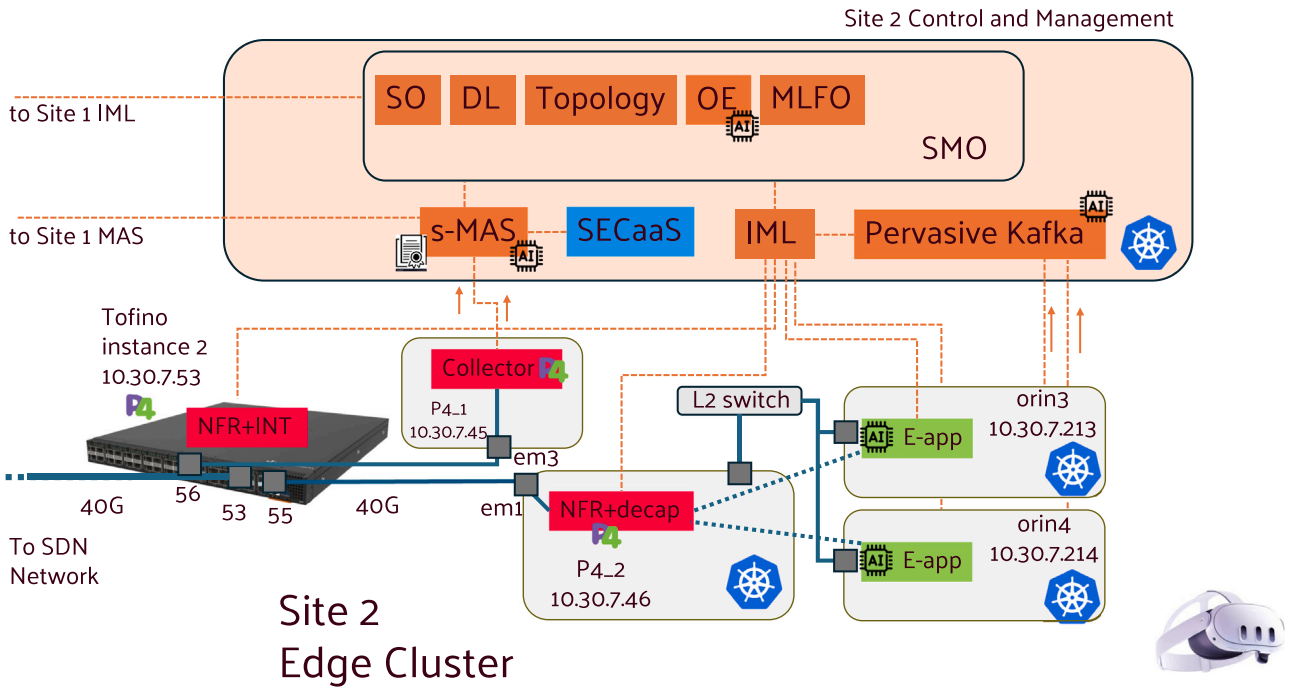


Fig. 12. Setup details of the edge cluster of Site 2.

and execution time. The focus of this work is therefore on system-level validation and relative performance trends rather than statistical benchmarking. Nevertheless, the consistency of measurements across runs confirms the reliability of the observed behaviors.

7.1. RAN performance

To characterize the 5G Open RAN segment, a 30-second `iperf3` UDP test was executed between the drone UE and the edge cluster. Fig. 13 reports downlink (DL) and uplink (UL) throughput, jitter, and packet loss observed at the UE.

Downlink throughput remains stable between 95–100 Mbit/s, with occasional dips associated to temporarily scheduling fluctuations. Uplink throughput varies between 35 and 65 Mbit/s, averaging approximately 50 Mbit/s. Average jitter remains below 0.2 ms throughout the test, demonstrating reliability for latency-critical mission traffic. Downlink packet loss averages 4%–5%, with isolated peaks up to 25%, typically associated with RLC/MAC retransmission bursts. Recovery is fast, confirming adaptive modulation and coding effectiveness. The measured AR application uplink traffic needs for the video stream are around 20MB/s as shown in Fig. 13 this demonstrates that the O-RAN deployment provides sufficient uplink and downlink capacity for real-time XR workloads, while the jitter and recovery behavior confirm the stability needed for low-latency services.

Although the measured RAN performance does not target URLLC-grade operation, it remains sufficiently stable for the distributed AR service considered in this work. In particular, the observed throughput and low average jitter provide adequate support for the uplink video stream and bidirectional control traffic, while the packet loss behavior is mainly characterized by transient peaks rather than persistent degradation. These events are consistent with short-lived scheduling fluctuations and RLC/MAC retransmission dynamics in the experimental deployment. From the application perspective, the measured latency regime is compatible with stable AR execution in the considered setup, even though it does not represent the ultra-reliable operating region required by stricter closed-loop wireless control applications.

7.2. End-to-end latency across DESIRE6G domains

End-to-end latency was measured between the drone endpoint and the edge processing node with all DESIRE6G network functions active, including UE2SM, UE-INT, SDN forwarding, and telemetry-exporting Tofino nodes. Fig. 14 shows the RTT distribution across thousands of samples.

Most RTT values lie in the 15–25 ms region, forming a narrow peak with low jitter. Occasional samples reach 35–40 ms, corresponding to short-lived queue buildup or retransmissions at the radio interface. The histogram underscores the stability of the complete multi-hop route, including wireless, midhaul, and edge processing.

In typical operation, the system remains below the 30 ms threshold required for real-time object detection, video overlay rendering, and closed-loop AR feedback. Even the right tail values remain under 50 ms, validating the feasibility of distributed XR pipelines over programmable RAN and SDN infrastructures.

7.3. Workflow 1: Service deployment evaluation

Workflow 1 evaluates the automated deployment of the AR service across the three DESIRE6G sites. The objective is to validate correct service graph decomposition, site-specific instantiation, programmable data-plane configuration, and quantify the time contribution of each architectural component.

Implementation

The AR service is defined as an end-to-end service spanning the three sites. The initial service is described through a YAML-based Network Service Descriptor (NSD) defining service metadata, application functions, forwarding graphs, and an end-to-end round-trip latency budget of 25 ms.

The descriptor includes the AR source application (pre-deployed at the UE), the RAN function (pre-deployed), and the AR edge application (scalable, two instances). Forwarding graphs explicitly define upstream (RAN → edge) and downstream (edge → RAN) connectivity.

The OE converts the YAML descriptor into an internal graph representation using NetworkX [30], enriching it with abstracted resource

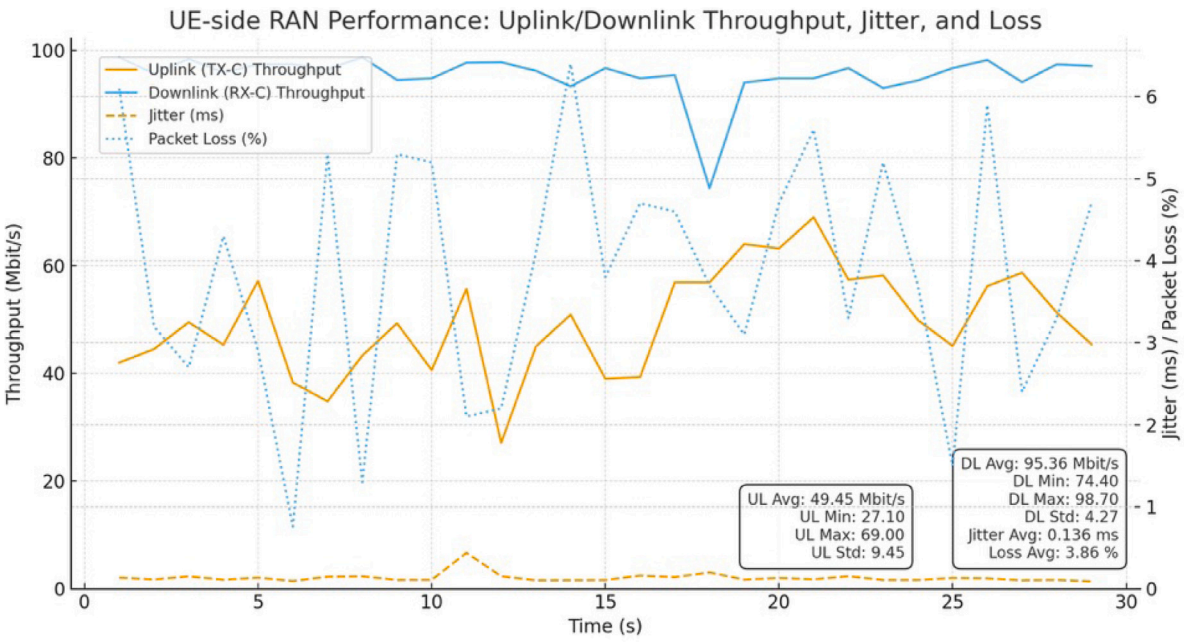


Fig. 13. RAN throughput, jitter, and packet loss measured via UDP `iperf3`.

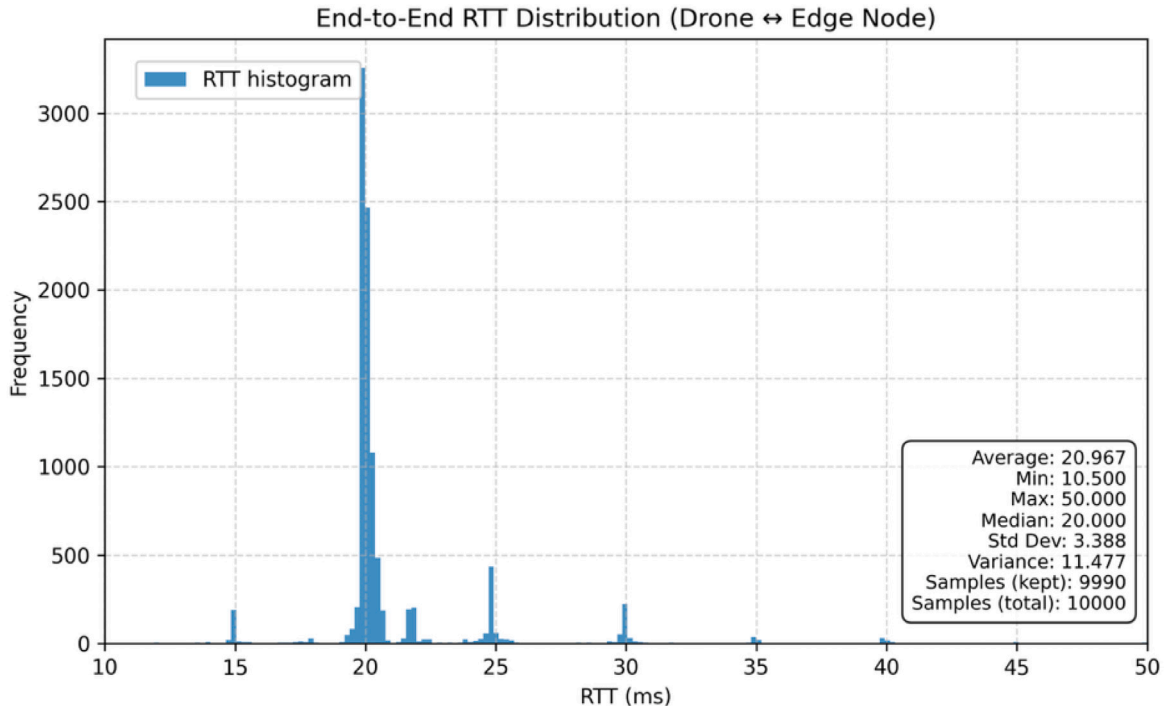


Fig. 14. End-to-end RTT distribution with DESIRE6G data plane fully enabled. Histogram bars indicate frequency, while the line represents polynomial regression.

availability and topology information. The optimization objective is to partition the service graph across sites while respecting latency constraints. Algorithm selection is assisted by a local LLM-based mechanism that chooses the most suitable partitioning strategy from an internal Algorithm Pool containing curated graph partitioning implementations [26]. The selected strategy performs latency-aware service partitioning. The resulting decomposed subgraphs are translated into site-specific Local Network Service Descriptors (LNSDs).

Two IML instances are involved: one controlling Site 1 (RAN) and one controlling Site 2 (Edge Cluster). Site 0 is assumed pre-deployed. At Site 1, the LNSD interconnects the RAN to Site 2. In upstream direction, INT is enabled; downstream telemetry is disabled. The IML parses the logical subgraph, determines the physical path, and configures NFR and INT functions via REST-based infraNF control-plane APIs. At Site 2, the LNSD includes the AR2 application. The IML selects the appropriate Kubernetes node, deploys the application pods, and instantiates software infraNFs (NFR and UE2SM). In upstream direction, DESIRE6G headers and INT metadata are removed before traffic reaches the application, and telemetry reports are forwarded to the site collector. In downstream direction, traffic is re-encapsulated via UE2SM and forwarded through NFR. The SDN controller, developed in Python, configures the SDN programmable switches N_x of Site 2 during deployment.

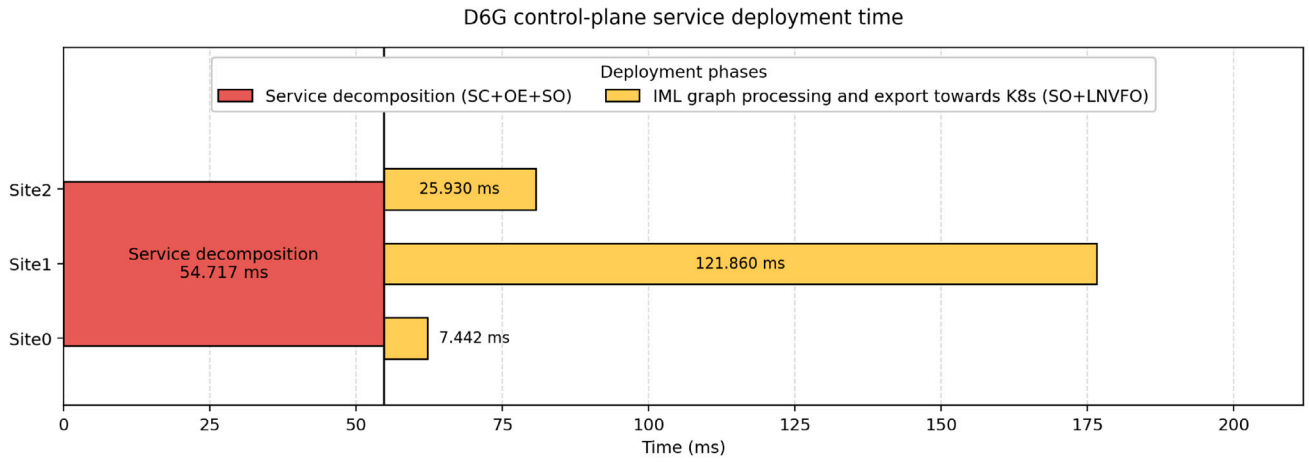


Fig. 15. Service deployment time breakdown: SMO modules.

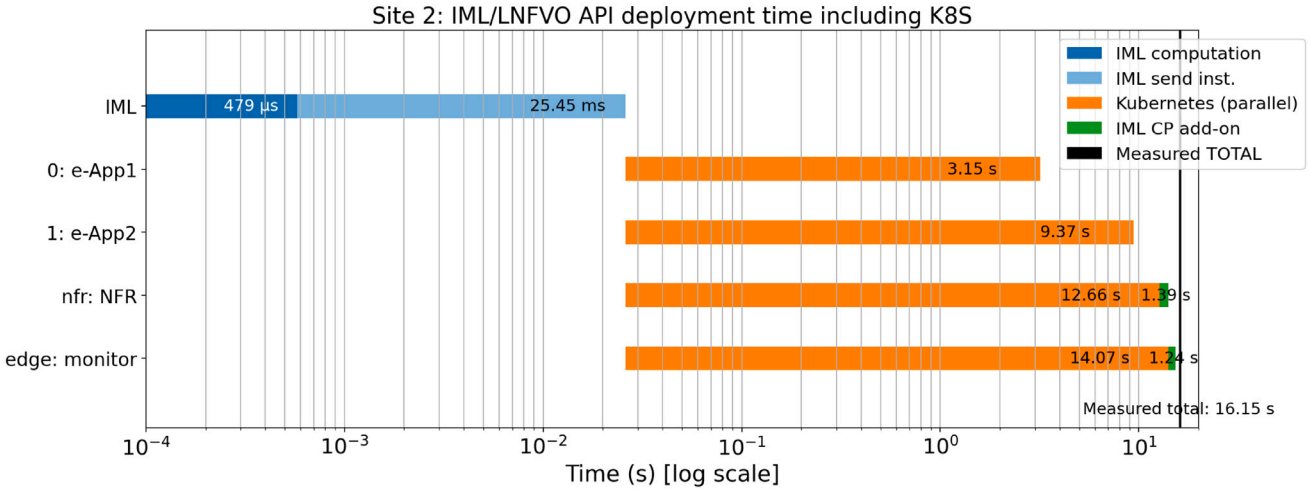


Fig. 16. Service deployment time breakdown: IML modules and function deployment.

It computes primary and backup paths and installs forwarding rules using a Layer-2.5 label-based scheme. A custom L2.5 header inserted at ingress enables INC deterministic routing and cumulative queuing delay monitoring, preparing the domain for in-band control in Workflow 3.

Tests and results

The tests validate full service deployment across the ARNO testbed and measures time contributions of SMO, OE, SC, IML, SDN controller, and Kubernetes components. Prior to execution, all SMO and IML subsystems were operational in steady state. A tenant request containing the full service graph and the 25 ms latency requirement was issued to the SO. Timestamps were collected at SMO and IML API levels to measure the processing time of each component. Fig. 15 presents the SMO-level processing breakdown. Service Catalog retrieval requires approximately 5 ms. The SO–OE interaction for decomposition takes 27 ms, including communication overhead, with the OE internal computation accounting for 13 ms. After optimization, the SO composes site-specific descriptors in 13 ms and retrieves IML endpoints in 9 ms. The total SMO contribution is approximately 54 ms. Including SDN controller interaction and Site 1 switch configuration, the total control-plane operation from the SMO perspective completes in approximately 175 ms. The Tofino rule enforcement contributes around 122 ms.

As discussed in Section 3, the IML operates with millisecond-scale decision latency, while the actual deployment and migration actions depend on the underlying orchestration platform and may occur at a longer timescale. Fig. 16 shows the IML-level measurements at Site 2. Upon receiving the LNSD, the IML parses the request in less than 500 µs and prepares pod configurations in approximately 25 ms. Kubernetes API calls then trigger parallel instantiation of four pods (two AR edge pods, one NFR instance, and one monitoring engine). Pod instantiation dominates the deployment time and requires several seconds depending on image loading and initialization. After pods are active, control-plane rules are computed and sent to infraNFs in less than 1.5 s. The total IML-driven deployment time at Site 2 is approximately 16 s.

These results highlight a clear separation between IML control-plane latency and infrastructure-level execution time. The IML decision and configuration phase accounts for less than 30 ms overall (including parsing and deployment preparation), corresponding to less than 0.2% of the total deployment time. In contrast, Kubernetes-based pod instantiation accounts for more than 90% of the total latency, with individual components requiring between ~3 s and ~14 s depending on initialization complexity. The subsequent control-plane configuration phase contributes approximately 1–1.5 s. This quantitative breakdown confirms that the IML itself introduces negligible overhead, while the dominant latency component is due to the underlying container orchestration platform. Such behavior is consistent with cloud-native systems, where fast control-plane reactions trigger comparatively slower execution processes in the infrastructure layer. It is worth noting that faster deployment times can be achieved by adopting alternative execution models, such as pure Function-as-a-Service (FaaS) frameworks (e.g., Knative), which enable pre-warmed instances and reduced startup latency.

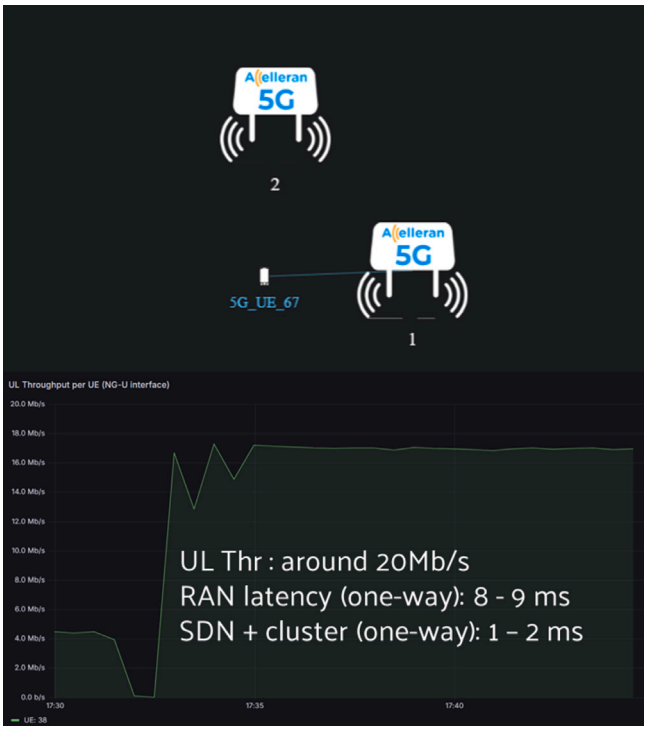


Fig. 17. Deployed AR application: performance.

7.3.1. Deployed service performance

The performance of the deployed AR application is evaluated under two streaming encoding configurations: Motion JPEG (1280 × 768) and H.264 (1280 × 720). Motion JPEG generates a stable uplink throughput of approximately 20 Mbit/s, while H.264 reduces bandwidth consumption to 5–12 Mbit/s depending on scene dynamics. Fig. 17 shows the Motion JPEG case measured at the RIC side, representing the higher bandwidth demand. In both configurations, the one-way network latency from drone to edge remains within 9–11 ms. The RAN segment contributes 8–9 ms, including the wireless link, while the SDN and edge cluster account for 1–2 ms. The Tofino switch latency is negligible (around 4 µs), with residual delay mainly introduced by BMv2 software switches and Kubernetes data plane processing.

During the flying mission, the drone follows a predefined MAVLink trajectory while the AR pipeline operates continuously. The different sub-app functions are shown in Fig. 18. The AR1 on the Jetson payload performs object detection and transmits annotated video to the edge AR2, where pose estimation and semantic processing are executed. The processed metadata are then forwarded to the Quest 3 headset, where 3D reconstruction and spatial alignment render a skeletal overlay on the passthrough view. This enables the operator to visualize detected persons even when partially occluded by obstacles, enhancing situational awareness beyond direct line of sight. The system sustains > 30 fps during flight. Application-stage processing adds approximately 8 ms, resulting in a total one-way end-to-end latency of about 26 ms. The dominant latency contributors are AI inference stages and the RAN segment, while the programmable data plane introduces minimal overhead.

7.3.2. SMO scalability analysis

To evaluate the scalability of the SMO, we conducted a stress test focusing on the OE, which represents the main computational component in the control plane. The OE was horizontally scaled using Docker-based replicas, with the number of instances varying from $N = \{1, 2, 4, 8, 16, 32\}$. For each configuration, a total of 2048 service instantiation requests were issued to the SO, while the internal timing of all components was recorded and aggregated over 500 execution samples. The evaluation considers both latency and throughput under increasing concurrency levels. Results are reported in Fig. 19.

Results show that, with a single OE instance, the average end-to-end SMO latency is approximately 562 ms. In this configuration, the dominant contribution is due to message queuing and OE processing overhead (around 500 ms), while the remaining SMO components (Service Catalogue, orchestration logic, and deployment triggering) contribute less than 70 ms. By scaling the OE horizontally, the system exhibits a significant reduction in latency. In particular, with 32 OE instances, the average SMO latency decreases to approximately 217 ms, corresponding to a 2.6× improvement compared to the single-instance baseline. This reduction is primarily due to the decrease in queuing and processing overhead, which drops to around 140–150 ms. In terms of throughput, the system scales efficiently with the number of OE instances. When more than 16 workers are deployed and sufficient request parallelism is maintained, the SMO achieves a peak throughput of approximately 138 requests per second. These results confirm that the SMO architecture can effectively scale through horizontal replication of its computational components, supporting high request rates while maintaining bounded control-plane latency. This analysis demonstrates that the SMO does not represent a scalability bottleneck in the proposed architecture, and that its performance can be tuned according to the workload by dynamically adjusting the number of OE instances.

7.4. Workflow 2: Secure MAS verification

The MAS runtime integrity mechanism is implemented through the integration of the SECaaS wrapper, D-MUTRA sidecars, and a blockchain-based verifier network, as introduced in the workflow description 5.2.

Prior to deployment, each MAS agent is wrapped using SECaaS. During this phase, the executable ELF binary is analyzed and a reference SHA-256 signature is generated from its .text section. The wrapper reproduces the same aggregation logic used at runtime to ensure deterministic comparison. Reference signatures are stored in an off-chain PostgreSQL database and cryptographically anchored to the Distributed Ledger Technology (DLT) infrastructure, forming the trust baseline.

At runtime, each agent is paired with a D-MUTRA sidecar container implemented in Go. The sidecar periodically extracts the agent's executable memory segment directly from /proc/<pid>/mem, computes an SHA-256 digest, and aggregates consecutive identical measurements (300 by default) before producing a combined attestation hash (agent + sidecar self-integrity). CPU consumption is constrained to 10% via Linux cgroup v2 to guarantee non-intrusive operation. The resulting evidence is submitted to the blockchain network.

The DLT infrastructure consists of a 10-node Hyperledger Besu Proof-of-Authority (PoA) network, ensuring deterministic finality and low-latency transaction confirmation. The attestation smart contract governs evidence submission, manages verifier election, and records verification outcomes. Upon submission, an elected verifier retrieves the trusted reference signature and recomputes the hash to validate runtime integrity.

For performance monitoring, timing probes are collected locally by each sidecar and temporarily stored in Redis. Aggregated metrics are periodically exported through Prometheus Pushgateway to a centralized Prometheus instance, enabling systematic evaluation of attestation latency and system overhead.

Tests and results

Two prover MAS agents were evaluated: a multiflow agent and a telemetry agent. Each agent performs local measurement extraction and submits attestation evidence to the blockchain.

The experiment started with the deployment of a 10-node Hyperledger Besu Proof-of-Authority (PoA) blockchain network and initialization of the attestation smart contract. MAS agents were registered

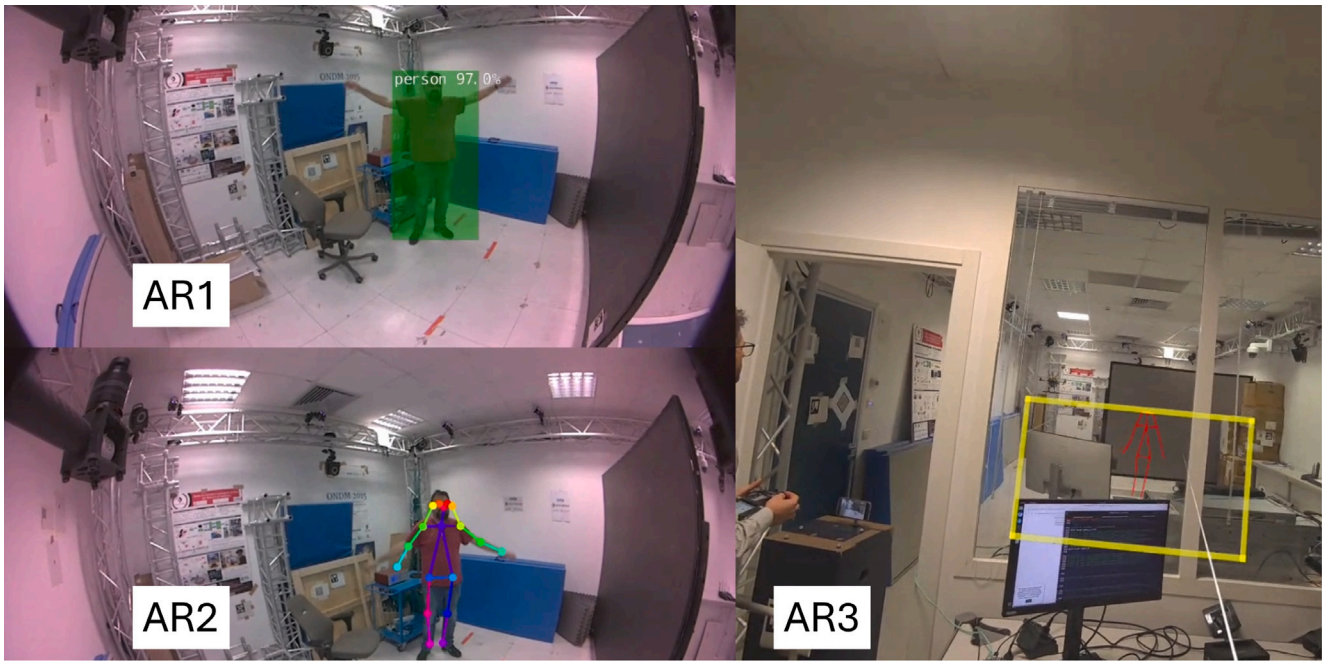


Fig. 18. Augmented reality functions: AR1 at the drone, AR2 at the edge, AR3 on the Quest headset.

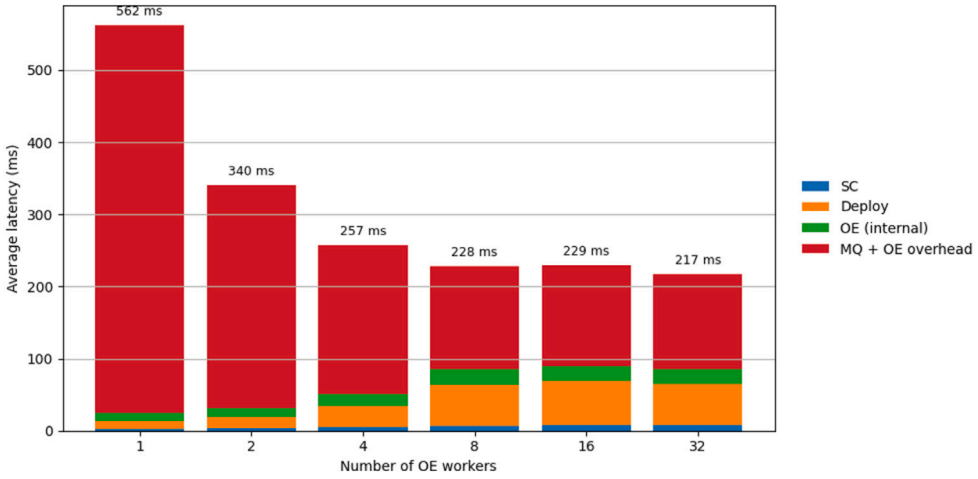


Fig. 19. SMO scalability analysis results.

through the SECaaS wrapper, which generated and stored their reference signatures. The agents and their associated sidecars were then launched, enabling continuous runtime attestation. A total of 100 attestation cycles per agent were executed, while timing and resource usage metrics were collected and exported to Prometheus for analysis.

Table 4 summarizes the obtained measurements. For the multiflow agent, the end-to-end attestation cycle duration is 4.0613 s. Local measurement time (hashing of the code section) is approximately 1.11 ms. The DLT commit phase requires 7.8 ms, while the verifier recomputation and validation require 2.5 ms. For the telemetry agent, the end-to-end cycle duration is 4.0886 s. Measurement time remains approximately 1.1 ms. DLT commit time is 12.5 ms, and verification requires 2.0 ms. The CPU overhead introduced by the protected agents themselves is negligible (0.00–0.01%). The sidecar containers introduce 6.73% CPU usage for the multiflow agent and 8.94% for the telemetry agent, reflecting hashing operations, blockchain interaction, and protocol management.

The results demonstrate that runtime code integrity verification can be performed with negligible impact on protected MAS agents.

Table 4
Workflow 2 – Runtime Attestation Performance Results.

Category	Metric	Multiflow	Telemetry
Attestation	End-to-End Time (s)	4.0613	4.0886
	Measurement (s)	0.001110	0.001106
	DLT Commit (s)	0.0078	0.0125
	Verification (s)	0.0025	0.0020
Overhead	Agent CPU (%)	0.01	0.00
	Sidecar CPU (%)	6.73	8.94

The hashing and verification phases operate in the millisecond range, confirming the computational efficiency of the integrity measurement mechanism. The dominant component of the 4 s end-to-end attestation cycle is not local computation but asynchronous DLT propagation and network latency. The blockchain commit and verification phases themselves remain below 15 ms. Importantly, these results validate the "Attest After Starting" model adopted in DESIRE6G. A traditional

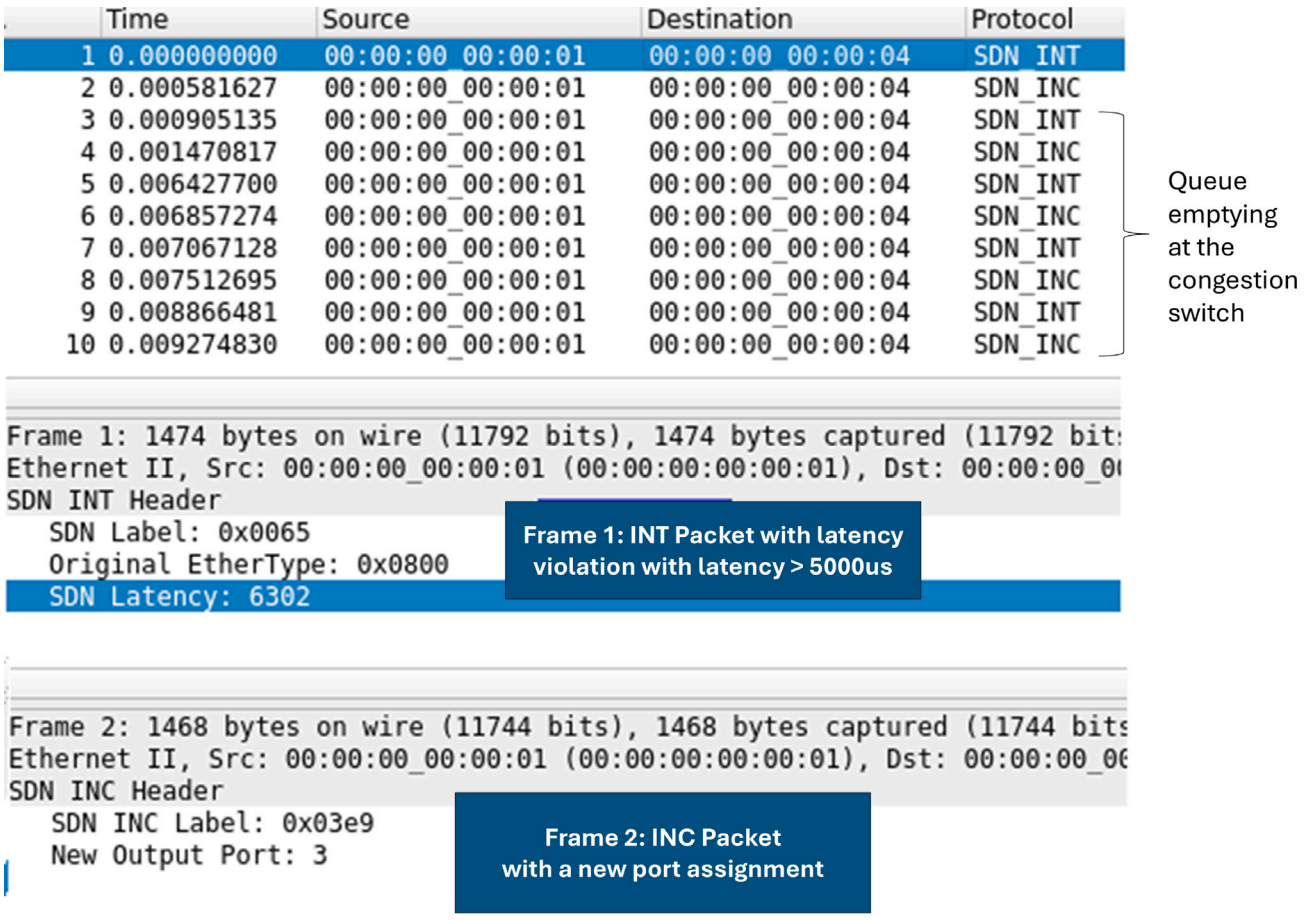


Fig. 20. a packet capture showing both INT and INC packets.

startup-onboarding attestation approach would introduce several seconds of delay before agent activation. Instead, runtime attestation allows MAS agents to start immediately while integrity is continuously verified in the background.

7.5. Workflow 3: In-band control reaction

Workflow 3 closed loop evaluates the hierarchical service assurance mechanism combining (i) ultra-fast in-band control executed directly in the programmable data plane within the SDN transport domain, and (ii) a slower MAS-driven re-optimization loop. The MAS intervention is similar to W5 and all MAS results will be reported in Section. Here, the goal is to validate that traffic bursts that cause congestion on the primary SDN path are first mitigated by immediate in-network rerouting (INC).

In-band network control implementation

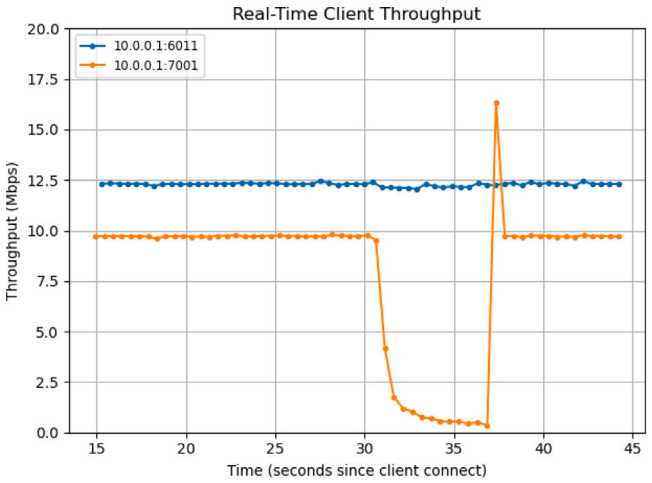
The In-band Network Control (INC) mechanism is implemented in the P4 BMv2 switches and operates within the SDN transport domain attached to the edge cluster (i.e., nodes Nx in Fig. 9) using an L2.5 label-based forwarding scheme. Two headers are used: (i) an SDN INT header for routing and cumulative queuing delay monitoring, and (ii) an INC control header for in-domain rerouting, both headers are shown in Fig. 20. The SDN INT header is inserted by the ingress SDN node immediately after the Ethernet header. It includes a globally unique 16-bit `sdn_label` assigned by the SDN controller (representing a set of flows grouped by source/destination and priority), an `original_ether_type` field to restore the original EtherType at egress, and a 48-bit `sdn_latency` field that is initialized to zero and incremented at each hop by adding local queuing delay. The egress node removes the L2.5 header and forwards the original packet outside the SDN domain.

The SDN controller configures, for each monitored label, a latency threshold that is checked at the SDN egress node, which also acts as the Data Plane Active Collector (DPAC). When a packet exceeds the configured threshold, the DPAC immediately generates an INC packet derived from the violating packet and sends it back to the ingress node. The INC message includes the monitored `sdn_label` and an `output_port` field (9 bits) indicating the port to be used for rerouting. The ingress node uses the received label to identify the affected flow class and updates its forwarding state in the data plane, enabling fast rerouting without waiting for controller interaction. After rerouting is applied, the DPAC informs the SDN controller so that the controller can update its view and recompute routing if needed.

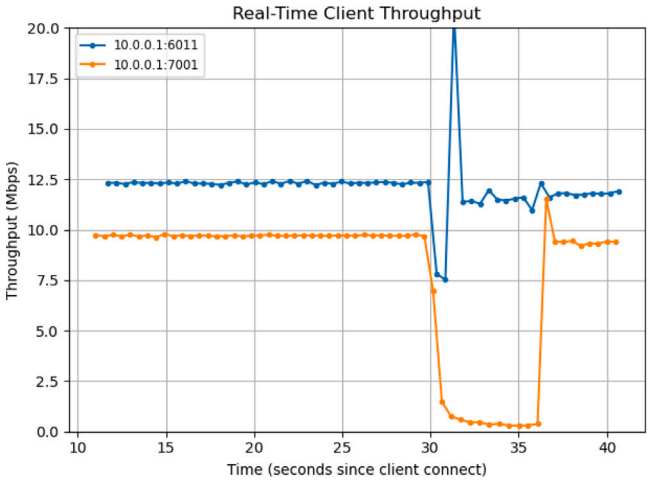
Tests and results

The test is executed on the AR traffic and involves INC, the Telemetry Collector/DPAC. A service instance is assumed already deployed, with traffic flowing through the SDN network. A Spirent traffic generator is used to inject traffic spikes and induce congestion.

The throughput measurements in Fig. 21 compare two iterations: one using in-band control and one relying on controller-based batch processing of telemetry as a baseline for comparison. Under congestion on the main path, the in-band control iteration preserves the throughput of the monitored low-latency flow, while the best-effort flow degrades as expected. In contrast, the baseline scenario processes telemetry at the controller, leading to longer reconfiguration times (hundreds of milliseconds). This results in visible degradation of the monitored flow before recovery occurs.



(a) Monitored Flow (Blue) INC rerouted



(b) Monitored Flow (Blue) rerouted by SDN Controller

Fig. 21. Comparison of congestion and reroute effect on a monitored flow (in blue) against non-monitored flow (in orange). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 20 details the timing of the in-band control reaction. In the reported trace, a packet carrying a cumulative SDN hop latency of 6302 µs triggers the DPAC threshold mechanism, which generates an INC control packet after approximately 0.58 ms. Due to buffering effects, packets already queued at the ingress node continue to arrive for a short interval; the queue is observed to empty after approximately 22 ms.

A packet-level capture (Fig. 22) shows that the time between the first INC message generation at the DPAC and the appearance of recovered packets on the backup-path interface is below 1.5 ms. A second trace interpretation reports approximately 1.78 ms from violation detection to traffic being observed on the backup path. Overall, the results confirm sub-2 ms recovery for the data-plane reroute in the software-switch setup; the same mechanism is expected to complete in tens of microseconds on hardware switches.

7.6. Workflow 4: IML-driven elasticity under compute saturation

To evaluate infrastructure-level assurance, a CPU overload was induced on the Jetson Orin running the AR2 serverless function. The pervasive Kafka monitoring service detected resource saturation and predicted SLA violation. The IML triggered migration of the AR2 function to a different node, while traffic steering rules were updated via the P4 switches.

To support proactive elasticity, the Kafka monitoring framework feeds a forecasting engine that predicts the evolution of resource utilization based on recent telemetry samples. Fig. 23 shows the predicted CPU load compared with the actual measured utilization during the experiment. The forecasting model anticipates the saturation trend several steps ahead, allowing the control loop to trigger mitigation before a hard SLA violation occurs. This predictive capability enables the IML to initiate migration and traffic steering in advance of full resource exhaustion, reducing the risk of frame drops and latency spikes.

Table 5
W4 quantitative timing results.

Measurement	Value
Metric sampling granularity	1 s
Forecast update interval	6 s
End-to-end elasticity reaction time	215 ms
Forecast inference latency (median, GPU)	44 ms
Forecast inference variability (std.)	<1.5 ms

Table 5 summarizes the quantitative measurements collected during Workflow 4. The reported values correspond to measurements collected on the integrated physical demonstrator under the considered test conditions. The monitoring subsystem operates at a 1 s granularity, while forecasting is performed every 6 s, providing a continuous yet computationally lightweight prediction cycle. The forecasting engine exhibits highly stable inference behavior, with a median latency of 44 ms and negligible variability (standard deviation below 1.5 ms), confirming that predictive analytics can be executed at the edge without introducing timing jitter. Most importantly, the closed-loop elasticity reaction time, measured from alarm generation to scaling enforcement, is 215 ms. This interval includes alarm reception, decision computation, and control-plane enforcement through NFR flow rule installation. The sub-second reaction demonstrates that predictive monitoring enables proactive mitigation before full resource exhaustion occurs. Combined with dataplane traffic steering, these results confirm that the monitoring–forecasting–IML chain can sustain latency-critical AR workloads under compute stress while maintaining service continuity.

Without the DESIRE6G IML and its forecasting capabilities, scaling would be reactive rather than proactive, preventing hitless (make-before-break) service migration. As a result, service degradation would occur before scaling is triggered, leading to temporary SLA violations.

7.7. Workflow 5: MAS-driven recovery under RAN latency anomalies

Workflow 5 evaluates the third layer of the DESIRE6G hierarchical assurance architecture, where service degradation occurring inside the RAN segment is detected through end-to-end telemetry correlation and mitigated via MAS-driven cross-domain reconfiguration. The objective is to validate latency anomaly detection in the RAN midhaul and demonstrate autonomous recovery through RIC-assisted handover.

MAS agents implementation

The distributed collaborative Multi-Agent System (MAS) is composed of autonomous agents deployed at selected network nodes. The implementation relies on a modular software architecture where components communicate through a Redis-based pub/sub messaging bus, enabling lightweight and near-real-time inter-process and inter-agent communication. Each agent is structured into functional modules: (i) telemetry collection interfaces, (ii) telemetry processing and aggregation, (iii) state construction and optional short-term prediction, (iv) a Deep Reinforcement Learning (DRL) decision engine, and (v) actuation interfaces toward the data plane. External systems are integrated through dedicated interface components that translate telemetry

No.	Time	Source	Destination	Protocol
174	-0.001784480	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INT
175	-0.001567896	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INT
176	-0.001470084	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
177	-0.001251432	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
272	-0.001060293	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
178	-0.001039283	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
273	-0.000968305	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
274	-0.000842913	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INC
1	0.000000000	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INT
2	0.000637267	00:00:00_00:00:01	00:00:00_00:00:04	SDN_INT

Fig. 22. INC results: detection and recovery time details.

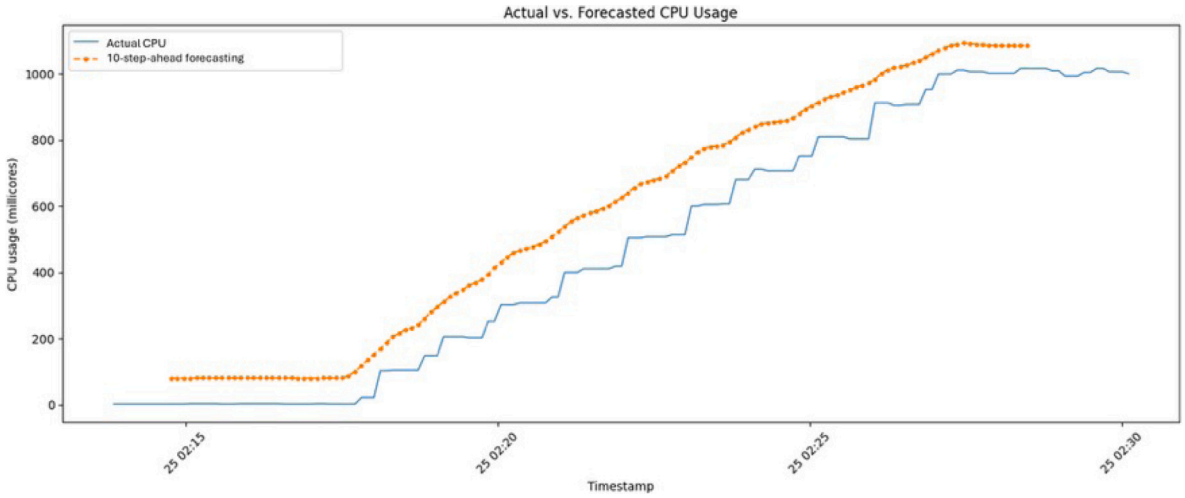


Fig. 23. Actual vs. forecasted CPU utilization of the AR2 pod. The forecasting engine anticipates the saturation trend, enabling proactive IML scaling.

streams into internal Redis messages, ensuring loose coupling between the MAS logic and the underlying infrastructure.

Telemetry is collected from a two-stage P4 collector (end-to-end and domain-splitted delay metrics), and from the RAN Intelligent Controller (i.e., radio indicators such as radio block usage and signal quality). These measurements are aggregated into a state vector and provided to the DRL module. The DRL agent formulates routing selection as a cost-minimization problem, where significant penalties are applied when the maximum one-way delay exceeds the SLA threshold. Based on the observed state, the DRL policy selects either the routing configuration to be enforced on the ingress P4 switch via a control interface (e.g., P4Runtime API) or the RAN Handover action through dedicated xAPP procedure.

RAN-internal delay anomaly creation

A programmable P4 switch is inserted in the RAN midhaul path between the DUs and the CU. This switch implements a custom *User Plane Delay Tuner*, a P4 network function capable of injecting deterministic latency into user-plane flows while leaving control-plane traffic unaffected. Flow classification is performed via subnet-based rules, ensuring that only user-plane traffic is delayed. Since the injected delay occurs within the RAN transport segment and does not originate from DU/CU container overload, the RIC does not directly detect the anomaly via native metrics. Instead, latency accumulation is captured through UE-level In-Band Telemetry (UE-INT) metadata embedded at the drone and updated along the path. INT metadata include timestamps collected at the drone, at the first and at the second Tofino instance, thus enabling the computation of RAN, SDN and end-to-end segment latencies.

Telemetry reports are collected by the Telemetry Collector at Site 2 and forwarded to MAS agents. The MAS correlates per-segment measurements to localize the anomaly domain. Upon confirming that the degradation originates within the RAN midhaul segment, the MAS triggers a corrective action via the RIC xApp interface.

The corrective mechanism consists of initiating a handover procedure from DU1–RU1 to DU2–RU2, bypassing the impaired midhaul path.

Tests and results

The experiment is conducted with the AR service active and stable under normal latency conditions. The Delay Tuner is then activated to inject controlled additional delay in the RAN midhaul user-plane traffic.

The following metrics are collected: per-segment latency (RAN, SDN, end-to-end) via INT, MAS anomaly detection time, RIC handover execution time, End-to-end service recovery time.

Table 6 summarizes the mean timing contributions of each step measured separately in the reconfiguration loop plus/minus two standard deviations from the mean value for that step. The combined time for telemetry processing, MAS detection, reaction generation, and P4 enforcement remains well below 100 ms. INT collection and aggregation are performed over groups of four packets. The time required to send the collector report is on average around 220 μs, while the INT server requires approximately (10 ± 1) ms to process and forward aggregated reports to the MAS. For RAN anomalies induced via the Delay Tuner, the DRL-based MAS anomaly detection time also remains around 10 ms (see MAS logs in Fig. 24), with an overall reaction time below 29 ms, including the communication overhead with the

```
13:06:28.319 Received QoS measurement: {'flow_id': '10', 'paths': {'p1': {'delay': {'max': 5.3, 'min': 4.0, 'avg': 1
00000000000.0}, 'jitter': {'max': 1.1, 'min': 0.2, 'avg': 0.8}}, 'p2': {'delay': {'max': 10.2, 'min': 2.0, 'avg': 7.
5}, 'jitter': {'max': 2.1, 'min': 0.3, 'avg': 1.1}}}, 'init_time': 1763471188.308243}
13:06:28.322 QoS measurement processed
13:06:28.329 Anomaly detected, sending petition to handover the UE
```

Fig. 24. MAS logs: DRL-based anomaly detection time.

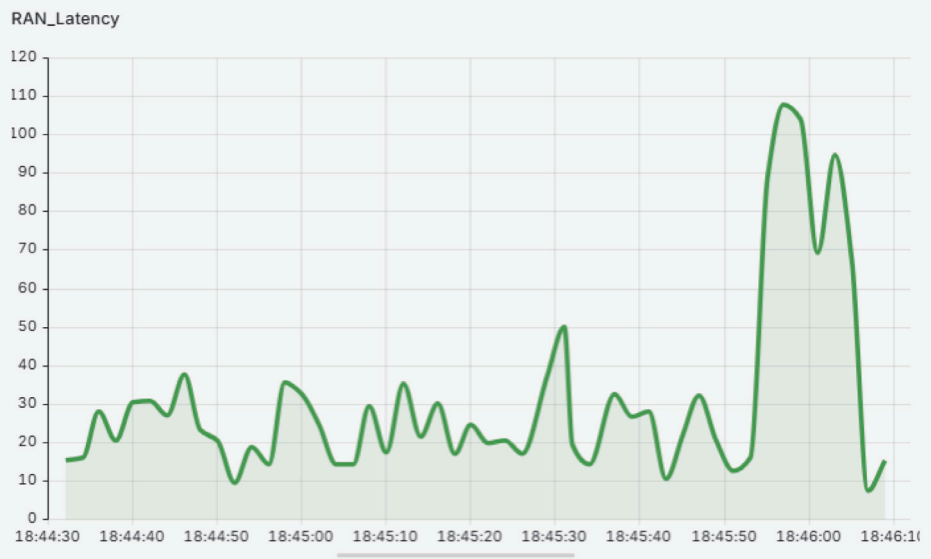


Fig. 25. RAN latency time series collected by MAS during the experiment.

xAPP. The observed MAS behavior is coherent with previous studies comparing DRL-assisted adaptation against threshold-based reactive heuristics [25], where reactive approaches suffered from increased delay and traffic degradation due to their delayed response to traffic variations, while DRL-based policies better adapted to dynamic conditions. These results confirm that anomaly localization and corrective decision generation operate at near-real-time scale for both domains.

The full mitigation requiring handover from DU1 to DU2, the overall recovery time is approximately 8 s. As shown in Fig. 25, steady-state RAN latency during the flying mission is around 15 ms, with some spikes due to radio dynamics. When the Delay Tuner injects additional latency, RAN segment delay increases up to approximately 100 ms. After the MAS triggers handover through the RIC xApp, latency returns to nominal values. The measured 8 s recovery time is primarily due to RIC/xApp execution and UE synchronization procedures. The DESIRE6G platform complete detection and handover request generation within less than 50 ms. The remaining delay is attributable to RAN internal processing and UE radio-level handover synchronization.

Without the MAS layer, the system reduces to domain-specific control as in Workflow 3, where recovery relies solely on SDN rerouting, since the RAN does not have any native mechanism to monitor internal delay. This leads to slower reactions and incomplete mitigation, as RAN-induced latency cannot be addressed, resulting in persistent end-to-end performance degradation.

7.8. Discussion on DESIRE6G control loops versus baseline loops

To better highlight the impact of the proposed architecture, we compare the behavior of the system with and without the DESIRE6G control loops. The baseline corresponds to a traditional setup where only domain-specific control mechanisms are available, such as SDN-based routing or static resource allocation, without hierarchical coordination or predictive capabilities.

Table 7 summarizes the observed differences across the main workflows. In Workflow 3, baseline operation relies on centralized SDN rerouting, which enables recovery but with slower reaction times and limited visibility of end-to-end conditions. In contrast, the DESIRE6G approach leverages in-band telemetry and data-plane mechanisms to achieve fast and localized reactions, fully avoiding latency spikes, and significantly reducing recovery time.

Table 6
Workflow 5 — Closed-loop timing breakdown of DESIRE6G components (milliseconds)

Component	Time (ms)
Collector Summary	0.22 ± 0.04
INT Server Processing	10 ± 1
MAS Detection	10 ± 0.75
MAS Reaction and xApp communication	25 ± 4
Total Loop Time	45 ± 5.79

In Workflow 4, the absence of the IML and its forecasting capabilities leads to purely reactive scaling, where resource adaptation is triggered only after service degradation occurs. This prevents hitless (make-before-break) scaling and may introduce transient SLA violations. The DESIRE6G approach enables predictive scaling and traffic redirection, ensuring service continuity during resource reallocation.

In Workflow 5, the lack of the MAS layer limits the system to domain-specific control, similar to Workflow 3. In this case, SDN-based rerouting may partially mitigate network congestion, but cannot address RAN-induced latency. As a result, end-to-end performance degradation may persist. By contrast, the DESIRE6G architecture enables cross-domain coordination through MAS and RIC interaction, allowing the system to restore latency to nominal levels.

7.9. Discussion on scalability and deployment limits.

The experimental validation presented in this paper focuses on a limited-scale but fully integrated deployment, and therefore does not provide a large-scale scalability characterization in terms of increasing

Table 7
Comparison of baseline behavior and DESIRE6G control loops.

Workflow	Baseline	DESIRE6G
W3	SDN rerouting, slower recovery, latency peaks not mitigated (centralized control)	Sub-ms data-plane reaction, extremely fast recovery with no SDN involvement
W4	Reactive scaling, SLA violations during overload	Predictive hitless scaling, no service disruption, fast IML-based recovery
W5	No RAN control, persistent latency degradation, SDN rerouting	Anomaly detection and domain localization, Cross-domain recovery (MAS+RIC), la7.8tency restored

numbers of gNBs, DESIRE6G sites, or concurrent services. Nevertheless, the proposed architecture was explicitly designed to improve scalability through hierarchical decomposition of monitoring and control functions. In particular, fast and localized events are handled at the lowest possible layer: INC mitigates transport impairments directly in the programmable data plane, the IML performs site-local recovery actions for compute-related degradations, and the MAS is activated only for cross-domain anomalies requiring coordinated decisions across RAN, transport, and edge segments. This hierarchy reduces the load on centralized components and limits the amount of control traffic that must be propagated across the full system.

When scaling to a larger number of gNBs or sites, the main expected challenges are related to telemetry collection and aggregation, coordination overhead among distributed MAS agents, and the consistency of reconfiguration actions across multiple domains. As the number of monitored paths, radio segments, and candidate recovery actions increases, both the volume of telemetry and the complexity of global coordination can grow significantly. In addition, a larger deployment may require stronger partitioning of telemetry processing, more selective triggering of MAS actions, and careful placement of collectors and control functions to avoid introducing bottlenecks at the orchestration layer. These considerations are coherent with the role separation already implemented in DESIRE6G, where telemetry is collected locally, site-specific actions are handled by local IML instances, and only higher-level anomalies are escalated to cross-domain control loops.

Overall, while a full scalability analysis is outside the scope of the present demonstrator, the proposed architecture provides a modular basis for scaling because it avoids relying on a single monolithic control loop for all impairments.

8. Conclusions

This paper presented the first integrated demonstration of the DESIRE6G architecture for real-time, latency-critical services, validated through a distributed Augmented Reality surveillance application combining UAV sensing, edge processing, and immersive visualization. The demonstrator deployed a complete 6G-ready framework across radio, SDN transport, edge computing, and orchestration domains, leveraging cloud-native principles, programmable data planes, pervasive telemetry, and secure multi-agent control.

Results showed that the system maintains end-to-end latency within 15–25 ms during steady-state operation and provides hierarchical service assurance through three complementary mechanisms: (i) *in-band control* for microsecond-level congestion mitigation, (ii) *IML-driven elasticity* for sub-second migration and scaling, and (iii) *MAS-driven optimization* for cross-domain root-cause identification and recovery. The demonstrator confirmed that telemetry-enriched data paths do not compromise real-time performance, and that the combination of programmable switches and AI-assisted orchestration enables robust, self-adaptive behavior under network or compute perturbations.

Beyond the specific XR use case, the architecture generalizes to a wide range of mission-critical applications such as industrial automation, cooperative robotics, and remote operation of autonomous systems. The integration of UE-side telemetry, SDN-based in-network control, O-RAN intelligence, and Kubernetes orchestration illustrates a viable pathway toward self-managing 6G infrastructures capable of enforcing demanding service level agreements.

This work contributes primarily at the system level by demonstrating how existing technologies can be orchestrated into a coherent, hierarchical service assurance framework. The novelty lies in the integration of multi-layer control loops driven by in-band telemetry and their validation in a real end-to-end deployment, providing practical evidence of how cross-domain adaptation can be achieved under strict latency constraints.

These results provide a reference design for future 6G systems, highlighting how coordination across data plane, infrastructure, and AI-driven control layers can enable resilient and latency-aware services beyond current 5G capabilities.

Future work will expand the orchestration logic to multi-tenant environments, support finer-grained serverless function chaining, and integrate additional rApps/xApps for RAN-layer analytics. It will also investigate more explicit coordination and conflict-mitigation mechanisms across the different optimization layers, particularly in scenarios with overlapping control objectives across RAN, transport, and edge domains. Large-scale deployment across geographically distributed sites and multi-operator environments will further evaluate scalability and cross-domain coordination. Finally, extending telemetry to include energy, reliability, and semantic-level indicators will unlock new optimization strategies for sustainable, intent-driven 6G networks.

CRediT authorship contribution statement

Francesco Paolucci: Writing – review & editing, Methodology, Conceptualization. **Emilio Paolini:** Writing – original draft, Conceptualization. **Faris Alhamed:** Writing – original draft, Methodology, Conceptualization. **Massimo Satler:** Writing – original draft, Methodology, Conceptualization. **Domenico Uomo:** Writing – original draft, Methodology, Conceptualization. **Michelangelo Guaitolini:** Writing – original draft, Methodology, Conceptualization. **Pol Gonzalez:** Writing – original draft, Methodology, Conceptualization. **Marc Ruiz:** Writing – original draft, Methodology, Conceptualization. **Luis Velasco:** Writing – original draft, Methodology, Conceptualization. **Sandor Laki:** Writing – original draft, Methodology, Conceptualization. **David Kis:** Writing – original draft, Methodology, Conceptualization. **Gergely Pongracz:** Writing – original draft, Methodology, Conceptualization. **Attila Mihaly:** Writing – original draft, Methodology, Conceptualization. **Anestis Dalgkitsis:** Writing – original draft, Methodology, Conceptualization. **Chrysa Papagianni:** Writing – original draft, Methodology, Conceptualization. **Anastassios Nanos:** Writing – original draft, Methodology, Conceptualization. **Stephen Parker:** Writing – original draft, Methodology, Conceptualization. **Vincent Lefebvre:** Writing – original draft, Methodology, Conceptualization. **Mark Angoustures:** Writing – original draft, Methodology, Conceptualization. **Juan Jose Vegas Olmos:** Writing – original draft, Methodology, Conceptualization. **Andrea Sgambelluri:** Writing – original draft, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The research leading to these results has received funding from the Smart Networks and Services Joint Undertaking under the European Union's Horizon Europe research and innovation programme under G.A. No. 101096466 (DESIRE6G).

Data availability

Data will be made available on request.

References

[1] W. Saad, M. Bennis, M. Chen, A vision of 6G wireless systems: Applications, trends, technologies, and open research problems, IEEE Netw. 34 (3) (2019) 134–142.
[2] G.P. Fettweis, The tactile internet: Applications and challenges, IEEE Veh. Technol. Mag. 9 (1) (2014) 64–70.
[3] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, M. Zorzi, Toward 6G networks: Use cases and technologies, IEEE Commun. Mag. 58 (3) (2020) 55–61.
[4] M. Polese, L. Bonati, S. D'oro, S. Basagni, T. Melodia, Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges, IEEE Commun. Surv. & Tutorials 25 (2) (2023) 1376–1411.
[5] D. Kreutz, F.M. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2014) 14–76.
[6] A. Agrawal, C. Kim, Intel tofino2–a 12.9 tbps p4-programmable ethernet switch, in: 2020 IEEE Hot Chips 32 Symposium, HCS, IEEE, 2020, pp. 1–32.
[7] F. Alhamed, D. Scano, P. Castoldi, J.J.V. Olmos, I. Vershkov, F. Paolucci, F. Cugini, P4 telemetry collector, Comput. Netw. 227 (2023) 109727.
[8] K. Senjab, S. Abbas, N. Ahmed, A.U.R. Khan, A survey of Kubernetes scheduling algorithms, J. Cloud Comput. 12 (1) (2023) 87.
[9] F. Paolucci, M. Guaitolini, A. Sgambelluri, F. Alhamed, D. Uomo, E. Paolini, M. Satler, P. Gonzalez, M. Ruiz, L. Velasco, et al., Augmented reality in the DESIRE6G cloud-native and programmable infrastructure with multi-agent system and pervasive monitoring, in: 2025 IEEE Conference on Standards for Communications and Networking, CSCN, IEEE, 2025, pp. 1–3.
[10] D. Scano, A. Giorgetti, F. Paolucci, A. Sgambelluri, J. Chammanara, J. Rothman, M. Al-Bado, E. Marx, S. Ahearne, F. Cugini, Enabling P4 network telemetry in edge micro data centers with kubernetes orchestration, IEEE Access 11 (2023) 22637–22653.
[11] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L.J. Wobker, et al., In-band network telemetry via programmable dataplanes, in: ACM SIGCOMM, vol. 15, 2015, pp. 1–2.
[12] M.S. Elbamby, C. Perfecto, M. Bennis, K. Doppler, Toward low-latency and ultra-reliable virtual reality, IEEE Netw. 32 (2) (2018) 78–84.
[13] K. Koutlia, S. Lagén, On the impact of open RAN fronthaul control in scenarios with XR traffic, Comput. Netw. 253 (2024) 110722.
[14] V. Ziegler, H. Viswanathan, H. Flinck, M. Hoffmann, V. Räisänen, K. Hätönen, 6G architecture to connect the worlds, IEEE Access 8 (2020) 173508–173520.
[15] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration, IEEE Commun. Surv. & Tutorials 19 (3) (2017) 1657–1681.
[16] M. Doostmohammadian, A. Aghasi, M. Pirani, E. Nekouei, H. Zarrabi, R. Keypour, A.I. Rikos, K.H. Johansson, Survey of distributed algorithms for resource allocation over multi-agent systems, Annu. Rev. Control. 59 (2025) 100983.
[17] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, et al., Illixr: An open testbed to enable extended reality systems research, IEEE Micro 42 (4) (2022) 97–106.
[18] A. Paradise, S. Surve, J.C. Menezes, M. Gupta, V. Bisht, K.R. Jang, C. Liu, S. Qiu, J. Dong, J. Shin, et al., RealTHASC—a cyber-physical XR testbed for AI-supported real-time human autonomous systems collaborations, Front. Virtual Real. 4 (2023) 1210211.
[19] F. Alhamed, A. Sgambelluri, C. Papagianni, F. Paolucci, In-band collection and control of end-to-end latency in programmable packet-optical networks, in: Optical Fiber Communication Conference (OFC) 2025, Optica Publishing Group, 2025, p. M3H.4,
[20] G. Pongrácz, A. Mihály, I. Gódor, S. Laki, A. Nanos, C. Papagianni, Towards extreme network KPIs with programmability in 6G, in: Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, MobiHoc '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 340–345.
[21] L. Valcarenghi, P. Castoldi, A. Sgambelluri, E. Paolini, A. Pacini, A flexible forecasting platform enabling zero touch networking and digital twinning, in: 2023 23rd International Conference on Transparent Optical Networks, ICTON, IEEE, 2023, pp. 1–4.
[22] K. Lad, M.A.A. Dewan, F. Lin, Trust management for multi-agent systems using smart contracts, in: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), IEEE, 2020, pp. 414–419.
[23] P. Gonzalez, F. Alhamed, H. Shakespear-Miles, S. Barzegar, F. Paolucci, A. Sgambelluri, J.J. Vegas Olmos, M. Ruiz, L. Velasco, Near-real-time 6G service operation enabled by distributed intelligence and in-band telemetry, J. Opt. Commun. Netw. 17 (3) (2025) A247–A258.
[24] S. Barzegar, M. Ruiz, L. Velasco, Autonomous flow routing for near real-time quality of service assurance, IEEE Trans. Netw. Serv. Manag. 21 (2) (2024) 2504–2514.
[25] L. Velasco, S. Barzegar, F. Tabatabaeimehr, M. Ruiz, Intent-based networking and its application to optical networks [invited tutorial], J. Opt. Commun. Netw. 14 (1) (2022) A11–A22.
[26] A. Dalgkitsis, C. Hsu, C. Papagianni, P. Grosso, C. de Laat, LLM-based optimization algorithm selection for high-performance networks orchestration, in: Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC Workshops '25, Association for Computing Machinery, New York, NY, USA, 2025, pp. 854–859.
[27] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788.
[28] B. Koonce, ResNet 50, in: Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization, Springer, 2021, pp. 63–72.
[29] Y. Zhou, K. Yang, Exploring tensorrt to improve real-time inference for deep learning, in: 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), IEEE, 2022, pp. 2011–2018.
[30] A.A. Hagberg, D.A. Schult, P.J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, CA, USA, 2008.